On the Formalization and Analysis of a Spatio-Temporal Role-Based Access Control Model

Manachai Toahchoodee and Indrakshi Ray
Department of Computer Science
Colorado State University
{toahchoo, iray}@cs.colostate.edu

Abstract

With the growing use of wireless networks and mobile devices, we are moving towards an era of pervasive computing. Such environments will spawn new applications that use contextual information to provide enhanced services. Traditional access control models cannot protect such applications because the access requirements may be contingent upon the location of the user and the time of access. Consequently, we propose a new spatio-temporal role-based access control model that supports delegation for use in such applications. The model can be used by any application where the access is contingent not only on the role of the user, but also on the locations of the user and the object and the time of access. We describe how each entity in the role-based access control model is affected by time and location and propose constraints to express this. We also show how the formal semantics of our model can be expressed using graph-theoretic notation. The various features of our model give rise to numerous constraints that may interact with each other and result in conflicts. Thus, for any given application using our model, it is important to analyze the interaction of constraints to ensure that conflicts or security breaches do not occur. Manual analysis is tedious and error-prone. Towards this end, we show how the analysis can be automated using Coloured Petri Nets. Since automated anal-

ysis for large applications is time consuming, we propose an approach that reduces the analysis time.

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 [14] and CMU's Aura [17], that will make life easier for people. Pervasive computing applications introduce new security issues that cannot be addressed by existing access control models and mechanisms. For instance, access to a computer should be automatically disabled when a user walks out of the room. Traditional models, such as Discretionary Access Control (DAC) or Role-Based Access Control (RBAC), do not take into account such environmental factors in making access decisions. Consequently, access control models are needed that use environmental factors, such as, time and location, while determining access.

Researchers have proposed various models that use contextual information, such as, location and time, for performing access control [1, 9, 10, 11, 14, 17, 24, 28, 33, 34, 36, 39]. Many of these were developed with commercial applications in mind and are based on RBAC. Examples include TRBAC [9], Geo-RBAC [10], Geotemporal RBAC [2, 3], and STRBAC [36]. These models are more expressive than their traditional counterparts, and have various features which the users can selectively use based on the application requirements. The different features of these models interact in subtle ways resulting in inconsistencies and conflicts. Consequently, it is important to formalize the models, analyze and understand their semantics, before they can be widely deployed.

We propose a new spatio-temporal access control model supporting delegation that improves upon many of the previous works. It can be used by any application where access is contingent upon the role of the user, the locations of the user and the object, and the time of access. Since RBAC simplifies role-management and is widely used in commercial organizations, we base our

work on it. Access control provided by our model depends not only on the role of the user but also on the spatio-temporal constraints associated with the various entities. The different features are expressed using formal predicates that must be satisfied by applications using our model. The formal semantics of the model are defined using a graph-theoretic notation. The authorization requirements of an application can be represented using this notation, which we term as its *access control graph*.

Various features of the model may interact in subtle ways resulting in inconsistencies and conflicts. The access control constraints of an application using our model must be analyzed to ensure that such problems, which, in turn, may cause security breaches, do not occur. Manual analysis of the access control specifications of complex, real-world applications is tedious and error-prone. Towards this end, we propose the use of Coloured Petri Net (CPN) [19, 22, 27] for doing automated analysis. A number of reasons motivated this choice. First, CPN has a well-defined semantics that allows us to unambiguously define the behavior of the model. Second, the language supported by CPN is expressive enough to specify various kinds of systems. Third, CPN offers interactive simulations that can be used for studying the behavior of the system. Fourth, CPN provides tool support for graphically representing the model and for performing simulation and formal analysis [22]. Fifth, it has been used successfully for the verification of real-time concurrent systems.

Since access control specification of a real-world system may be very complex, the CPN analysis may take a significant amount of time. Towards this end, we show how to improve the performance by transforming the access control graph to a more condensed version, which we term the *privilege acquisition graph*. Although the privilege acquisition graph contains less information than the corresponding access control graph, it is adequate for checking some problems with the access control specification. Once a problem has been found, the source of the problem can be identified by analyzing a sub-graph of the access control graph.

The rest of the paper is organized as follows. Section 2 describes the related work. Section

3 presents our spatio-temporal access control model. Section 4 uses graph-theoretic notations to provide a formal semantics of the model and shows how the access control requirements of a real-world dengue decision support system can be formalized using our approach. Section 5 discusses how the access control specification of the application can be analyzed using CPNs. Section 6 provides an approach for speeding up the analysis. Section 7 concludes the paper with some pointers to future directions.

2 Related Work

Location-based access control has been addressed in works not pertaining to RBAC [1, 2, 3, 17, 18, 28, 31, 33, 48]. Atluri and Chun proposed the Geospatial Data Authorization Model (GSAM) which is an authorization model for managing geospatial information [2, 3]. The requester can get access to geospatial information provided his credentials and time of access matches the credential and temporal expressions defined in the authorization policy. Ardagna et al. [1] present the Location-Based Access Control (LBAC) model where the access is contingent upon the location information of the user and his credentials. Yu et al. [48] proposed LTAM, a location-temporal authorization model, which focuses on controlling user access to the different locations. Pu et al. [31] present the context access control model, called CACM, which integrates the context information to the $UCON_{ABC}$ usage control model. Context Sensitive Access Control (CSAC) proposed by Hulsebosch et al. [18] focus on using context information such as time, location, velocity to control the accessibility of service while preserving the privacy of user information. Hengartner et al. [17] discuss how location information pertaining to a user can be securely accessed.

Our work is based on RBAC [16] 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 [6, 7, 8, 15, 52]. Researchers have also worked on extending RBAC to support pervasive computing applica-

tions [13, 14, 38]. Sampemane et al. [38] present a new access control model for active spaces which denote the computing environment integrating physical spaces and embedded computing software and hardware entities. Environmental aspects are adopted into the access control model for active spaces, and the space roles are introduced into the implementation of the access control model based on RBAC. Covington et al. [14] introduce environment roles in a generalized RBAC model (GRBAC) to help control access to private information and resources in ubiquitous computing applications. The environment roles differ from the subject roles in RBAC but do have similar properties including role activation, role hierarchy and separation of duty. Environment roles are also associated with permissions, and environment roles are activated when the environmental conditions change. In a subsequent work [13], 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. [9] that adds the time dimension to the RBAC model. The authors in this paper introduce the concept of role enabling and disabling. Temporal constraints determine when the roles can be enabled or disabled. A role can be activated only if it has been enabled. Joshi et al. [24] 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. In another work, Joshi and Bertino [25] extend the GTRBAC model to support fine-grained delegation. Although the authors support various forms of delegation, they do not discuss the effect of temporal constraints or the delegation chain in this paper. The formal analysis of the different types of time-based hybrid hierarchy, introduced in the earlier works [25, 24], is proposed by Joshi et al. in [26]. Here, the authors introduce the notion of *uniquely activable set* (UAS), which is a set of roles that can be activated by the user assigned to the senior-most role in the hierarchy. This information can be used by the system administrator to determine the access capabilities of a user within a session. The authors also define the notion of equivalence between different hierarchies

based on the permissions they possess. The transformation of hierarchy via the modification of role is analyzed in the latter part of the paper. The impact of separation of duty constraints on the role hierarchy are outside the scope of this work.

Researchers have also extended RBAC to incorporate spatial information [10, 34]. The GEO-RBAC model, proposed by Bertino et al. [10], makes role activation dependent on the location of the user. For instance, a user can acquire the role of teacher only when he is in the school. Outside the school, he can acquire the role of citizen. The model supports role hierarchies but does not deal with separation of duty constraints. Another work incorporating spatial information is by Ray et al. [34]. Here again, the authors show how each component of RBAC is influenced by location. The authors define their formal model using the Z specification language. Role hierarchy and separation of duty constraints are not addressed in this paper. None of these works discuss the impact of time on location.

Incorporating both time and location in RBAC has been addressed in several works [11, 12, 36, 39]. Chandran's work [11] combines the main features of GTRBAC and GEO-RBAC. Here again, role is enabled by time constraints. The user can activate the role if the role is enabled and the user satisfies the location constraints associated with role activation. Samuel et al. [39] propose GST-RBAC which provides a framework to incorporate topological spatial constraints to the existing GTRBAC model. The authors do this by augmenting GTRBAC operations with spatial constraints. The operations are allowed only if the spatial and temporal constraints are satisfied. The model also introduces the notion of Spatial Role Hierarchy and Spatial Separation of Duty (spSoD) constraints. Our early work also extends RBAC with spatial and temporal constraints [36]. Although the goal of this work is similar to that proposed by Chandran et al. and Samuel et al., our model can express some real-world constraints that are not possible in the other ones. Subsequently, we extend our model to support different types of spatio-temporal delegation [37]. The model allows user/role to grant privileges to the delegatee which could be either user or role. However, the model does not

support the transfer operation or the concept of delegation chain. Our current work eliminates this shortcoming. Chen and Crampton develop a graph based representation for spatio-temporal RBAC in [12]. The RBAC entities are represented by vertices while their relationships are represented by the edges of a directed graph. Each vertex and edge in the graph is associated with spatio-temporal constraints. The authors propose three types of models: standard, strong, and weak. In the standard model, component v_1 is said to be authorized to component v_n if all vertices along the authorization path satisfy the spatio-temporal constraints. In the strong model, component v_1 is said to be authorized to component v_n if all vertices together with the edges along the authorization path satisfy the spatio-temporal constraints. In the weak model, component v_1 is said to be authorized to component v_n if both vertices satisfy the spatio-temporal constraints. The major contribution of this work is that the authors provide a well-defined semantics for the three types of spatio-temporal RBAC. The authors do not discuss separation of duty or delegation constraints in this paper. Identifying how inconsistencies and conflicts can be detected in these models is also outside the scope of this work.

A lot of work also appears that attempt to analyze RBAC and its extensions. Some have used the Z specification language for specifying RBAC [49] and LRBAC [34]. Although Z language can represent RBAC and its constraints in a formal manner, it does not have tool support for automated verification. Others have used an extension of the Unified Modeling Language (UML) called parameterized UML to visualize the properties of RBAC constraints [35]. The model describes how one can visualize the conflicts that may occur with RBAC constraints. However, the approach is not automated.

Researchers have advocated the use of existing formal specification languages with tool support for modeling and verifying RBAC and its extensions. Specifically, Schaad et al. [41] and Zao et al. [50] describe how to analyze static properties of RBAC, such as, user-role assignment, permission-role assignment, role hierarchy and static separation of duty. Samuel et al. [39] also

illustrate how GST-RBAC can be specified in Alloy. They describe how the various GST-RBAC functionalities, that is, user-role assignment, role-permission assignment, and user-role activation, can be specified by Alloy. However, analyzing the interaction of the various features and identifying sources of potential conflicts are not addressed in this paper. Our recent works [45, 46] fill this gap. Specifically, one [45] shows how the numerous features in our spatio-temporal access control model can be represented in Alloy and how their interactions can be analyzed. In a subsequent work [46], we analyze a more complex model – a spatio-temporal RBAC with delegation – to identify the sources of conflicts. However, both these works focus on analyzing the model in isolation and identify potential conflicts that may occur due to feature interactions. Such analysis is independent of the application. However, when this model is to be used for any given application, we need to ensure that no inconsistencies occur with respect to the access control requirements of the given application. For example, we need to ensure that there are no users, roles, or permissions in the application that are not connected to other entities. We may also need to ensure that separation of duty constraints or delegation constraints are not violated in the application. In one of our recent work [47], we show how the access control requirement can be specified using the Unified Modeling Language (UML). The UML2Alloy tool transforms this model to an Alloy specification which is then automatically verified by the Alloy Analyzer.

Although Alloy supports automated analysis, it has limitations with respect to the types of verifications it can perform. For example, analyzing and understanding the behavior of the application using Alloy is non-trivial. Such analysis is needed for dynamic systems where we need to ensure that the system does not enter an undesirable state. Towards this end, researchers [5, 23, 27, 29, 32] have investigated alternate approaches, such as, Coloured Petri Nets (CPNs) [19, 22, 27] for automated analysis. CPN allows one to represent the model in a graphical language, has a well-defined semantics and has automated tools for doing simulation and verification. Rasmussen and Singh [32] show how CPN is used in designing the PRISMA C96 intruder alarm system. The interac-

tions of components were modeled and verified using CPN to detect if the configurations have any conflicts. CPN has also been used in access control model verification. Jiang et al. [23] develop a CPN model to verify the security properties of the Bell LaPadula (BLP) model. Laborde et al. [27] propose the use of CPN for analyzing the traditional RBAC-based policies of network security mechanisms. This work focuses on verifying confidentiality, integrity, availability, and filtering rules.

Lu et al. [29] show how access control properties of workflows can be verified using CPNs. Specifically, they describe how to formalize the control flow, authorization rules, and separation of duty constraints in a workflow in the presence of role activation hierarchy. The authors first show how to model each part (namely, control flow, authorization rules, and separation of duty) in isolation. Subsequently, the authors propose an approach for producing the integrated model which allows one to study the interactions of the parts, such as RBAC authorization policy with separation of duty constraints. Reachability analysis is used to detect conflicts between the features. The size of the integrated model increases exponentially when new entities are added. To prevent state explosion during reachability analysis, the authors introduce two rules for reducing the size of the model. The model analyzed by the authors does not support many features which are needed in workflow applications: permission inheritance hierarchy, separation of duty for permission-role assignment, and delegation. Atluri et al. [4] propose an authorization model to use for workflows. The model specifies constraints that allows authorized subjects to gain access on the required objects for the duration of the task. Subsequently, the authors extend this work to support task-based separation of duty constraints and show how this extended model can be specified using CPN [5]. The authors then show how to do a reachability analysis to check whether the given tasks can be executed in the presence of authorization constraints. Shafiq et al. [42] show how the various constraints of GTRBAC, such as, cardinality constraints, SoD constraints, and role hierarchy can be modeled using CPN. The reachability analysis reveals the presence of infeasible paths where an en-

tity cannot invoke the privileges assigned to him. However, analyzing the interaction of constraints is not discussed in these works.

3 Our Model

3.1 Representing Location and Time

Representing Location

In order to perform location-based access control, we need to formalize the concept of location [10, 11] and propose the location comparison operators that are used in our model. 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 is formally defined by a set of points in a three-dimensional geometric space. 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. The granularity of each co-ordinate is dependent upon the application. 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 a mapping function *m* that converts a logical location to a physical one.

Definition 1

[Mapping Function *m***]** *m* is a total function that converts a logical location into a physical one. Formally, $m : L \rightarrow P$, where P is the set of all possible physical locations and L is the set of all logical locations.

Different kinds of operations can be performed on location data. We define two binary operators, namely, *containment* $\subseteq$, and *equality* $=$, that we use in this paper. 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 follow-

ing 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. Intuitively, a physical location $ploc_j$ is contained in another physical location $ploc_k$, if all points in $ploc_j$ also belong to $ploc_k$. Two physical locations $ploc_r$ and $ploc_s$ are *equal* if $ploc_r \subseteq ploc_s$ and $ploc_s \subseteq ploc_r$. Note that these operators are defined on physical locations. Thus, logical locations must be transformed into physical locations (using mapping function *m* defined above) before we can apply these operators. We define a logical location called *Universe* that contains all other locations.

Representing Time

Our model uses two kinds of temporal information. The first is known as time instant and the other is time interval. A *time instant* is one discrete point on the time line. The exact granularity of a time instant is 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 instants. We use the notation $t_i \in d$ to mean that t_i is a time instant in the time interval d. Here again, we define operators containment $\subseteq$ and equality $=$ for operating on time intervals. A time interval d_j is said to be *contained in* another time interval d_k, denoted as, $d_j \subseteq d_k$, if the following condition holds: $\forall t_i \in d_j, t_i \in d_k$. The interval d_j is called the contained interval and d_k is referred to as the containing or the enclosing interval. Two time intervals d_s and d_r are said to be equal if $d_r \subseteq d_s$ and $d_s \subseteq d_r$. We define a time interval called *Always* that includes all other time intervals.

Representing Time and Location as Spatio-Temporal Points

In order to simplify our presentation, we use the concept of spatio-temporal points to represent time and location. A spatio-temporal point is represented as a pair of the form (d, l) where d represents the temporal component and l represents the spatial one. Note that, d and l represent time interval

and location respectively. We say that a spatio-temporal point (d,l) is contained in another (d',l'), denoted by $(d,l) \subseteq (d',l')$ iff $(d \subseteq d') \wedge (l \subseteq l')$. The union of two spatio-temporal points, denoted as $(d,l) \cup (d',l')$, is given by $(d,l) \cup (d',l') = (d \cup d', l \cup l')$. The intersection of two spatio-temporal points, denoted as $(d,l) \cap (d',l')$, is given by $(d,l) \cap (d',l') = (d \cap d', l \cap l')$.

3.2 Relationship of Core-RBAC Entities and Relationships with Time and Location

We begin by describing how the entities in RBAC, namely, *Users*, *Roles*, *Sessions*, *Permissions*, and *Objects*, are associated with location and time.

Users

We assume that each valid user, interested in doing some location-sensitive operation, carries a locating device that 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, the following constraint must be true. Note that, in this and all the subsequent formulae, we omit the quantification symbols.

$$(UserLocation(u,t) = l_i) \wedge (UserLocation(u,t) = l_j) \Leftrightarrow (l_i \subseteq l_j) \vee (l_j \subseteq l_i)$$

We define a similar function $UserLocation(u,d)$ 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. 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 *ObjLocation(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 on-campus student role only when he is in the campus during the semester. 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. $RoleAllocTimeLoc(r)$ gives the set of spatio-temporal points where the role can be allocated.

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.

$$UserRoleAssign(u,r,d,l) \Rightarrow (UserLocation(u,d) = l) \wedge ((d,l) \subseteq RoleAllocTimeLoc(r))$$

Some roles can be activated only if the user is in some specific locations at given times. 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* [9, 24] 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. $RoleEnableTimeLoc(r)$ gives the set of spatio-temporal points where role *r* can be activated.

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 and the duration of role activation must be a subset of the allowable spatio-temporal points for the activated role and the role can be activated only if it is assigned.

$$UserRoleActivate(u,r,d,l) \Rightarrow ((d,l) \subseteq RoleEnableTimeLoc(r)) \wedge UserRoleAssign(u,r,d,l)$$

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 time-dependent, location-dependent, or both. 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 and duration. The allowable spatio-temporal points where a session of type *st* can be created is denoted by $SessionTimeLoc(st)$.

When a user *u* wants to create a session *s*, the session duration *d* and the location of the user *l* must be contained within the spatio-temporal points associated with the session. The predicate $SessionUser(u,s,d,l)$ indicates that a user *u* has initiated a session *s* for duration *d* at location *l*.

$$SessionUser(u,s,d,l) \Rightarrow (d,l) \subseteq SessionTimeLoc(Type(s))$$

Since sessions are associated with time and locations, not all roles can be activated within some session. The predicate $SessionRoles(u,r,s,d,l)$ states that user *u* initiates a session *s* and activates a

role r for duration d and at location l. This is possible only if user u can activate role r for duration d and at location l and the session can be created during the same time and at the same location.

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

Permissions

Our model allows us to specify 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 only if he is in the bank, the file location is the bank secure room, and the time of access is during the working hours. Our model should be capable of expressing such requirements.

Permissions are associated with roles, objects, and operations. We associate 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 spatio-temporal point (d,l) must be contained in the point where the role r can be enabled and where the permission p can be invoked by r.

$$PermRoleAcquire(p,r,d,l) \Rightarrow (d,l) \subseteq RoleEnableTimeLoc(r) \cap (PermDur(p) \times PermRoleLoc(p,r))$$

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 can be acquired

by role r during time d and at location l, user u can activate role r at the same time and location, and object location matches those specified in the permission.

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

3.3 Impact of Time and Location on Role-Hierarchy

The structure of an organization in terms of lines of authority can be modeled as a hierarchy. This organization structure is reflected in RBAC in the form of a role hierarchy [40]. Role hierarchy is a transitive and anti-symmetric relation among roles. Roles higher up in the hierarchy are referred to as senior roles and those lower down are junior roles. The major motivation for adding role hierarchy to RBAC was to simplify role management. Senior roles can inherit the permissions of junior roles, or a senior role can activate a junior role, or do both depending on the nature of the hierarchy. This obviates the need for separately assigning the same permissions to all members belonging to a hierarchy. Joshi et al. [24] identify two basic types of hierarchy. The first is the permission inheritance hierarchy where a senior role x inherits the permission of a junior role y. The second is the role activation hierarchy where a user assigned to a senior role can activate a junior role. Each of these hierarchies may be constrained by location and temporal constraints. Consequently, we have a number of different hierarchical relationships in our model.

[Unrestricted Permission Inheritance Hierarchy] Sometimes we want a senior role to inherit permissions of a junior role without any additional spatio-temporal constraints. For example, a contact author can inherit the permissions of the author without any extra spatio-temporal constraints. That is, the contact author can invoke the author's permission wherever and whenever the author can invoke them. Unrestricted permission inheritance hierarchy allows the senior role to

acquire inherited permissions whenever and wherever the junior role can acquire them.

Let x and y be roles such that $x \geq_{(Always,Universe)} y$, that is, senior role x has an unrestricted permission-inheritance relation over junior role y. In such a case, x inherits y's permissions without any additional spatio-temporal constraints. This is formalized as follows:

$$(x \geq_{(Always,Universe)} y) \wedge PermRoleAcquire(p,y,d,l) \Rightarrow PermRoleAcquire(p,x,d,l)$$

[Unrestricted Activation Hierarchy] Sometimes a senior role may want to activate a junior role without placing any additional constraints. For example, a user who has a role of mobile user can activate the weekend mobile user role only if he/she is in the US during the weekend. Unrestricted activation hierarchy allows the senior role to be activated whenever and wherever the junior role can be activated.

Let x and y be roles such that $x \succcurlyeq_{(Always,Universe)} y$, that is, senior role x has an unrestricted role-activation relation over junior role y. Then, a user assigned to role x can activate role y at any time and at any place that y can be activated. This is formalized as follows:

$$(x \succcurlyeq_{(Always,Universe)} y) \wedge UserRoleActivate(u,x,d,l) \wedge (d,l) \subseteq RoleEnableTimeLoc(y) \Rightarrow$$
$$UserRoleActivate(u,y,d,l)$$

[Time Restricted Permission Inheritance Hierarchy] Sometimes a senior role can inherit a junior role only at certain times. For example, a company may have a policy that allows the project manager to inherit the permissions of the code developer role only when the product deadline date is less than a given threshold. Time restricted permission inheritance hierarchy allows the senior role to acquire the permissions of the junior role when the temporal constraints associated with the hierarchy hold and the senior role satisfies the spatio-temporal constraints that are needed by the junior role to invoke those permissions.

Let x and y be roles such that $x \geq_{(d',Universe)} y$, that is, senior role x has a time restricted permission-inheritance relation over junior role y. In such a case, x inherits y's permissions together with

the temporal constraints associated with the permissions and the hierarchy. This is formalized as follows:

$$(x \geq_{(d',Universe)} y) \wedge PermRoleAcquire(p,y,d,l) \Rightarrow PermRoleAcquire(p,x,d \cap d',l)$$

[Time Restricted Activation Hierarchy] In some applications, the senior role may need to be activated only during specific periods. For example, the account auditor role can activate the accountant role only during the auditing period. Time restricted activation hierarchy allows the senior role to activate the junior role when the temporal constraints associated with the hierarchy hold and the senior role satisfies the spatio-temporal constraints that are needed to activate the junior role.
Let x and y be roles such that $x \succcurlyeq_{(d',Universe)} y$, that is, senior role x has a role-activation relation over junior role y. Then, a user assigned to role x can activate role y only at the location and time when role y can be enabled and the additional temporal constraints are satisfied. This is formalized as follows:

$$(x \succcurlyeq_{(d',Universe)} y) \wedge UserRoleActivate(u,x,d,l) \wedge (d,l) \subseteq RoleEnableTimeLoc(y) \Rightarrow$$
$$UserRoleActivate(u,y,d \cap d',l)$$

[Location Restricted Permission Inheritance Hierarchy] Sometimes a senior role can inherit a junior role only in certain locations. For example, a top secret nuclear scientist inherits the permissions of a nuclear scientist only in top secret locations. Location restricted permission inheritance allows the senior role to acquire the permissions of the junior role when the location constraints associated with the hierarchy hold and the senior role satisfies the spatio-temporal constraints that are needed by the junior role to invoke those permissions.
Let x and y be roles such that $x \geq_{(Always,l')} y$, that is, senior role x has a location restricted permission-inheritance relation over junior role y. In such a case, x inherits y's permissions together with the location constraints associated with the permission and the hierarchy. This is formalized as follows:

$(x \geq_{(Always,l')} y) \wedge PermRoleAcquire(p,y,d,l) \Rightarrow PermRoleAcquire(p,x,d,l \cap l')$

[Location Restricted Activation Hierarchy] Sometimes we want the senior role to be able to activate the junior role only at certain locations. For example, a department chair can activate a staff role only when he is in the department. Location restricted activation hierarchy allows the senior role to activate the junior role when the location constraints needed for the hierarchy activation hold and the senior role satisfies the spatio-temporal constraints needed to activate the junior role.

Let x and y be roles such that $x \succcurlyeq_{(Always,l')} y$, that is, senior role x has a role-activation relation over junior role y. Then, a user assigned to role x can activate role y only at the places when role y can be enabled and the location constraints of the hierarchy are satisfied. This is formalized as follows:

$(x \succcurlyeq_{(Always,l')} y) \wedge UserRoleActivate(u,x,d,l) \wedge (d,l) \subseteq RoleEnableTimeLoc(y) \Rightarrow$
$UserRoleActivate(u,y,d,l \cap l')$

[Time Location Restricted Permission Inheritance Hierarchy] Sometimes we may want to place additional temporal as well as spatial constraints on the permission inheritance hierarchy. For instance, a doctor can inherit the daytime nurse role only when he is in the hospital at the daytime. Time-location restricted permission inheritance hierarchy allows the senior role to invoke the permissions of the junior role provided the senior role satisfies the spatio-temporal constraints of the inheritance hierarchy and also the spatio-temporal constraints needed to acquire the permissions of the junior role.

Let x and y be roles such that $x \geq_{(d',l')} y$, that is, senior role x has a time-location restricted permission-inheritance relation over junior role y. In such a case, x inherits y's permissions together with the temporal and location constraints associated with the permission together with the temporal and location constraints associated with the hierarchy. This is formalized as follows:

$(x \geq_{(d',l')} y) \wedge PermRoleAcquire(p,y,d,l) \Rightarrow PermRoleAcquire(p,x,d \cap d', l \cap l')$

[Time Location Restricted Activation Hierarchy] Sometimes additional spatial and temporal constraints must be satisfied for a senior role to activate a junior role. Emergency physicians can activate the role of primary care physicians only when the patient is in an emergency room. Time location restricted activation hierarchy allows the senior role to activate the junior role when the spatio-temporal constraints associated with the hierarchy are satisfied together with the spatio-temporal constraints associated with the invocation of the junior role.

Let x and y be roles such that $x \succcurlyeq_{(d',l')} y$, that is, senior role x has a role-activation relation over junior role y. Then, a user assigned to role x can activate role y only at the places and during the time when role y can be enabled, and the additional spatio-temporal constraints assigned to the hierarchy are satisfied. This is formalized as follows:

$$(x \succcurlyeq_{(d',l')} y) \wedge UserRoleActivate(u,x,d,l) \wedge (d,l) \subseteq RoleEnableTimeLoc(y) \Rightarrow$$
$$UserRoleActivate(u,y,d \cap d', l \cap l')$$

It is also possible for a senior role and a junior role to be related with both permission inheritance and activation hierarchies. In such a case, the application will choose the type of inheritance hierarchy and activation hierarchy needed.

3.4 Impact of Time and Location on Static Separation Of Duty Constraints

Separation of duty (SoD) protects against the fraud that may be caused from a user or role gaining too much power [44]. SoD can be either static or dynamic. Static Separation of Duty (SSoD) comes in two varieties. The first one, which is referred to as *SSoD – User Role Assignment (SSoD-URA)*, is with respect to user-role assignment. SSoD-URA is specified as a relation between roles – the same user cannot be assigned to the roles that are related by the SSoD-URA relation. The second one, which is referred to as *SSoD – Permission Role Assignment (SSoD-PRA)*, is with respect to permission-role assignment. SSoD-PRA is specified as a relation between permissions – the same role cannot be assigned to the permissions that are related by the SSoD-PRA relation. Due to the

presence of temporal and spatial constraints, we can have different flavors of separation of duties – some that are constrained by temporal and spatial constraints and others that are not. In the following, we describe the different types of separation of duty constraints.

[Weak Form of SSoD - User-Role Assignment] Let x and y be two roles such that $x \neq y$. $(x,y) \in SSOD\_URA_w$ if the following condition holds:

$$UserRoleAssign(u,x,d,l) \Rightarrow \neg\, UserRoleAssign(u,y,d,l)$$

The above definition says that a user u assigned to role x during time d and location l cannot be assigned to role y at the same time and location if x and y are related by $SSOD\_URA_w$. An example where this form is useful is that a user should not be assigned the audience role and mobile user role at the same time and location.

[Strong Temporal Form of SSoD - User-Role Assignment] Let x and y be two roles such that $x \neq y$. $(x,y) \in SSOD\_URA_t$ if the following condition holds:

$$UserRoleAssign(u,x,d,l) \Rightarrow \neg\, (\exists d' \subseteq always \bullet UserRoleAssign(u,y,d',l))$$

The above definition says that a user u assigned to role x during time d and location l cannot be assigned to role y at any time in the same location if x and y are related by $SSOD\_URA_t$. The consultant for oil company A will never be assigned the role of consultant for oil company B in the same country.

[Strong Spatial Form of SSoD - User-Role Assignment] Let x and y be two roles such that $x \neq y$. $(x,y) \in SSOD\_URA_l$ if the following condition holds:

$$UserRoleAssign(u,x,d,l) \Rightarrow \neg\, (\exists l' \subseteq Universe \bullet UserRoleAssign(u,y,d,l'))$$

The above definition says that a user u assigned to role x during time d and location l cannot be assigned to role y at the same time at any location if x and y are related by $SSOD\_URA_l$. A person cannot be assigned the roles of student and instructor of the same course at the same time at any location.

[Strong Form of SSoD - User-Role Assignment] Let x and y be two roles such that $x \neq y$. $(x,y) \in SSOD\_URA_s$ if the following condition holds:

$$UserRoleAssign(u,x,d,l) \Rightarrow \neg\ (\exists l' \subset Universe, \exists d' \subseteq always \bullet UserRoleAssign(u,y,d',l'))$$

The above definition says that a user u assigned to role x during time d and location l cannot be assigned to role y at any time or at any location if x and y are related by $SSOD\_URA_s$. The same employee cannot be assigned the roles of minority and non-minority employee at any given corporation.

We next consider the second form of static separation of duty that deals with permission-role assignment. The idea is that the same role should not acquire conflicting permissions.

[Weak Form of SSoD - Permission-Role Assignment] Let p and q be two permissions such that $p \neq q$. $(p,q) \in SSOD\_PRA_w$ if the following condition holds:

$$PermRoleAcquire(p,x,d,l) \Rightarrow \neg\ PermRoleAcquire(q,x,d,l)$$

The above definition says that if permissions p and q are related through weak SSoD Permission-Role Assignment and x has permission p at time d and location l, then x should not be given permission q at the same time and location. The same role should not be assigned the permission of chairing the session and presenting the paper in the conference at the same location and at the same

time.

[Strong Temporal Form of SSoD - Permission-Role Assignment] Let p and q be two permissions such that $p \neq q$. $(p,q) \in SSOD\_PRA_t$ if the following condition holds:

$$PermRoleAcquire(p,x,d,l) \Rightarrow \neg\ (\exists d' \subseteq always \bullet PermRoleAcquire(q,x,d',l))$$

The above definition says that if permissions p and q are related through strong temporal SSoD Permission-Role Assignment and x has permission p at time d and location l, then x should not get permission q at any time in location l. The accountant should not get both the permissions of modifying accounts and auditing accounts at the same branch location at any time.

[Strong Spatial Form of SSoD - Permission-Role Assignment] Let p and q be two permissions such that $p \neq q$. $(p,q) \in SSOD\_PRA_l$ if the following condition holds:

$$PermRoleAcquire(p,x,d,l) \Rightarrow \neg\ (\exists l' \subseteq Universe \bullet PermRoleAcquire(q,x,d,l'))$$

The above definition says that if permissions p and q are related through strong spatial SSoD Permission-Role Assignment and x has permission p at time d and location l, then x should not be given permission q at the same time. The same role should not be given the permission of grading the exam and taking the exam at the same time at any location.

[Strong Form of SSoD - Permission-Role Assignment] Let p and q be two permissions such that $p \neq q$. $(p,q) \in SSOD\_PRA_s$ if the following condition holds:

$$PermRoleAcquire(p,x,d,l) \Rightarrow \neg\ (\exists l' \subseteq Universe, \exists d' \subseteq always \bullet PermRoleAcquire(q,x,d',l'))$$

The above definition says that if permissions p and q are related through strong SSoD Permission-Role Assignment, then the same role should never be given the two conflicting permissions. The

permission to authorize a check and issue it should not be given to the same role at any time and at any location.

3.5 Impact of Time and Location on Dynamic Separation of Duty Constraints

Dynamic separation of duty addresses the problem that a user is not able to activate conflicting roles during the same session.

[Weak Form of DSoD] Let x and y be two roles such that $x \neq y$. $(x,y) \in DSOD_w$ if the following condition holds:

$$SessionRole(u,x,s,d,l) \Rightarrow \neg\, SessionRole(u,y,s,d,l)$$

The above definition says that if roles x and y are related through weak DSoD and if user u has activated role x in some session s for duration d and location l, then u cannot activate role y during the same time and in the same location in session s. In the same session, a user can activate a sales assistant role and a customer role. However, both these roles should not be activated at the same time in the same location.

[Strong Temporal Form of DSoD] Let x and y be two roles such that $x \neq y$. $(x,y) \in DSOD_t$ if the following condition holds:

$$SessionRole(u,x,s,d,l) \Rightarrow \neg\, (\exists d' \subseteq always, \bullet SessionRole(u,y,s,d',l))$$

The above definition says that if roles x and y are related through strong temporal DSoD and if user u has activated role x in some session s, then u can never activate role y any time at the same location in the same session. In a teaching session in a classroom, a user cannot activate the the grader role once he has activated the student role.

[Strong Spatial Form of DSoD] Let x and y be two roles such that $x \neq y$. $(x,y) \in DSOD_l$ if the following condition holds:

$$SessionRole(u,x,s,d,l) \Rightarrow \neg\ (\exists l' \subseteq Universe \bullet SessionRole(u,y,s,d,l'))$$

The above definition says that if roles x and y are related through strong DSoD and if user u has activated role x in some session s, then u can never activate role y in session s during the same time in any location. If a user has activated the graduate teaching assistant role in his office, he cannot activate the lab operator role at the same time.

[Strong Form of DSoD] Let x and y be two roles such that $x \neq y$. $(x,y) \in DSOD_s$ if the following condition holds:

$$SessionRole(u,x,s,d,l) \Rightarrow \neg\ (\exists l' \subseteq Universe, \exists d' \subseteq always \bullet SessionRole(u,y,s,d',l'))$$

The above definition says that if roles x and y are related through strong DSoD and if user u has activated role x in some session s, then u can never activate role y in the same session. A user cannot be both a code developer and a code tester in the same session.

3.6 Impact of Time and Location on Delegation

Many situations require the temporary delegation of access rights to accomplish a given task. For example, a doctor may give certain privileges to a trained nurse when he is taking a break. In such situations, the doctor can give a subset of his permissions to the nurse for a given period of time. This requirement can be fulfilled by the delegation operation. The entity who delegates his privileges temporarily to another entity is referred to as the delegator. The entity who receives the privilege is known as the delegatee. Delegation can be either *grant* or *transfer*. Granting

of privileges allows the delegator to temporarily assign his privileges to the delegatee without relinquishing his own privileges. Transferring of privileges allows the delegator to transfer his privileges temporarily to the delegatee. Note that, during the period of delegation the delegator does not have the privileges which he has transferred to the delegatee.

The delegator can be either a user or a role. System administrators are responsible for overseeing delegation when the delegator is a role. Individual users administer delegation when the delegator is an user. The delegator can delegate either a set of permissions that he possesses by virtue of being assigned to different roles or he can delegate a set of roles assigned to him directly by the user-role assignment or indirectly by the effect of the activation hierarchy. We can therefore classify delegation on the basis of role delegation or permission delegation. For role delegation, the delegatee can be either role or user. For permission delegation, the delegatee can be role only. This is to maintain the intent of RBAC – permissions should be assigned to user via role, not to user directly.

Role Delegation

A delegator (user or role) can delegate a role to a delegatee. Note that, for a delegator to delegate a role r for time d and at location l, the delegator must have been assigned to the role r during time d and location l either directly or indirectly. Depending on the type of delegation (grant or transfer), the delegator may or may not continue to enjoy the privileges he has delegated.

Let $Delegate_R(dtr, dte, r, \{g,t\}, d, l)$ be the predicate that allows the delegator $dtr \in U \cup R$ to grant (g) or transfer (t) a role r to the delegatee $dte \in U \cup R$ during time d and at location l. This will allow individual user (if $dte \in U$) or all users assigned to dte (if $dte \in R$) to be temporary assigned to role r at the specific location and time. The following specifies the various conditions under which user u' acquires role r for duration d' and location l' by virtue of delegation.

1. $Delegate_R(u, u', r, g, d', l') \Rightarrow UserRoleAssign(u', r, d', l')$

2. $Delegate_R(u,u',r,t,d',l') \Rightarrow UserRoleAssign(u',r,d',l') \wedge \neg UserRoleAssign(u,r,d',l')$
3. $Delegate_R(r',u',r,g,d',l') \Rightarrow UserRoleAssign(u',r,d',l')$
4. $Delegate_R(r',u',r,t,d',l') \wedge UserRoleAssign(u,r',d',l')$
$\Rightarrow UserRoleAssign(u',r,d',l') \wedge \neg UserRoleAssign(u,r,d',l')$
5. $Delegate_R(u,r',r,g,d',l') \wedge UserRoleAssign((u',r',d',l') \Rightarrow UserRoleAssign(u',r,d',l')$
6. $Delegate_R(u,r',r,t,d',l') \wedge UserRoleAssign((u',r',d',l')$
$\Rightarrow UserRoleAssign(u',r,d',l') \wedge \neg UserRoleAssign(u,r,d',l')$
7. $Delegate_R(r'',r',r,g,d',l') \wedge UserRoleAssign(u',r',d',l') \Rightarrow UserRoleAssign(u',r,d',l')$
8. $Delegate_R(r'',r',r,t,d',l') \wedge UserRoleAssign(u',r',d',l') \Rightarrow UserRoleAssign(u',r,d',l')$
$\wedge UserRoleAssign(u,r'',d',l') \wedge \neg UserRoleAssign(u,r,d',l')$

The above eight conditions describe how user u' can be assigned to role r for duration d' and location l' under user to user, role to user, user to role and role to role delegation with the grant and transfer mode. Note that, the transfer mode causes the delegator to lose his privileges. With the effect of role activation hierarchy, the delegatee of a delegated role can also activate all junior roles in the activation hierarchy. Moreover, the delegatee inherits all permissions that the delegated role can acquire directly through the permission-role assignment and indirectly through the permission inheritance hierarchy.

Permission Delegation

A delegator (user or role) can delegate a permission to a delegatee. Note that, for a delegator to delegate a permission p for time d and at location l, the delegator must have acquired the privilege r during time d and location l either directly or indirectly. Depending on the type of delegation grant or transfer, the delegator may or may not continue to enjoy the privileges he has delegated.

Let $Delegate_P(dtr,dte,p,\{g,t\},d,l)$ be the predicate that allows the delegator $dtr \in U \cup R$ to grant or transfer a permission p to the delegatee $dte \in R$ during time d and at location l. The

following specifies the various conditions that allow permission p to be delegated to role r' during time d' and location l'.

1. $Delegate_P(u, r', p, g, d', l') \Rightarrow PermRoleAcquire(p, r', d', l')$
2. $Delegate_P(r, r', p, g, d', l') \Rightarrow PermRoleAcquire(p, r', d', l')$
3. $Delegate_P(r, r', p, t, d', l') \Rightarrow PermRoleAcquire(p, r', d', l') \wedge \neg PermRoleAcquire(p, r, d', l')$

The first two conditions say that if a user u or role r has granted privilege p to role r' for duration d' and location l', then role r' acquires permission p for duration d' and location l'. The last condition says that if a role r' has transferred privilege p to role r' for duration d' and location l', then role r' acquires permission p for duration d' and location l', and role r loses permission p for duration d' and location l'. Note that, we have not specified transfer of privilege from user u to role r'. Since privileges are not directly assigned to any user, permissions cannot be removed directly from the user. The only way to remove permission from a user is to revoke the permission from the role assigned to the user and associated with the permission. However, this will impact all users assigned to this role. Consequently, we do not allow transfer of permission from user to role. Since privileges are not directly assigned to the user, we do not define the permission delegation in which the delegatee is the user.

Delegation Chains

In some cases, the delegator may allow the delegatee to further delegate the privileges that he has acquired by virtue of delegation. This could cause a sequence of delegations called the *delegation chain* or *delegation path* [25, 51]. Once a delegatee is granted a privilege, he can grant or transfer this privilege to another delegatee if the delegation chain is permitted by the delegator. However, if a delegatee is transferred a privilege, he can only transfer it to another delegatee in the presence of the delegation chain. Thus, the transfer operation is more restrictive than grant operation ($grant \geqslant transfer$). We now formally define the two delegation chains that our model supports:

Monotonic Role Delegation Chain and *Monotonic Permission Delegation Chain*.

[Monotonic Role Delegation Chain] Monotonic role delegation chain is the delegation chain of the form:

$$\bigwedge_{i=0}^{n-1} Delegate_R(dte_i, dte_{i+1}, r, gt_{i+1}, d_{i+1}, l_{i+1})$$

where dte_0 represents the original delegator, dte_i represents the delegatee in the i^{th} delegation, gt_i refers to grant or transfer, d_i, l_i refers to the time and location where the i^{th} delegation is valid, and $gt_i \geqslant gt_{i-1}$, $d_{i+1} \subseteq d_i$, and $l_{i+1} \subseteq l_i$. The above formalism implies that this delegation will gradually reduce the spatio-temporal points where the delegation can be granted or transferred. We define monotonic permission delegation chain in a similar manner.

[Monotonic Permission Delegation Chain] Monotonic permission delegation chain is the delegation chain of the form:

$$\bigwedge_{i=0}^{n-1} Delegate_P(dte_i, dte_{i+1}, p, gt_{i+1}, d_{i+1}, l_{i+1})$$

where dte_0 represents the original delegator, dte_i represents the delegatee in the i^{th} delegation, gt_i refers to grant or transfer, d_i, l_i refers to the time and location where the i^{th} delegation is valid, and $gt_i \geqslant gt_{i-1}$, $d_{i+1} \subseteq d_i$, and $l_{i+1} \subseteq l_i$.

The delegator may want to restrict the length of the delegation chain. Let $\mathcal{DC}(dtr, e)$ denote the delegation chain starting from the original delegator *dtr* with respect to delegated entity *e*. The function *depth* when applied to this delegation chain, that is, $depth(\mathcal{DC}(dtr, e))$ gives the total number of delegation operations that occurs in $\mathcal{DC}(dtr, e)$.

4 Graph-Theoretic Representation of the Model

Although our proposed spatio-temporal model is syntactically strong and can represent the spatio-temporal access control policies needed in real-world applications, we propose a graph-theoretic representation that accurately reflects the semantics of the model. Our graph-theoretic representation was inspired by the work of Chen and Crampton [12]. However, we adapt this model to better reflect our semantics. 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). The relationships of our spatio-temporal role-based access control model constitute the edges $E = UA \cup PA \cup PO \cup RH \cup SD \cup RD \cup PD$ 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) = $R \times R$ which can be categorized into
  - activation hierarchy RH_a consisting of unrestricted activation RH_{au}, time restricted activation RH_{at}, location restricted activation RH_{al} and time location restricted activation RH_{atl} hierarchies.
  - permission inheritance hierarchy RH_i consisting of unrestricted permission inheritance RH_{iu}, time restricted permission inheritance RH_{it}, location restricted permission inheritance RH_{il}, and time location permission inheritance RH_{itl} hierarchies.
- Separation of Duty (SD) = $(R \times R) \cup (P \times P)$ which can be categorized into
  - static separation of duty for user role assignments $RSSD$ consisting of weak-form $RSSD_w$, strong temporal form $RSSD_t$, strong spatial form $RSSD_l$ and strong form $RSSD_s$.
  - static separation of duty for permission-role assignment $PSSD$ consisting of weak-form $PSSD_w$, strong temporal form $PSSD_t$, strong spatial form $PSSD_l$ and strong form

$PSSD_s$.

  - dynamic separation of duty DSD consisting of weak-form DSD_w, strong temporal form DSD_t, strong spatial form DSD_l and strong form DSD_s.

- Role Delegation (RD), which can be categorized into
  - Role Delegation to User (RD_U) = $U \times R$
  - Role Delegation to Role (RD_R) = $R \times R$
- Permission Delegation (PD) = $R \times P$

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 $(v_1, v_2) \in (UA \cup RD_U)$ and $(v_{i-1}, v_i) \in (RH_a \cup RD_R)$ for $i = 3, \ldots, n$. An activation path $(v_1, v_2, \ldots, v_n)$ allows user v_1 to activate role v_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 $(v_i, v_{i+1}) \in RH_i$ for $i = 1, \ldots, n-2$, and $(v_{n-1}, v_n) \in (PA \cup PD)$. An usage path $(v_1, v_2, \ldots, v_n)$ allows role v_1 to acquire permission v_n. 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 $(v_{n-1}, v_n) \in PO$. An access path $(v_1, v_2, \ldots, v_{n-1}, v_n)$ allows user v_1 to access object v_n using permission v_{n-1}. We define two functions, ρ and μ, on the edges E of the graph, where $E = UA \cup PA \cup PO \cup RH \cup SD \cup RD \cup PD$. Function ρ represents information associated with delegation edges and is specified as follows. $\rho : (RD_U \cup RD_R \cup PD) \rightarrow (U \cup R) \times \mathbb{N}$ that maps the delegation edge to the corresponding delegator and delegation depth. If a delegator further delegates his delegated entity, the delegation depth of the newly created delegation edge is calculated by subtracting one from the delegation depth of its immediate preceding delegation edge. μ represents the spatio-temporal constraints associated with all the edges in the graph and is defined as follows. $\mu : E \rightarrow 2^{\mathcal{D}}$ where $\mathcal{D}$ denotes the spatio-temporal domain. For $e = (v, v') \in E$, $\mu(v, v')$ denotes the set of spatio-temporal points at which the association between v and v' is enabled. In the following, we describe the value of μ for each type of edge in our graph.

- if $(u,r) \in UA$, then $\mu(u,r) = \{(d,l)|UserRoleActivate(u,r,d,l)\}$ denotes the set of spatio-temporal points in which user u can activate role r.
- if $(r,p) \in PA$, then $\mu(r,p) = \{(d,l)|PermRoleAcquire(p,r,d,l)\}$ denotes the set of spatio-temporal points in which permission p is assigned to role r.
- if $(p,o) \in PO$, then $\mu(p,o) = PermDur(p) \times PermObjLoc(p,o)$ denotes the set of spatio-temporal points at which object o can be accessed by virtue of permission p.
- if $(r',r) \in RH_{au} \cup RH_{iu}$, then $\mu(r',r) = RoleEnableTimeLoc(r)$ because senior role can activate the junior role, or inherit permissions of junior role at all the spatio-temporal points where the junior role can be enabled.
- if $(r',r) \in RH_{at} \cup RH_{it}$, then $\mu(r',r) = (d', Universe) \cap RoleEnableTimeLoc(r)$, where $r' \succcurlyeq_{(d',Universe)} r$ or $r' \geq_{(d',Universe)} r$, because senior role can activate the junior role, or inherit permissions of junior role when the junior role can be enabled and the hierarchy temporal constraints are satisfied.
- if $(r',r) \in RH_{al} \cup RH_{il}$, then $\mu(r',r) = (Always, l') \cap RoleEnableTimeLoc(r)$, where $r' \succcurlyeq_{(Always,l')} r$ or $r' \geq_{(Always,l')} r$, because senior role can activate the junior role, or inherit permissions where the junior roles can be enabled and the hierarchy spatial constraints are satisfied.
- if $(r',r) \in RH_{atl} \cup RH_{itl}$, then $\mu(r',r) = (d',l') \cap RoleEnableTimeLoc(r)$, where $r' \succcurlyeq_{(d',l')} r$ or $r' \geq_{(d',l')} r$, because senior role can activate the junior role, or inherit permissions where and when both the roles can be enabled, and the spatio-temporal constraints of the hierarchy are satisfied.
- if $(r',r) \in RSSD_w \cup DSD_w$, then $\mu(r',r) = (d,l)$ denotes the set of points in space-time where no user should be assigned/allowed to activate roles r and r'.
- if $(r',r) \in RSSD_t \cup DSD_t$, then $\mu(r',r) = (Always, l)$ because the same user cannot be assigned/allowed to activate roles r and r' at specified location l at any time.
- if $(r',r) \in RSSD_l \cup DSD_l$, then $\mu(r',r) = (d, Universe)$ denotes the spatio-temporal points

where the same user cannot be assigned or allowed to activate roles r and r' from any location.

- if $(r',r) \in RSSD_s \cup DSD_s$, then $\mu(r',r) = (Always, Universe)$ because no user can be assigned or allowed to activate roles r and r' from any place and at any time.
- if $(p',p) \in PSSD_w$, then $\mu(p',p) = (d,l)$ denotes the set of points in space-time where no role should be assigned to conflicting permissions p and p'.
- if $(p',p) \in PSSD_t$, then $\mu(p',p) = (Always,l)$ denotes the set of spatio-temporal points where the same role cannot be assigned to conflicting permissions p and p' at any time.
- if $(p',p) \in PSSD_l$, then $\mu(p',p) = (d,Universe)$ denotes the set of spatio-temporal points where the same role cannot be assigned to conflicting permissions p and p' at any location.
- if $(p',p) \in PSSD_s$, then $\mu(p',p) = (Always,Universe)$ because no role can be assigned to conflicting permissions p and p' from any place and at any time.
- if $(u',r) \in RD_U$, then $\mu(u',r) = \{(d,l) | Delegate_R(u,u',r,\{g,t\},d,l) \vee Delegate_R(r',u',r,\{g,t\},d,l)\}$ denotes the set of points in space-time where user u' has been delegated role r.
- if $(r',r) \in RD_R$, then $\mu(r',r) = \{(d,l) | Delegate_R(u,r',r,\{g,t\},d,l) \vee Delegate_R(r'',r',r,\{g,t\},d,l)\}$ denotes the set of points in space-time where role r' has been delegated role r.
- if $(r,p) \in PD$, then $\mu(r,p) = \{(d,l) | Delegate_P(u,r,p,g,d,l) \vee Delegate_P(r'',r,p,\{g,t\},d,l)\}$ denotes the set of points in space-time where role r has acquired permission p by virtue of permission delegation.

We write $\hat{\mu}(v_1,\ldots,v_n) = \hat{\mu}(v_1,v_n) \subseteq \mathcal{D}$ to denote $\bigcap_{i=1}^{n-1} \mu(v_i,v_{i+1})$. Hence, $\hat{\mu}(v_1,v_n)$ is the set of points at which every edge in the path is enabled. The authorization scheme in the access control graph can be summarized as follows:

- 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 there exists an u-path

Table 1: DDS Permissions List

	Task		Task
p_1	Read Premise Information	p_{10}	Read Vector Control (VC) Protocols
p_2	Change Premise Information	p_{11}	Change Vector Control Protocols
p_3	Read Case Information	p_{12}	Read Work Record
p_4	Change Case Information	p_{13}	Change Work Record
p_5	Read Patient Record	p_{14}	Read VC Material Information
p_6	Change Patient Record	p_{15}	Change VC Material Information
p_7	Read Patient Names	p_{16}	Signal VC for Dengue Virus (DV)
p_8	Read Work Schedule	p_{17}	Signal VC for Dengue Hemorrhagic Fever (DHF)
p_9	Change Work Schedule		

$v = v_1, v_2, \ldots, v_n = v'$ 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 $d \in \hat{\mu}(v, v'')$.

Note that, generating the access control graph consists of two steps. First, we have to create all vertices corresponding to the entities which takes $O(V)$ time. Next, we have to create all edges corresponding to the relationships between entities. We label these edges and also indicate the constraints associated with them. This step takes $O(E)$ time. Hence, total time to create the whole graph is $O(V+E)$.

4.1 Example Application

In this section, we present a real-world application called the Dengue Decision Support (DDS) system to illustrate our approach. The DDS helps state-level public health officials respond to local outbreaks of dengue. Response consists of vector control and vector surveillance, namely spraying for mosquitoes (control) and investigating locations where they might be breeding and living (surveillance) and where the level of confirmed dengue cases have increased above a prescribed threshold. Public health officials are organized in jurisdictions, based on population, and multiple

jurisdictions are included in a single state. When the threshold is reached, officials at both levels respond. The jurisdiction officer activates vector control and surveillance teams that are local to the jurisdiction, with instructions regarding the specific control and surveillance protocols to follow and the locations where they are to be performed. The state officer releases materials for control to the team, and the local team then performs the controls and surveillance ordered. The jurisdiction and state vector control officials are often located in different buildings, although the vector control team is co-located with the jurisdiction officer. All control materials are located in warehouses elsewhere, and for coordination reasons are controlled by the state officer. Information about specific cases of dengue is retained in what is called an epidemiological study. This data includes information about the patient, the location where the patient lives (the premise), the case, and control and surveillance actions performed at the premise. The patient and case data are considered private information, and are only available to epidemiologists at the jurisdiction and state levels. The vector control team receives premise information along with orders for control and surveillance. However, the team also needs to have names associated with the premises in order to validate the location. The team therefore needs access to some of the patient data for a fixed period of time, in order to perform control and surveillance duties. For lack of space, we omit giving the full specification of the DDS.

4.1.1 DDS Security Policies

Entities DDS system consists of the following entities

- Users: *Alice*, *Bob*, *Ben*, *Charlie*, *Claire* and *David*
- Roles: State Epidemiologist (*State Epi*), Jurisdiction Epidemiologist (*Juris Epi*), Clinic Epidemiologist (*Clinic Epi*), Clinician (*Clinician*), State Vector Control (*State VC*), Jurisdiction Vector Control (*Juris VC*), and Local Jurisdiction Vector Control Team (*Local VC Team*).
- Permissions: p_i where $1 \leq i \leq 17$ whose descriptions are given in Table 1.

Table 2: DDS Role-Permission Assignment Constraints

Role	Tasks	Location Constraint	Time Constraint
State Epi	p_{16}	A–State Office, B–Juris Office	a–Regular Hours
Juris Epi	p_1, p_3 p_{17}	B–Juris Office B–Juris Office	a–Regular Hours b–Always
Clinic Epi	p_{17}	D–Universe	b–Always
Clinician	p_1, p_2	C–Clinic	a–Regular Hours
State VC	p_{11}, p_{15}	A–State Office	a–Regular Hours
Juris VC	p_1, p_8	B–Juris Office	a–Regular Hours
Local VC Team	p_7	E–Emergency Location	a–Regular Hours, c–Emergency Hours

- Objects are omitted from the example to keep it simple.

Role Assignment The user-role assignments and permission-role assignments are specified as follows.

- User-role assignments: $UserRoleAssign(Alice, State\ Epi, b, A \cup B)$, $UserRoleAssign(Bob, Clinic\ Epi, b, C)$, $UserRoleAssign(Ben, Clinician, a, C)$, and $UserRoleAssign(Charlie, State\ VC, a, A \cup B)$.
- Permission-role assignments are summarized in Table 2.

Role Hierarchy Two pairs of roles are related by the unrestricted permission inheritance hierarchy. These relationships are specified as follows:

- *State Epi* $\geq_{(Always, Universe)}$ *Juris Epi*, *State VC* $\geq_{(Always, Universe)}$ *Juris VC* and *Juris VC* $\geq_{(Always, Universe)}$ *Local VC Team*.

Separation of Duty There are three separation of duty constraints in DDS system:

- User should not have permission to change VC protocols at the same time as he has permission to change VC materials.
- User should not have permission to signal DV at the same time as signal DHF.

- User should not be assigned to both Epidemiologist and Vector Control roles at any place and time.

These can be represented as follows:

- $SSOD\_PRA_l = \{(p_{11}, p_{15}), (p_{16}, p_{17})\}$
- $SSOD\_URA_s =$ { (*State Epi, State VC*), (*Juris Epi, State VC*), (*Clinic Epi, State VC*), (*State Epi, Juris VC*), (*Juris Epi, Juris VC*), (*Clinic Epi, Juris VC*) }

Delegation Only one delegation constraint is specified for this application. The system administrator decided to transfer permission p_{17} from *Clinic Epi* role to *Clinician* role during emergency hours at the clinic. The administrator does not allow the delegatee to delegate the permission further. This can be represented in our model as follows:

- $Delegate_P$(*Clinic Epi, Clinician,* p_{17}, t, c, C)
- $depth(\mathcal{DC}$(*Clinic Epi,* p_{17})) $= 1$

The graph representation of the DDS security policies are shown in Figure 1. To avoid crowding the graph, we show the spatio-temporal and delegation constraints in Table 3. The *PD* edge is represented by dashed arrow. *SD* edges are represented by dotted bi-directional arrows. The activation paths and their associated spatio-temporal constraints are listed below:

- *(Alice, State Epi)* where $\hat{\mu}$ *(Alice, State Epi)* $= [b, A \cup B]$
- *(Ben, Clinician)* where $\hat{\mu}$*(Ben, Clinician)* $= [a, C]$
- *(Bob, Clinic Epi)* where $\hat{\mu}$*(Bob, Clinic Epi)* $= [b, C]$
- *(Charlie, State VC)* where $\hat{\mu}$*(Charlie, State VC)* $= [a, A \cup B]$

Some examples of usage paths and their associated spatio-temporal constraints are given below:

- *(Clinician,* p_1*)* where $\hat{\mu}$*(Clinician,* p_1*)* $= [a, C]$

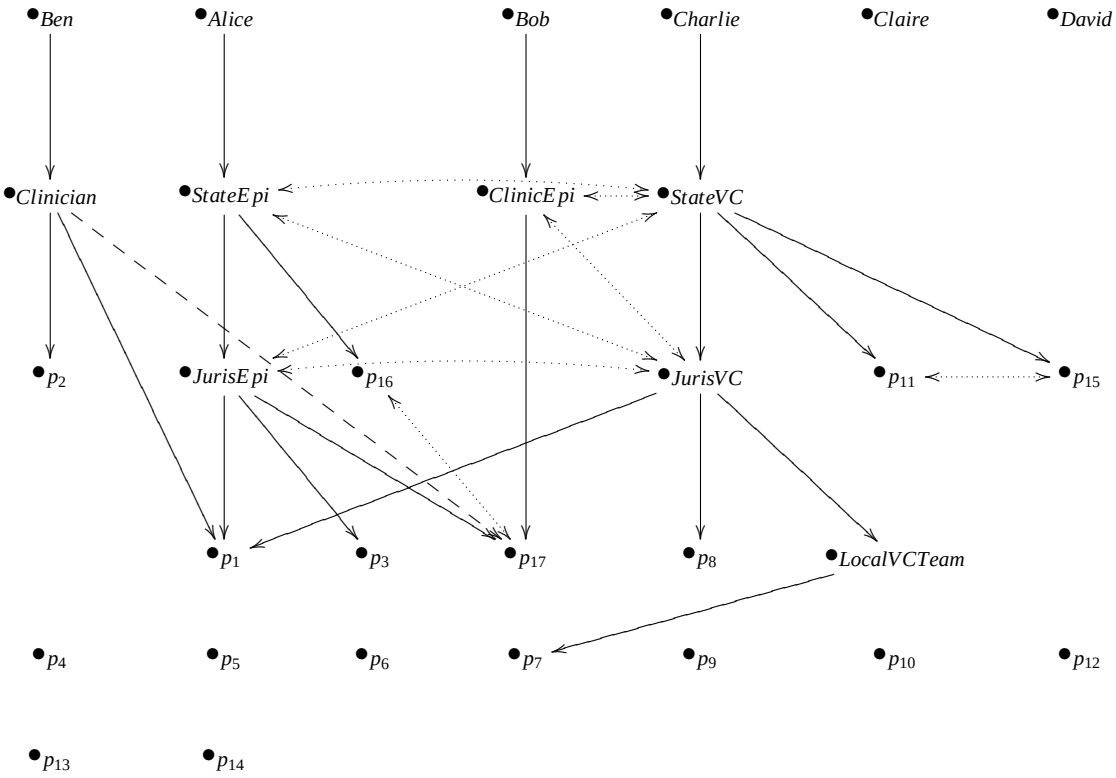

Figure 1: DDS System's Access Control Graph

- *(Juris VC, Local VC Team)* where $\hat{\mu}$*(Juris VC, Local VC Team)* $= [a \cup c, E]$
- *(State VC, Juris VC,* p_1*)* where $\hat{\mu}$*(State VC, JurisVC,* p_1*)* $= [a, B]$
- *(State VC, Juris VC, Local VC Team,* p_7*)* where $\hat{\mu}$*(State VC, Juris VC, Local VC Team,* p_7*)* $= [a, \emptyset]$

Some examples of access paths are as follows:

- *(Alice, State Epi,* p_{16}*)* where $\hat{\mu}$*(Alice, State Epi,* p_{16}*)* $= [a, A \cup B]$
- *(Bob, Clinic Epi,* p_{17}*)* where $\hat{\mu}$*(Bob, Clinic Epi,* p_{17}*)* $= [b, C]$
- *(Charlie, State VC, JurisVC,* p_1*)* where $\hat{\mu}$*(Charlie, State VC, Juris VC,* p_1*)* $= [a, B]$

5 Model Analysis

The model that we proposed earlier has numerous features that can interact with each other to produce inconsistencies and conflicts. For example, incorrect spatio-temporal constraints may prevent a user from invoking his permission. Similarly, incorrect delegation may cause violation of separation of duty constraints. Thus, we must perform an analysis to ensure that inconsistencies or security violations do not occur when a given application is using our model. Manual analysis is error-prone and tedious. Towards this end, we show how Coloured Petri Nets (CPNs) can be used for detecting problems in the authorization specifications.

5.1 Coloured Petri Nets

CPNs [19, 21, 22] have been widely used to model and analyze various types of real-world applications. With a comprehensive graphical representation and well-defined semantics, CPNs allow users to perform formal analysis on a wide range of problems. We start by presenting the characteristics of a CPN model as described by Laborde et al. [27]. The states of a CPN are represented by the *places*, which are drawn as ellipses or circles. Each place is associated with a *color set* that determines the type of data that the place may contain. A state of a CPN is called a *marking*. A marking consists of a number of tokens that belongs to the individual places. Each token carries a value (color), which belongs to the type of the place on which the token resides. The tokens present on a particular place compromise the marking of that place. The tokens of a CPN are distinguishable from each other. The marking of a place is, in general, a multi-set of token values.

The actions of a CPN are represented by means of transitions, which are represented as rectangles. Transitions and places are connected by arcs. Arcs cannot connect any two places or any two transitions. An activation (firing) of a transition removes tokens from places connected to the transition's incoming arcs (input places) and adds tokens to places connected to transition's outgoing arcs (output places). This results in the changes of the marking (state) of the CPN. The exact

number of tokens added and removed by the occurrence of a transition, and their data values are determined by the arc expressions. In addition to the arc expressions, it is possible to attach a boolean expression (with variables), called *guards*, to each transition. The transition can be activated and fired only when its guard function evaluates to true. The initial initial state of the system is described using *initial marking* and the final state is represented by *dead marking*.

Figure 2 shows a simple example of a CPN. This CPN consists of three places and one transition. The *Users* place has a data type *USER* and is assigned an initial marking *AllUsers* consisting of six tokens, namely, *Alice, Ben, Bob, Charlie, Claire*, and *David*. Similarly, the *URA1* place has a data type *URA_TYPE* and is assigned an initial marking *AllURAs* consisting of four tokens. The values of these two places can activate the transition called *CheckURA1*. If there exists a value *u* in both places and the guard function of *CheckURA1* is satisfied, that is, *u* is not null, then the transition *CheckURA1* will be activated and the token of value *u* will be transferred to the terminal place called *AssignedUser*, which has type *USER*. For further details on CPN, we refer the interested reader to [21].

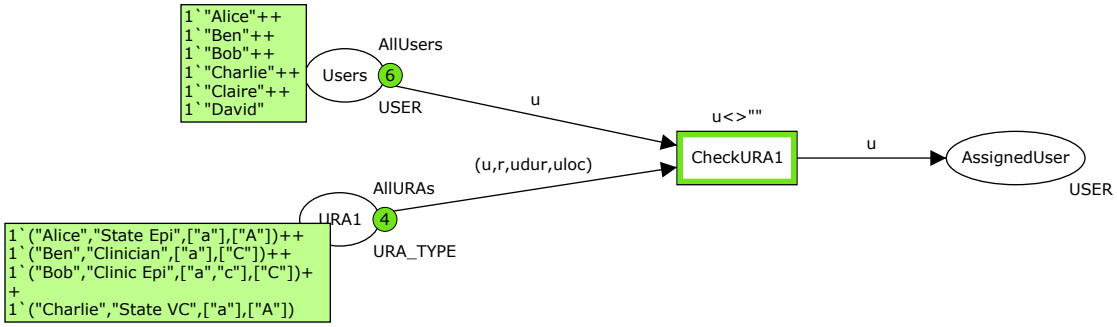

Figure 2: Simple example of CPN model

In this paper, we advocate the use of the CPN Tools [20, 22] to develop and analyze the CPN model representing the access control specification. Using CPN Tools allows us to investigate the behavior of the CPN model using simulation and state space analysis. CPN Tools will generate all possible states that are reachable together with the values of environmental variables that cause the

change. Checking all the generated states is a time consuming and error-prone task. To solve this problem, we create queries using the Standard ML language [30] to select only those states which have the exact properties that we are interested in. To avoid state explosion, we develop a CPN model for each of the problems that we try to detect. The models are populated using values from the access control graph representing the access control policies of the organization.

The CPN Tools have a well-defined user interface, shown in Figure 3, for creating the CPN models and populating them with the initial values. Once the CPN model has been created, the tool allows one to do a state space analysis. The CPN Tools will generate all possible *states* of the model. Figure 4 shows the graphical representation of one of the states generated by the model in Figure 3. Note that, different states will store different tokens in each place. The state is represented by a round cornered rectangle, where 1 is the state number, 0 is the number of predecessors, and 4 is the number of successors. The sharp cornered rectangle shows the description of the state which consists of the current values stored in each place at that specific state. Since the number of predecessors of the state in Figure 4 equals to zero, the figure represents the initial state where none of the tokens has been moved to the new place. Figure 5 shows the complete set of states generated by the model in Figure 3. The arrow connects a predecessor state to each of its successor states. For the complete reference on the CPN Tools, please refer to [22]. The details about how to perform the state space analysis can be found in [30].

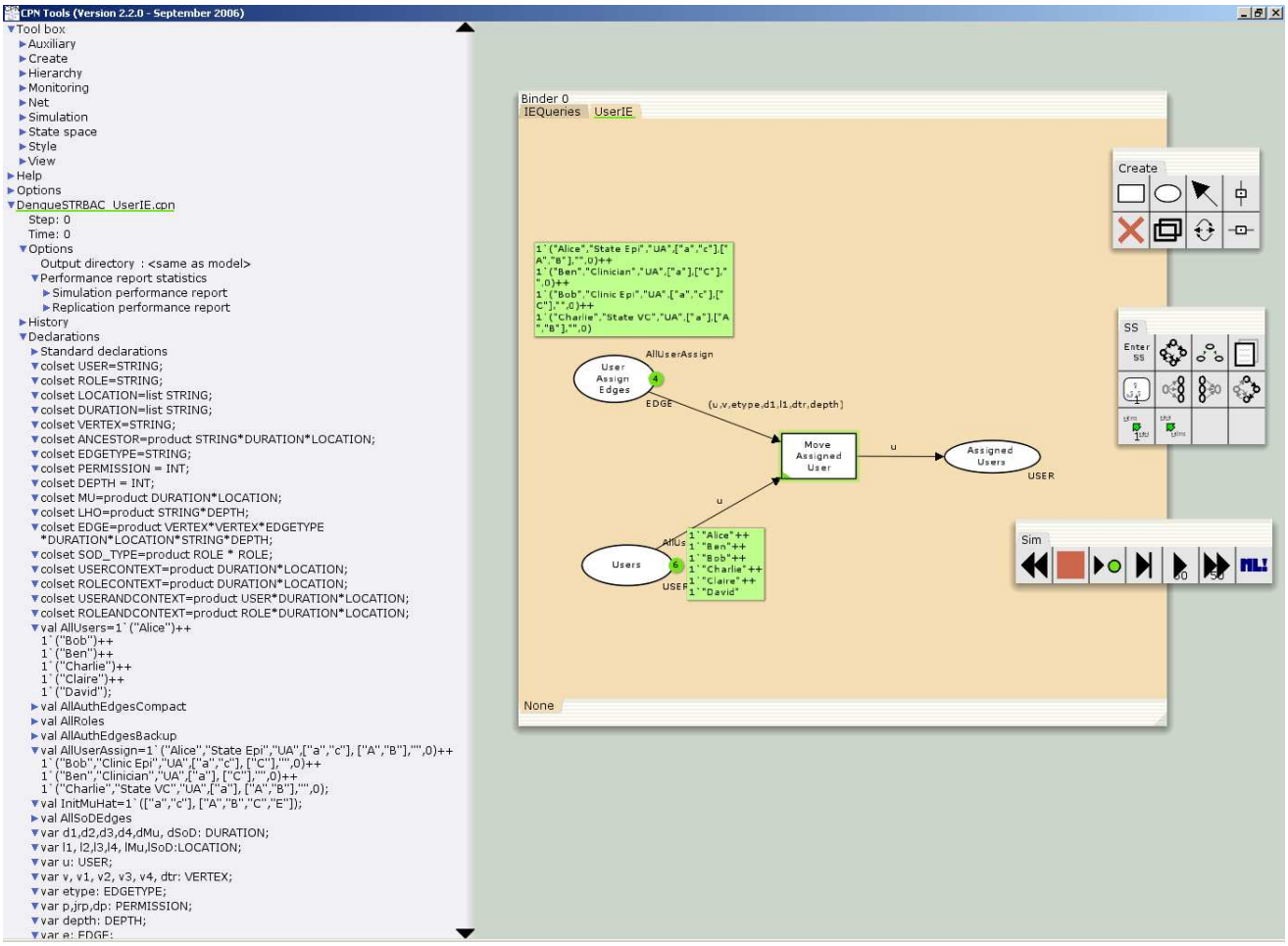

Figure 3: CPN Tools User Interface

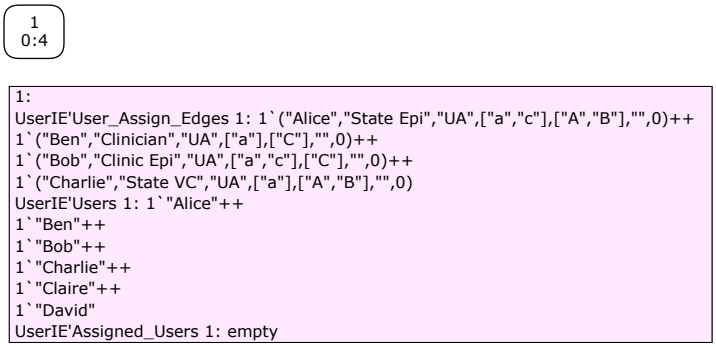

Figure 4: Example of values stored in each state

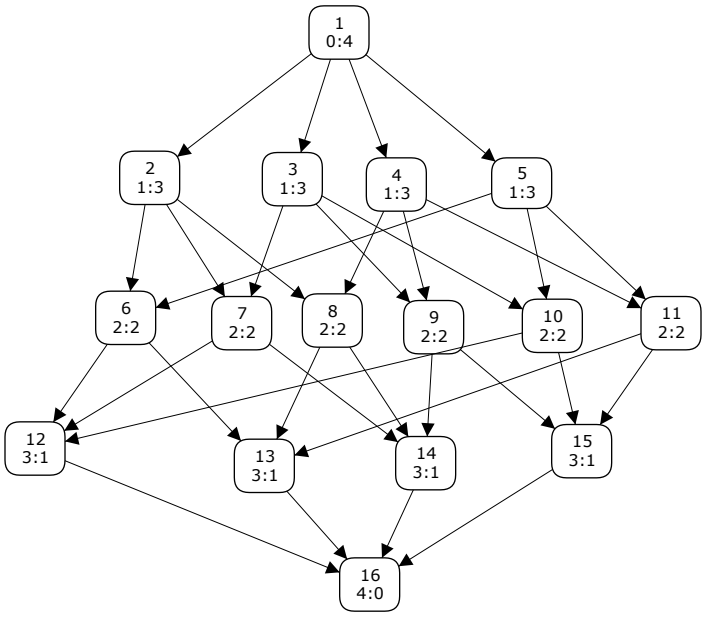

Figure 5: Example of complete state graph

In this paper, we detect the following problems with the access control specification:

- Isolated entity occurs when an entity is not connected to any other entity.
- Infeasible path occurs when a user cannot access a permission or an object in an access path.

- Delegation constraint violation occurs when the spatio-temporal constraints associated with delegation or the delegation depth constraint is violated.
- Separation of duty violation occurs when a user is assigned conflicting roles, when a permission is assigned conflicting roles, or when a user is able to activate conflicting roles.

5.2 Isolated Entity Detection

Isolated entity occurs when an entity is disconnected from other entities in the access control graph, thus making it useless with respect to the access control specification. Consider the DDS example discussed in Section 4.1. If we look at the graph in Figure 1 representing the access control policies of the DDS, we find that users *Claire* and *David* are not connected to any roles or permissions – these are examples of isolated entities. A similar argument can be made for permissions p_4 and p_5. In our model, we can have three types of isolated entities, corresponding to users, roles, and permissions, as described below.

1. *Type 1:* User who is not assigned to any role which prevents him from acquiring any permission.
2. *Type 2:* Role which is not assigned to any permission or junior role and therefore cannot use any permission.
3. *Type 3:* Permission that is not assigned to any role which prevents it from being invoked.

We develop CPN models to detect each of these types of isolated entities. In the following, we describe how to detect isolated users, that is, isolated entity of Type 1.

```
colset USER=STRING;
colset LOCATION = list STRING;
colset DURATION = list STRING;
colset VERTEX = STRING;
```

```
colset EDGETYPE = STRING;
colset DEPTH = INT;
colset EDGE = product VERTEX*VERTEX*EDGETYPE*DURATION*LOCATION*VERTEX*DEPTH;
```

All types of entities and relationships in our model are represented using *color sets*. From the declaration above, edge is represented by a tuple of vertices. The color set called *EDGETYPE* is used to distinguish between different types of edges. To represent μ function, we use the product of *DURATION* and *LOCATION*. Similarly to represent ρ function, we use the product of *STRING* and *DEPTH*.

We next model the states of the application that are of interest. The state of the application is represented using CPN's *places* which are drawn as ellipses or circles. Each place has an associated type, specified using color set, that determines the data type that the place may contain. In Figure 6, we have three places denoted by *User Assign Edges*, *Users*, and *Assigned Users* that have data types *USER*, *EDGE* and *USER* respectively. Each state of a CPN is called a *marking*. The marking of a place is represented by a multi-set of token values. The initial markings, representing the initial states, are initialized using values from the access control graph and are shown in the boxes adjoining the places. For example, the initial marking of the *Users* place, referred to as *AllUsers*, consists of six tokens corresponding to the users *Alice*, *Ben*, *Bob*, *Charlie*, *Claire* and *David* in the access control graph. *AllUsers* is described using a multi-set. Since all users are unique, the number of each multi-set member equals one. For example, the notation, `1‘("Alice")` indicates there is only one user *Alice*. The union operation ($++$) is used to represent situations when there are more than one member, as in our example. The initial marking of place *User Assign Edges*, referred to as *AllUserAssign*, is specified in a similar manner and are populated using User-Role Assignment and Role to User Delegation edges from the access control graph. Here, we repeat the specifications of the initial markings.

```
val AllUsers=1‘("Alice")++1‘("Bob")++1‘("Ben")++1‘("Charlie")++
```

```
1‘("Claire")++1‘("David");

val AllUserAssign=1‘("Alice","State Epi","UA",["a","c"], ["A","B"],"",0)++
1‘("Bob","Clinic Epi","UA",["a","c"], ["C"],"",0)++
1‘("Ben","Clinician","UA",["a"], ["C"],"",0)++
1‘("Charlie","State VC","UA",["a"], ["A","B"],"",0);
```

The actions of the CPN are described by transitions, which are represented using rectangles. Arcs connect transitions and places. An activation (firing) of a transition removes tokens from places connected to the transition's incoming arcs (input places) and adds tokens to the places connected to the transition's outgoing arcs (output places). This results in the markings of the CPN, that symbolizes its state, to change. It is also possible to attach a boolean expression, referred to as a guard, to each transition. In such a case, the guard function must evaluate to true before it can be activated. The exact number of tokens added or removed by the firing of a transition and their respective data values are determined by the arc expressions. The transitions can be fired repeatedly. When the marking of a place can no longer be changed, it is referred to as dead marking.

Figure 6 shows one transition *Move Assigned User* that is activated when the arc expressions match on the *u* values and the guard function of *Move Assigned User* verifies that *u* is not null. The initial markings cause this transition to be fired. The corresponding *u*, $(u, v, etype, d1, l1, dtr, depth)$ get removed from *Users* and *User Assign Edges* places respectively and *u* gets added to *Assigned Users*. The transitions are fired repeatedly until no more state change can take place. In the given example, the transitions are fired for users *Alice*, *Ben*, *Bob* and *Charlie*. The terminal state is reached when no more transitions can be fired.

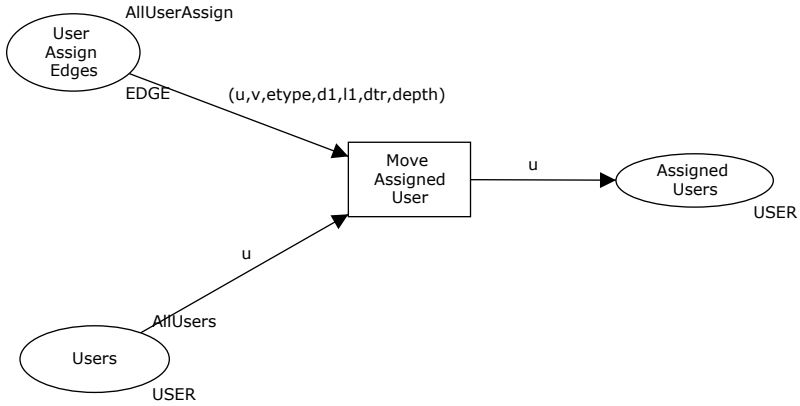

Figure 6: CPN Model for Isolated Entity Detection (Type 1)

Query 1

Show all terminal states

```
SearchNodes (EntireGraph,
fn n => (length(OutArcs(n)) = 0),
NoLimit,
fn n => n,
[],
op ::)
```

We use Query 1, which is the general query to show all terminal states, to detect isolated entity. This query is written using built-in query function of the State Space Tool called *SearchNodes* [20]. The first argument in *SearchNodes*, namely, `EntireGraph`, signify that we want to search the whole graph. The second argument, `fn n => (length(OutArcs(n)) = 0)`, states that we want to check all nodes that have no outgoing arcs, that is, the terminal nodes. The third argument, `NoLimit`, states that we want the query to return all possible results. The fourth argument, `fn n => n`, states that we do not want to change the value of the search result. The fifth argument states that the initial value of the result set is equal to empty list. The last argument, `op ::`,

will combine all search results into one list. From the explanation above, Query 1 will return the state where the transition cannot proceed anymore. The result is the state number 16 which can be viewed using the command: `print(NodeDescriptor 16)`.
The content of each place in state number 16 is shown below:

```
Users=1‘("Claire")++1‘("David");
User Assign Edges=empty;
Assigned User=1‘("Alice")++1‘("Ben")++1‘("Bob")++1‘("Charlie");
```

The result shows that tokens corresponding to users Claire and David are in places *Users* when the transitions cannot be fired anymore. These users cannot be transferred to the next state (*Assigned User*) and they are isolated entities. With trivial modification, we can develop the CPN models to detect the other types of isolated entities.

5.3 Infeasible Path Detection

Recall that in an access control graph, a user *u* is authorized for permission *p* through role *r* if there is an access path connecting *u*, *r*, and *p*. The spatio-temporal constraints may be specified in such a manner that it may not be possible for *u* to invoke *r* resulting in an infeasible path. Consider the following access path given in Figure 1: $(Ben, Clinician, p_{17})$. *Ben* is assigned to *Clinician* role during regular hours at the *Clinic*. However, the *Clinician* is delegated permission p_{17} only during emergency hours at the *Clinic*. Thus, the temporal constraints prohibit *Ben* from ever invoking permission p_{17}. This is an example infeasible path.

Figure 7 shows the CPN model for detecting infeasible paths. This model is developed to perform a depth first search on the access control graph and calculate the $\hat{\mu}$ function of each acs-path. If there is an acs-path where the $\hat{\mu}$ function equal to empty set, then this acs-path is the infeasible path. In this CPN model we have a transition called *Get Initial Vertex*. This transition will get the first token needed to start the analysis. Moreover, it will prevent other tokens from being

retrieved while the previous token is still in the analysis process. The transition *Retrieve Edge* will retrieve the authorization edge which starts at *v1*, then add it to the *Authorization Path* place as a record. Then the transition *Calculate Mu Hat* will calculate the current $\hat{\mu}$ value. If either the spatial value or temporal value of $\hat{\mu}$ equals empty set, it will trigger the *Infeasible Path* transition to fire. This transition will send boolean value *true* to the *Infeasible Path* place, which will notify us that there exists an infeasible path in our policy. The initial marking of the *Users* place, denoted by *AllUsers*, consists of all users in the access control model. The initial marking of the *Authorization Edges* place, denoted by *AllAuthEdges*, consists of all edges except the SoD edges in the access control graph.

Query 2

Infeasible Path

```
fun InfeasiblePath() : Node list
= SearchNodes(
EntireGraph,
fn n=>(
(size(Mark.UserInfeasiblePath'Infeasible_Path 1 n) <> 0)
),
NoLimit,
fn n=>n,
[],
op ::)
```

Query 2 checks the infeasible path that may occur due to incorrect specifications in the spatio-temporal constraints. The second argument in SearchNodes which represents a function states that we want to check the states where the number of tokens in `Infeasible Path` place is not equal to

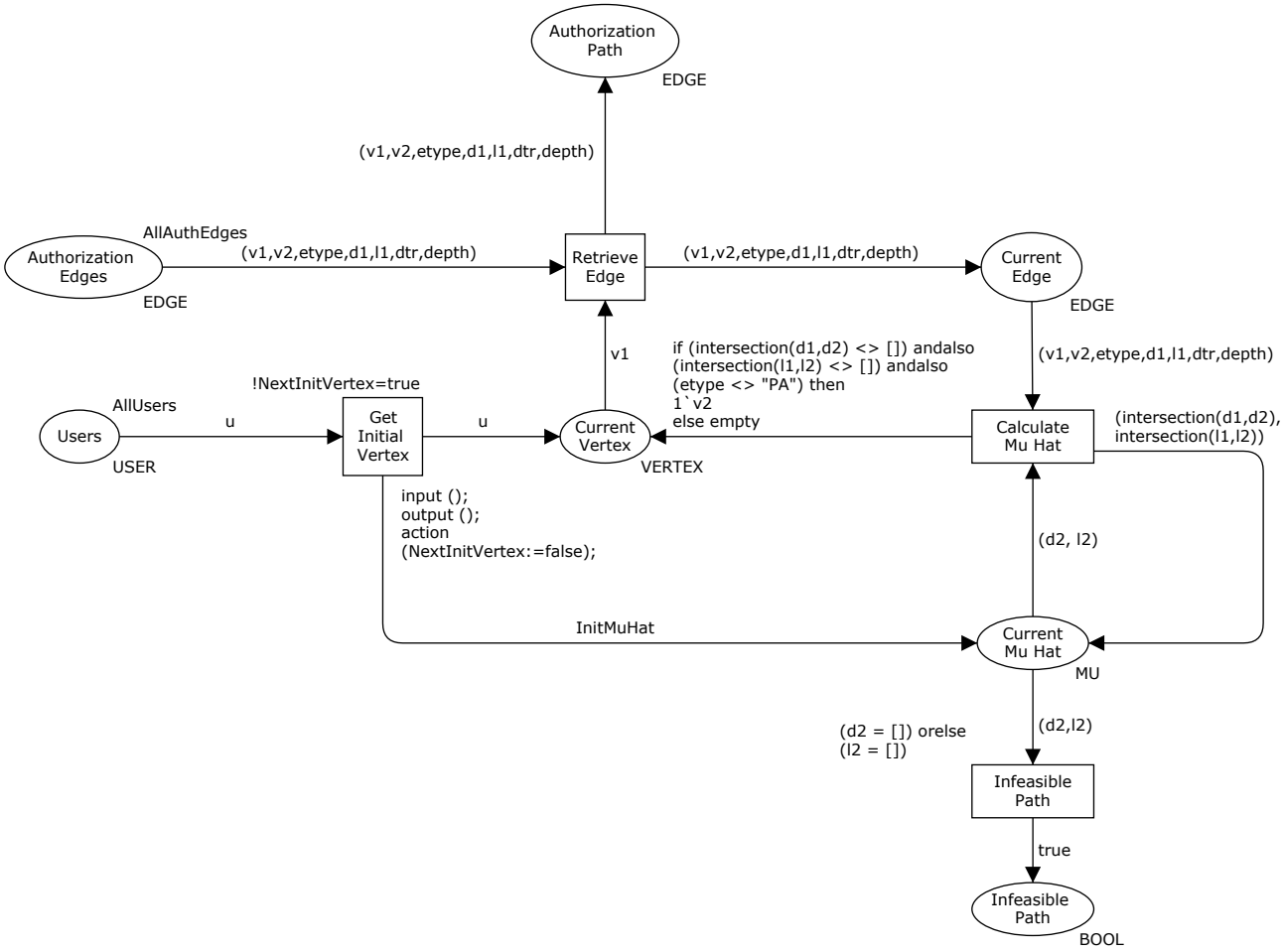

Figure 7: CPN Model for Infeasible Path Detection

zero. The result shows that states 37 and 47 contain the infeasible path. To observe the result, we print the content of state number 47. Below is part of the content of state 47.

```
Authorization Path = 1‘("Charlie","State VC","UA",["a"], ["A","B"],"",0)++
1‘("State VC","Juris VC","RHI",["a"], ["B"],"",0)++
1‘("Juris VC","Local VC Team","RHI",["a","c"], ["E"],"",0);
```

The analysis reveals another infeasible path that exists in our DDS example: *(Charlie, State*

VC, Juris VC, Local VC Team, p_7). This infeasible path is caused because no spatial constraints can be satisfied. *Charlie* is assigned the role *State VC* in the *State Office* and *Juris Office*. However, the *State VC* inherits *Juris VC*'s permissions only in the *Juris Office* and *Juris VC* inherits *Local VC Team*'s permission only in *Emergency Location*. This prevents *Charlie* from invoking any of the *Local VC Team*'s permission. State 37 reveals another infeasible path $(Ben, Clinician, p_{17})$ that exists in our application.

5.4 Delegation Constraint Violation Detection

A delegator can delegate only the roles or privileges assigned to him. Moreover, the delegation duration and location should satisfy the associated spatio-temporal constraints. In the context of our example, if *Clinic Epi* tried to delegate privilege p_3 (which he does not possess), then it would be an example of delegation constraint violation. Similarly, if the role *Juris Epi* delegated permission p_3 to *Clinician* at location *A* (State Office) and time *c* (Emergency Hours), then it would violate the delegation constraint. This is because the role *Juris Epi* does not have permission p_3 in location *A* at time *c*.

The delegation should also not violate the delegation depth constraint. This type of violation occurs when there is a chain of delegation and the delegatee further delegates the privilege beyond the specified depth. For example, if the delegation depth is specified as one, then a delegation depth violation will occur if the delegatee is trying to further delegate the privileges he has acquired by virtue of delegation. In the context of our example, the role *Clinic Epi* transfers the permission p_{17} to *Clinician* at time *c* and location *C* and the delegation depth is specified as 1. Now, if the *Clinician* further delegates privilege p_{17} to some other role, then the delegation depth constraint will be violated.

Figure 8 shows the CPN model to detect the delegation constraint violation. This model is developed to ensure that both delegation depth constraint and delegation spatio-temporal constraint

are satisfied by using the guard function of the transition *Check Delegation Depth* and *Check Delegation Constraint* respectively. If the *Check Delegation Depth* transition is activated, then there exists a delegation depth violation. Similarly, if the *Check Delegation Constraint* is activated, then there exists a spatio-temporal delegation constraint violation that indicates that the delegator is delegating the privileges to which he has no accessibility. The model will send the problematic edge to the place corresponding to each type of error to notify the error.

The initial markings of the *Delegation Edges* place, denoted by *AllDelEdges*, consists of all delegation edges, that is, $RD \cup PD$. The initial markings of the *Delegator Authorization Edges* place, denoted by *AllDtrAuthEdges*, consists of all edges belonging to $UA \cup RH \cup PA$.

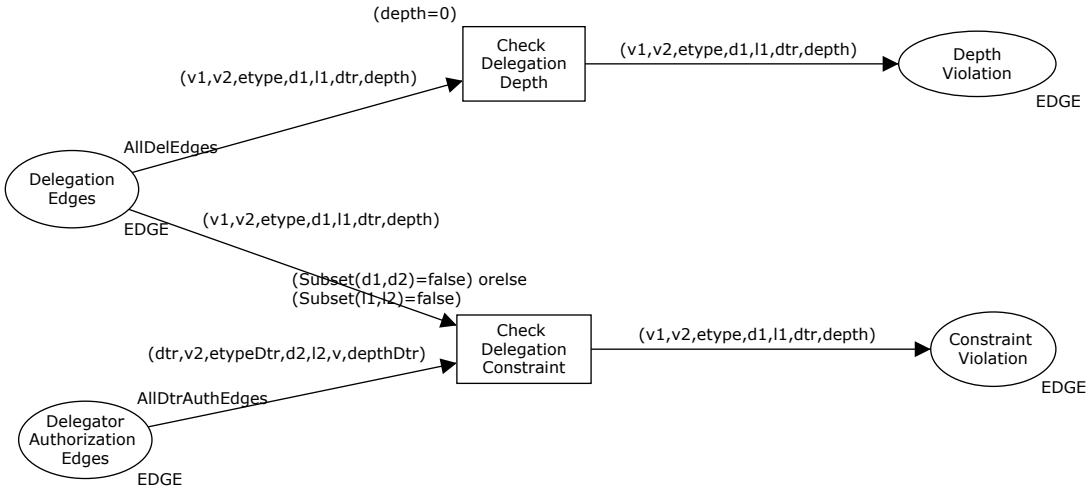

Figure 8: CPN Model for Delegation Constraint Violation Detection

We then formulate queries for delegation depth violation and delegation constraint violation. Both queries return empty list, which ensures that our model is free from both types of violation.

5.5 SoD Violation Detection

Separation of duty violations can be static or dynamic. Static separation of duty can be with respect to the user-role assignment or permission-role assignment. In DDS system we have two different types of SSoD–one with respect to user-role assignment and the other with respect to permission-role assignment. Let us take the example of SSoD for permission-role assignment. No role should have permissions p_{16} (Signal VC for Dengue Virus) and p_{17} (Signal VC for Dengue Hemorrhagic Fever) at the same time. Thus, if a role does have these conflicting permissions, SSoD will be violated.

Figure 9 shows the CPN model to detect separation of duty violations. The model will perform a reverse depth first search starting from the vertices associated with the SoD edge. The ancestors of the two vertices will be stored in two separate places called *V1 Ancestors* and *V2 Ancestors* respectively, together with their corresponding $\hat{\mu}$ value. If there exist a common ancestor and there is an overlap of spatio-temporal points, then SoD is violated. The model will then send the problematic SoD and its ancestor to *SoD Violate* and *SoD Violate Ancestor* places respectively to notify the error.

The initial marking of the *SoD Edges*, denoted by *AllSoDEdges* is populated by all SoD edges in the access control model. The initial markings of *Authorization Edges V1* and *Authorization Edges V2*, denoted by *AllAuthEdges*, consists of all edges except the SoD edges in the access control graph. The content of *AllSoDEdges* is shown below.

```
val AllSoDEdges=
1‘("State Epi","State VC","RSSD",["a","c"],["A","B","C","E"],"",0)++
1‘("State Epi","Juris VC","RSSD",["a","c"],["A","B","C","E"],"",0)++
1‘("Juris Epi","State VC","RSSD",["a","c"],["A","B","C","E"],"",0)++
1‘("Juris Epi","Juris VC","RSSD",["a","c"],["A","B","C","E"],"",0)++
1‘("Clinic Epi","State VC","RSSD",["a","c"],["A","B","C","E"],"",0)++
```

```
1‘("Clinic Epi","Juris VC","RSSD",["a","c"],["A","B","C","E"],"",0)++
1‘("p11","p15","PSSD",["a"],["A","B","C","E"],"",0)++
1‘("p16","p17","PSSD",["a"],["A","B","C","E"],"",0);
```

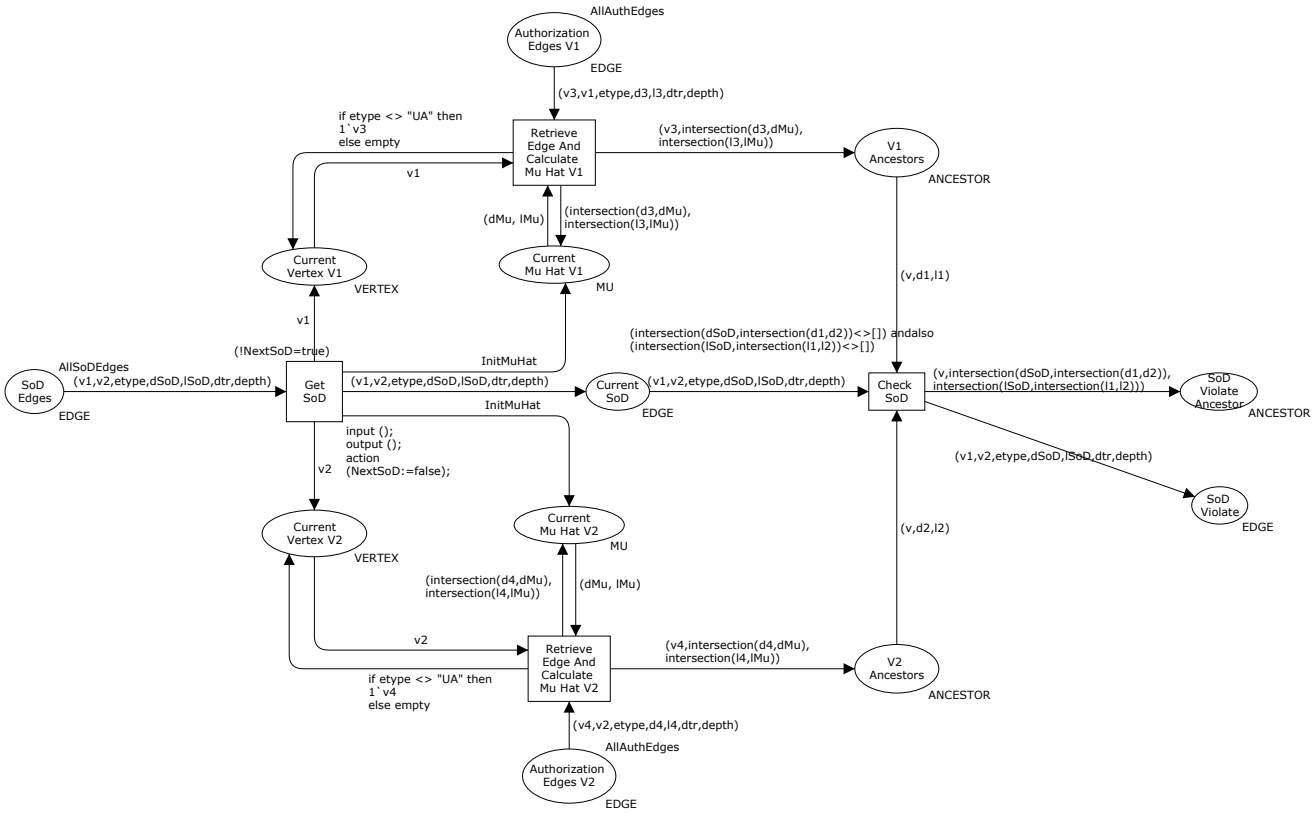

Figure 9: CPN Model for Separation of Duty Violation Detection

We then formulate a query to check for SoD violations. Our analysis reveals various SoD violations. For example, there is a SoD violation caused by assigning the role *State VC* two conflicting permissions p_{11} and p_{15}. Similarly, there is another SoD violation caused because role *State Epi* gets conflicting permissions p_{16} and p_{17}. Our analysis reveals that there is no SoD violation caused by any user being assigned conflicting roles.

5.6 Soundness and Completeness

The set of problems that we considered in this work are by no means exhaustive. For example, it is quite possible that the spatio-temporal constraints have been incorrectly specified but this error does not lead to isolated entities, infeasible path, SoD violation or delegation constraint violation and will not be detected. However, with respect to the problems that we do detect, we can make a few comments about the soundness and completeness. If the CPN model has been correctly constructed and populated using the access control graph, then we can prove soundness and completeness properties with respect to the given problem.

Consider, for example, the problem of detecting isolated users as shown in Figure 6. Let us recall how this CPN will operate. The initial marking *AllUsers* are populated using the user entities in the access control graph. Similarly, the initial marking *AllUserAssign* are initialized using *UA* and RD_U edges in the access control graph. The transition *Move Assigned User* will fire as long as some user u matches the user u in the edge $(u, v, etype, d1, l1, dtr, depth)$. This firing results in removing u and $(u, v, etype, d1, l1, dtr, depth)$ from $Users$ and *User Assign Edges* respectively and adding u to *Assigned Users*. When no more transitions can be fired, the terminal state has been reached and the place *Users* contain isolated users.

Suppose there is some isolated user u_i in the access control graph that is not detected by this CPN model. In other words, user u_i is not in the *Users* place when the terminal state is reached. This leads to two possibilities: either user u_i is in *Assigned Users* place in the terminal state or it is not. If u_i is in *Assigned Users*, then there exists an edge of type $(u_i, v, etype, d1, l1, dtr, depth)$ in the initial marking *AllUserAssign*. This is possible only if there is a corresponding *UA* or RD_U edge in the access control graph involving u_i; this, in turn, precludes u_i from being an isolated user. If u_i is not in *Assigned Users* in the terminal state and it is also not in *Users*, then u_i was not in the initial marking *AllUsers*. This is possible only if the access control graph does not contain u_i. Since both the cases are not possible, it means that user u_i must be in the *Users* places when the

terminal state is reached. Thus, all isolated users are detected by the CPN model.

Suppose u_j is detected as an isolated user by the CPN. This implies that u_j is in the place *Users* in the terminal state. In other words, there is no edge of the form $(u_j, v, etype, d1, l1, dtr, depth)$ in the initial marking *AllUserAssign*. In other words, there is no UA edge or RD_U edge associated with u_j in the access control graph. This implies that u_j is indeed an isolated user.

6 Improving the Analysis Performance

CPN explores the state space to check for violations of access control properties. Our investigations reveal that even a modest increase in the number of places and transitions cause a significant increment to the number of states of the state space. This state explosion problem increases the verification time substantially. In the worst case, if the number of states becomes too large, the model cannot be verified. Consequently, we need to find techniques for reducing the size of the CPN model.

We looked at the various CPN models that we generated for detecting problems with the access control specifications. We observed that the number of states generated in the CPN model were related to the number of edges traversed in the access control graph for detecting a specific problem. We looked at the number of states generated for each problem. For detecting delegation constraint violation, the number of states generated is of the order $O(|PD| + |RD|)$ where $|PD|$ and $|RD|$ represent the number of permission delegation edges and role delegation edges respectively. Since typically the number of delegation edges will be small, we did not think it necessary to produce further optimization. We next considered the problem of detecting infeasible paths. In this case, the number of states generated is of the order $O(|U||E| + |IP|)$ where $|U|$ is the number of users, $|E|$ is the number of edges in the access path, and $|IP|$ is the number of infeasible paths. Next, consider the problem of detecting SoD violations. Here, the number of states generated is of the order of $O(|SD||E| + |SoD|)$ where $|SD|$ is the number of SoD edges, $|E|$ is the number of edges in

the access path, and $|SoD|$ is the number of SoD violations. Thus, one way to reduce the number of states is to decrease the number of edges in the graph.

One way of reducing the number of edges is to flatten the hierarchy. We did some initial experiments in order to understand the effect of flattening the hierarchy on the state space. We created a very simple access control graph consisting of one user, one user-role assignment, one permission-role assignment, and multiple levels of hierarchy. With 10 levels of hierarchy the state space reduction was 40%, which is quite significant. This motivated us to transform the access control graph to a smaller graph, which we term, the *privilege acquisition graph*.

6.1 Privilege Acquisition Graph

In order to generate a smaller number of states in the CPN model that does efficient verification, we propose to transform the access control graph into the privilege acquisition graph. The privilege acquisition graph essentially flattens out the hierarchical structure.

It captures the following relationships: UA', PA', PO' and SD' where UA' represents the user-role assignment that occurs either directly or indirectly via hierarchy and delegation constraints, PA' represents permission-role assignment that occurs either directly or indirectly due to inheritance and delegation, PO' corresponds to the permission-object relationship (represented by PO in the access control graph), and SD' corresponds to separation of duty (represented by SD in the access control graph). Algorithm 1 shows the transformation process. Step 1 adds all the vertices of the access control graph to the privilege acquisition graph. Step 2 converts all the act-path in the access control graph to UA' edges in the privilege acquisition graph. Since the act-path consists of a sequence of activation hierarchies, the activation hierarchies are flattened out at this point. Step 3 converts all the u-path in the access control graph to PA' edges in the privilege acquisition graph. Since the u-path consists of a sequence of permission inheritance hierarchies, the permission inheritance hierarchies are flattened out in this step. Steps 4 and 5 adds all the PO, SD edges in

the access control graph to PO', SD' edges in the privilege acquisition graph respectively. The time complexity to generate the privilege acquisition graph is $O(VE)$, where V is the number of vertices and E is the number of edges of the original access control graph.

Theorem 1

The role authorizations and user authorizations are equivalent in the access control graph $G(V,E,\mu,\rho)$ and its corresponding privilege acquisition graph $G'(V',E',\mu')$.

Proof First, let us consider the case of role authorizations. Suppose role $v \in R$ is authorized for permission $v' \in P$ in the access control graph G. This is possible if there exists an u-path in the access control graph $(v, v_1, v_2, \ldots, v_n, v')$ and $\hat{\mu}(v,v') \neq \emptyset$. By step 3 of algorithm 1, if there exists an u-path $(v, v_1, v_2, \ldots, v_n, v')$ in the original graph G, it will be transformed to a PA' edge in its corresponding privilege acquisition graph G' with the same spatio-temporal constraint ($\mu'(v,v') = \hat{\mu}(v,v')$). Hence, for every u-path in G, there exists a PA' edge in G' that authorizes v to acquire permission v' at $\hat{\mu}(v,v')$ To show the converse, let us consider an edge $(v,v') \in PA'$ in G'. Since edge $(v,v') \in PA'$ in G' is created from some u-path in G, every role authorization in G' has a corresponding u-path in G that gives role v permission v' at the same spatio-temporal points. Thus, for every edge $(v,v') \in PA'$, there exists an u-path in G that gives role v permission v'.

Next, let us consider user authorizations. Let user $v \in U$ be authorized for permission $v' \in P$ with respect to object $v'' \in O$ in the access control graph G. This is possible if there exists an acs-path $(v, v_1, v_2, \ldots, v_i, \ldots, v', v'')$ such that $v_i \in R$ for some i, $(v_1, \ldots, v_i)$ is an act-path, $(v_i, \ldots, v')$ is an u-path, $(v', v'') \in PO$ and $\hat{\mu}(v,v'') \neq \emptyset$. Corresponding to this acs-path, the algorithm to generate the privilege acquisition graph creates three edges in Steps 2, 3, and 4. The edges created are $(v, v_i) \in UA'$ where $\mu'(v,v_i) = \hat{\mu}(v,v_i)$, $(v_i, v') \in PA'$ where $\mu'(v_i,v') = \hat{\mu}(v_i,v')$, and $(v', v'') \in PO'$ where $\mu'(v',v'') = \mu(v',v'')$. Moreover, $\mu'(v,v_i) \cap \mu'(v_i,v') \cap \mu'(v',v'') = \hat{\mu}(v,v'')$. These three edges (v,v_i), (v_i,v') and (v',v'') give the user v permission v' to access object v'' at points $\hat{\mu}(v,v'')$ in graph G'. To prove the converse, let us assume that the privilege acquisition graph G' provides some user

Algorithm 1 Transform access control graph to privilege acquisition graph

{**Input:** Access control graph $G(V,E,\mu,\rho)$}
{**Output:** Privilege acquisition graph $G'(V',E',\mu')$}
BEGIN
$V' \leftarrow \emptyset$
$E' \leftarrow \emptyset$
$\mu' \leftarrow \emptyset$
{**Step 1:** Add all vertices of the access control graph to the privilege acquisition graph}
for all $v \in V$ **do**
  $V' \leftarrow V' \cup v$
end for
{**Step 2:** Transform all act-path starting at each user vertex of the access control graph to the set of edges of the privilege acquisition graph (UA')}
for all $v \in U$ **do**
  for all act-path $act_i = (v,\ldots,v')$ **do**
    $E' \leftarrow E' \cup (v,v')$
    $\mu'(v,v') \leftarrow \hat{\mu}(v,v')$
    $\mu' \leftarrow \mu' \cup \mu'(v,v')$
  end for
end for
{**Step 3:** Transform all u-path starting at each role vertex and ending at the permission vertex of the access control graph to the set of edges of the privilege acquisition graph (PA')}
for all $(v \in R) \wedge (v' \in P)$ **do**
  for all u-path $u_i = (v,\ldots,v')$ **do**
    $E' \leftarrow E' \cup (v,v')$
    $\mu'(v,v') \leftarrow \hat{\mu}(v,v')$
    $\mu' \leftarrow \mu' \cup \mu'(v,v')$
  end for
end for
{**Step 4:** Add all PO edges from the access control graph to the set of edges of the privilege acquisition graph (PO')}
for all $(v,v') \in PO$ **do**
  $E' \leftarrow E' \cup (v,v')$
  $\mu'(v,v') \leftarrow \mu(v,v')$
  $\mu' \leftarrow \mu' \cup \mu'(v,v')$
end for
{**Step 5:** Add all SD edges from the access control graph to the set of edges of the privilege acquisition graph (SD')}
for all SD edge $sd_i = (v,v')$ **do**
  $E' \leftarrow E' \cup (v,v')$
  $\mu'(v,v') \leftarrow \mu(v,v')$
  $\mu' \leftarrow \mu' \cup \mu'(v,v')$
end for
Return $G'(V',E',\mu')$
END

u permission p for object o. This implies that there exists three edges of the $(u,r) \in UA'$, $(r,p) \in PA'$ and $(p,o) \in PO'$ and $(\mu'(u,r) \cap \mu'(r,p) \cap \mu'(p,o) \neq \emptyset$. The existence of these three edges is possible only if there is an act-path $(u, v_1, v_2, \ldots, v_n, r)$, an u-path $(r, v'_1, v'_2, \ldots, v'_m, p)$, an edge $(p,o) \in PO$ in the corresponding access control graph. Moreover, $\mu'(u,r) = \hat{\mu}(u,r)$, $\mu'(r,p) = \hat{\mu}(r,p)$, and $\mu'(p,o) = \mu(p,o)$. Thus, user u will get permission p to access object o at the same spatio-temporal points in graph G.

Lemma 1

Each isolated entity that exists in the access control graph G is also present in the corresponding privilege acquisition graph G' and vice-versa.

Proof Let u_i be an isolated user in the access control graph. This means that there is no act-path starting at u_i. Consequently, there is no edge in UA' in the privilege acquisition graph of the form (u_i, v). Since the edges in UA' are the only edges joining users to roles in the privilege acquisition graph, u_i is also an isolated entity in the privilege graph. Conversely, let u_j be an isolated user in the privilege acquisition graph. Thus, there is no edge of the form (u_j, v) in UA'. This is possible only if there is no act-path starting at u_j in the access control graph, which implies that u_j is an isolated entity in the access control graph. We can make similar arguments for isolated roles and permissions.

Lemma 2

For each infeasible path that exists in the access control graph G, there exists a corresponding infeasible path in the privilege acquisition graph G' and vice-versa.

Proof Let $P = (v_1, v_2, \ldots, v_n)$ be an infeasible path in the access control graph where $(v_1, \ldots, v_i)$ is an act-path, $(v_i, \ldots, v_{n-1})$ is an u-path and (v_{n-1}, v_n) be in PO. Since P is an infeasible path, $\hat{\mu}(v_1, v_n) = \hat{\mu}(v_1, v_i) \cap \hat{\mu}(v_i, v_{n-1}) \cap \hat{\mu}(v_{n-1}, v_n) = (d, l)$ where either $d = \emptyset$ or $l = \emptyset$. The construction of the privilege graph from this acquisition graph generates the following edges: (v_1, v_i), (v_i, v_{n-1}), and (v_{n-1}, v_n) where $(v_1, v_i) \in UA'$, $(v_i, v_{n-1}) \in PA'$ and $(v_{n-1}, v_n) \in PO'$ and $\mu'(v_1, v_i) = \hat{\mu}(v_1, v_i)$,

$\mu'(v_i, v_{n-1}) = \hat{\mu}(v_i, v_{n-1})$, and $\mu'(v_{n-1}, v_n) = \hat{\mu}(v_{n-1}, v_n)$. Thus, $\hat{\mu'}(v_1, v_i) = \hat{\mu}(v_1, v_n) = \hat{\mu}(v_1, v_i) \cap \hat{\mu}(v_i, v_{n-1}) \cap \hat{\mu}(v_{n-1}, v_n) = (d, l)$ where either $d = \emptyset$ or $l = \emptyset$. Thus, the path (v_1, v_i, v_{n-1}, v_n) is an infeasible path in the privilege acquisition graph. The converse can be similarly proved.

Lemma 3

For every SoD violation that exists in the access control graph G, there exists a corresponding SoD violation in the privilege acquisition graph G' and vice-versa.

Proof Suppose the access control graph has a SSoD role-permission violation of the form $(r_1, r_2, \ldots, r_n, p_i)$, $(r_1, r'_2, \ldots, r'_n, p_j)$ and (p_i, p_j) where $(r_1, r_2, \ldots, r_n, p_i)$, $(r_1, r'_2, \ldots, r'_n, p_j)$ are u-paths and (p_i, p_j) is a SoD edge and $\hat{\mu}(r_1, p_i) \cap \hat{\mu}(r_1, p_j) \cap \hat{\mu}(p_i, p_j) = (d, l)$ where $d \neq \emptyset$ and $l \neq \emptyset$. By construction, the following edges are generated for the privilege acquisition graph: (r_1, p_i) and (r_1, p_j) are edges in PA' and (p_i, p_j) is an edge in SD'. Since $\mu'(r_1, p_i) = \hat{\mu}(r_1, p_i)$, $\mu'(r_1, p_j) = \hat{\mu}(r_1, p_j)$, and $\mu'(p_i, p_j) = \hat{\mu}(p_i, p_j)$, we have $\mu'(r_1, p_i) \cap \mu'(r_1, p_j) \cap \mu'(p_i, p_j) = (d, l)$. Thus, the edges (r_1, p_i), (r_1, p_j) and (p_i, p_j) indicate there is a SoD violation. The converse can be proved in a similar manner. We can also prove the other types of SoD constraint violations similarly.

Theorem 2

The privilege acquisition graph accurately captures isolated entities, infeasible paths, and SoD violations.

Proof The proof follows from Lemmas 1, 2, and 3.

Note that, the privilege acquisition graph contains less information than the corresponding access control graph. For example, information about the role hierarchy is no longer present in the privilege acquisition graph. The CPN analysis of privilege acquisition graphs will detect the problems, but it may not have enough information to identify the source of the problem. Thus, if a problem exists, the access control graph or its subgraph related to the problem must be analyzed. For instance, if the analysis of the CPN corresponding to the privilege acquisition graph identifies

that there is an infeasible path (v_1, v_i, v_{n-1}, v_n), then to detect where the spatio-temporal constraints have been violated we need to find the subgraph of the access control graph involving these vertices and analyze it. Similarly, if the CPN analysis of the privilege acquisition graph reveals a potential SoD violation involving edges (r_1, p_i), (r_1, p_j), and (p_i, p_j), the corresponding subgraph of the access control graph must be analyzed to identify the source of the problem.

The subgraph can be generated by performing the reverse depth first search from both ends of the SoD edge in the original access control graph. The time complexity to generate the subgraph is $O(|V| + |E|)$. Once the subgraph is generated, we can replace the set of edges of the access control graph in the CPN model with the set of edges of the subgraph and perform the analysis in the same manner. This significantly reduces the analysis time because the size of the subgraph is substantially smaller.

6.2 DDS Example Privilege Acquisition Graph

We use Algorithm 1 to transform the access control graph of the DDS system into the privilege acquisition graph, shown in Figure 10. The new spatio-temporal constraints can be calculated from the $\hat{\mu}(v, v')$ function as described in Algorithm 1. For instance, $\mu(\textit{State Epi}, p_1)$ in the condensed graph can be calculated from $\hat{\mu}(\textit{State Epi}, p_1)$ of the original access control graph, which equals to $\mu(\textit{State Epi}, \textit{Juris Epi}) \cap \mu(\textit{Juris Epi}, p_1) = [b, B] \cap [a, B] = [b \cap a, B \cap B] = [a, B]$. Note that in this example duration b means *Always*, hence, $b \cap a = a$. We compute other spatio-temporal constraints in the same manner. All new spatio-temporal constraints are shown in Table 4.

6.2.1 Problem Detection using Privilege Acquisition Graph

In this section, we show how to detect infeasible paths and separation of duty violations using our modified approach.

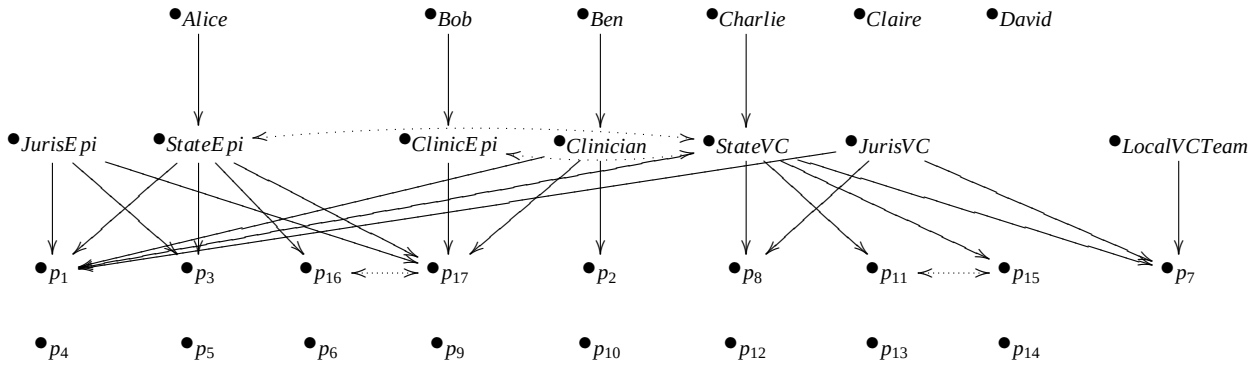

Figure 10: DDS System's Privilege Acquisition Graph

Infeasible Path Detection

We define the types in the model using colorset as shown in Section 5. We use the privilege acquisition graph instead of the access control graph to populate the initial markings of our previous CPN model shown in Figure 7. The initial marking for *Authorization Edges*, denoted as *AllAuthEdges* is populated by the UA' and the PA' edges of the privilege acquisition graph. The rest of the initial markings for other places are the same as before.

We allow the execution of this model and run the queries to detect infeasible paths. The analysis result shows that the system contains infeasible path. The query shows that a set of states $\{42, 43\}$ suffers from the infeasible path. To check this, we use the *print* command to check the descriptor (environmental variables) of the state. For instance, let us check the state 43. Below is part of the content of state 43.

```
Authorization Path = 1‘("Ben","Clinician","UA",["a"], ["C"],"",0)++
1‘("Clinician","p17","PA",["c"], ["C"],"",0);
```

The result shows that the infeasible path occurs because user *Ben* cannot acquire p_{17} assigned to him via the *Clinician* role. The percentage reduction in the number of states when using the privilege acquisition graph instead of the access control graph is only 8 percent in this case.

In this analysis, we do not have enough information about how *Ben* was assigned the *Clinician* role, whether through direct assignment or indirect assignment by hierarchy or delegation. If we are interested in knowing the source of conflict, we have to verify the original graph. However, since we know that only p_{17} causes the problem, we can bypass the verification of other irrelevant entities. To do this, from the access control graph, we create a subgraph consists of all entities related with p_{17} by performing a reverse depth first search starting from p_{17}. The subgraph derived from the access control graph is shown in Figure 11.

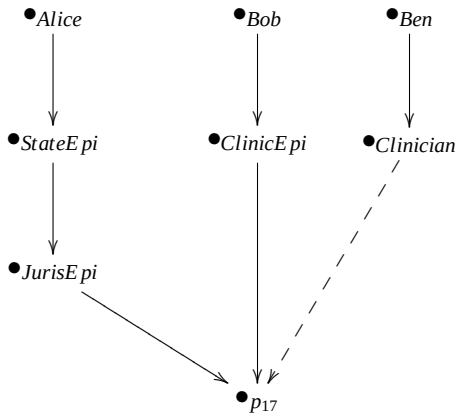

Figure 11: Subgraph of the related entities of p_{17}

We then run the same CPN model for the derived subgraph, which is similar to Figure 2. We observe the state variable, which shows that p_{17} is the delegated permission which has temporal conflict with role *Clinician*.

SoD Violation Detection

We run our previous CPN model for detecting the SoD violation shown in Figure 9 on the privilege acquisition graph. We then create the state space graph and execute the query to detect conflicts. The percentage reduction in the number of states obtained by using the privilege acquisition graph is 25 percent. The tools return a list of possible conflict states $\{38, 46, 48, 50, 53, 55, 56, 57, 58, 59\}$.

We run the print command to show the value of environmental variables of state number 46. Below is part of the content of state 46 which shows that the conflict occurs between p_{16} and p_{17}.

```
SoD Violate Ancestor = 1‘("State Epi",["a"], ["B"]);
SoD Violate = 1‘("p16","p17","PSSD",["a"], ["A", "B", "C", "D", "E"], "", 0);
```

Since CPNs based on the privilege acquisition graph can detect conflicts but not identify the source, we create a subgraph from the access control graph by performing a reverse depth first search starting from node for p_{16} and then for p_{17}. The resulting subgraph is shown in Figure 12. This subgraph can be analyzed as described in Section 5 to reveal the source of conflict. Since the subgraph is much smaller than the original access control graph, it will take significantly less time. We then run the model again on the derived subgraph. This time the model indicates that p_{17} is the inherited permission which together with the assigned permission p_{16} of role *State Epi* have violated the SSoD for permission-role assignment.

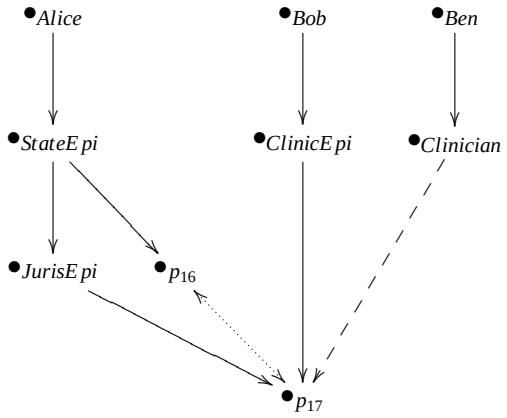

Figure 12: Subgraph of the related entities of p_{16} and p_{17}

7 Conclusion and Future Work

Traditional access control models base their authorization decisions solely on the attributes of the user (identity, security level, or role). Since they do not take into account environmental factors, such as time and location, in making access decisions, they may not be very suitable for mobile computing or pervasive computing applications. Towards this end, we propose a spatio-temporal role-based access control model where authorization decisions depend on the role of the user and other spatio-temporal constraints.

We investigated how the various entities and relationships in RBAC may be impacted by time and location and describe how the traditional RBAC can be enhanced by spatio-temporal constraints. The various features of the model are expressed using logical constraints and the formal semantics are specified using a graph-theoretic notation. The various features of the model may interact with each other in subtle ways resulting in conflicts and other inconsistencies. Consequently, we need to analyze the access control constraints of the application to ensure that such problems do not occur. Since manual analysis is tedious and error-prone, we show how the analysis can be automated using Coloured Petri Nets. For large complex applications, the analysis may take a significant amount of time. Towards this end, we show how to speed up the analysis by condensing the graph representing the application and verifying this condensed graph.

In this work, we have made some simplifying assumptions. We have assumed that the precise locations of subjects and objects are known at any given point of time. However, in practice this may not be feasible. Specifically, as pointed out by Shin and Atluri [43], approximate locations are maintained in mobile environments to minimize the updates. Some related works [1, 43] exist on how to use imprecise location data to make access decisions – we plan to incorporate some of these approaches in our future work. A similar problem exists with representing temporal information. In a future work, we plan to provide a more realistic representation of time and location in our models; this, in turn, will necessitate the use of alternative tools for analyzing problems with access control

specifications.

Since pervasive computing applications will typically be modeled as dynamic workflows, we need to augment our model to support them. The new model must be analyzed to ensure that authorization constraints and the various workflow dependencies do not give rise to conflicts and the workflow can execute correctly and complete. We also plan to implement our model. Implementation will require us to investigate additional issues, such as, how to store location and temporal information and perform operations involving spatio-temporal constraints in an efficient manner. Once we have an implementation, we plan to validate our model using the example real-world application for the dengue decision support system. Implementing the model for real-world applications will further help refine our model and make it more useful.

Acknowledgement

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

References

[1] C. A. Ardagna, M. Cremonini, E. Damiani, S. D. C. di Vimercati, and P. Samarati. Supporting Location-Based Conditions in Access Control Policies. In *Proceedings of the ACM Symposium on Information, Computer and Communications Security*, pages 212–222, Taipei, Taiwan, March 2006.

[2] V. Atluri and S. A. Chun. An Authorization Model for Geospatial Data. *IEEE Transactions on Dependable and Secure Computing*, 1(4):238–254, October-December 2004.

[3] V. Atluri and S. A. Chun. A Geotemporal Role-based Authorisation System. *International Journal of Information and Computer Security*, 1(1/2):143–168, January 2007.

[4] V. Atluri and W.-K. Huang. An Authorization Model for Workflows. In *Proceedings of the 4th European Symposium on Research in Computer Security*, pages 44–64, Rome, Italy, September 1996.

[5] V. Atluri and W.-K. Huang. A Petri Net Based Safety Analysis of Workflow Authorization Models. *Journal of Computer Security*, 8(2,3):209–240, August 2000.

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

[7] E. Barka and R. Sandhu. Framework for Role-Based Delegation Models. In *Proceedings of the 16th Annual Computer Security Applications Conference*, pages 168–176, New Orleans, LA, USA, December 2000.

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

[9] E. Bertino, P. A. Bonatti, and E. 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.

[10] E. Bertino, B. Catania, M. L. Damiani, and P. 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.

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

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

[13] M. J. Covington, P. Fogla, Z. Zhan, and M. 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.

[14] M. J. Covington, W. Long, S. Srinivasan, A. Dey, M. Ahamad, and G. 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.

[15] J. Crampton and H. Khambhammettu. Delegation in Role-Based Access Control. *International Journal of Information Security*, 7(2):123–136, March 2008.

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

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

[18] 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, New York, NY, USA, 2005.

[19] K. Jensen. *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use*. Springer-Verlag, New York, NY, USA, 1997.

[20] K. Jensen, S. Christensen, and L. M. Kristensen. CPN Tools State Space Manual, 2006. http://wiki.daimi.au.dk/cpntools-help/_files/manual.pdf.

[21] K. Jensen and L. M. Kristensen. *Coloured Petri Nets: Modelling and Validation of Concurrent Systems*. Springer, 2009.

[22] K. Jensen, L. M. Kristensen, and L. Wells. Coloured Petri Nets and CPN Tools for Modelling and Validation of Concurrent Systems. *International Journal on Software Tools for Technology Transfer*, 9(3):213–254, May 2007.

[23] Y. Jiang, C. Lin, H. Yin, and Z. Tan. Security Analysis of Mandatory Access Control Model. In *Proceedings of IEEE International Conference on Systems, Man and Cybernetics*, pages 5013–5018, The Hague, The Netherlands, October 2004.

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

[25] J. B. D. Joshi and E. Bertino. 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*, pages 81–90, Lake Tahoe, California, USA, June 2006.

[26] J. B. D. Joshi, E. Bertino, A. Ghafoor, and Y. Zhang. Formal Foundations for Hybrid Hierarchies in GTRBAC. *ACM Transactions on Information and System Security*, 10(4):1–39, January 2008.

[27] R. Laborde, B. Nasser, F. Grasset, F. Barrère, and A. Benzekri. A Formal Approach for the Evaluation of Network Security Mechanisms Based on RBAC Policies. *Electronic Notes in Theoretical Computer Science*, 121:117–142, February 2005.

[28] U. Leonhardt and J. Magee. Security Consideration for a Distributed Location Service. *Journal of Network and Systems Management*, 6(1), 1998.

[29] Y. Lu, L. Zhang, and J. Sun. Using Coloured Petri Nets to Model and Analyze Workflow with Separation of Duty Constraints. *The International Journal of Advanced Manufacturing Technology*, 40(1,2):179–192, January 2009.

[30] R. Milner, M. Tofte, R. Harper, and D. Macqueen. *The Definition of Standard ML - Revised*. The MIT Press, May 1997.

[31] F. Pu, D. Sun, Q. Cao, H. Cai, and F. Yang. 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*, pages 689–692, Pasadena, CA, USA, December 2006.

[32] J. L. Rasmussen and M. Singh. Designing a Security System by Means of Coloured Petri Nets. In *Proceedings of the 17th International Conference on Application and Theory of Petri Nets*, pages 400–419, London, UK, June 1996. Springer-Verlag.

[33] I. Ray and M. Kumar. Towards a Location-Based Mandatory Access Control Model. *Computers & Security*, 25(1):36–44, February 2006.

[34] I. Ray, M. Kumar, and L. 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.

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

[36] 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, USA, July 2007.

[37] I. Ray and M. Toahchoodee. 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*, pages 48–58, Turin, Italy, September 2008.

[38] G. Sampemane, P. Naldurg, and R. 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.

[39] A. Samuel, A. Ghafoor, and E. 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.

[40] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-Based Access Control Models. *IEEE Computer*, 29(2):38–47, February 1996.

[41] A. Schaad and J. D. Moffett. A Lightweight Approach to Specification and Analysis of Role-Based Access Control Extensions. In *Proceedings of the 7th ACM Symposium on Access Control Models and Technologies*, pages 13–22, Monterey, CA, USA, June 2002.

[42] B. Shafiq, J. B. D. Joshi, and A. Ghafoor. Petri-Net Model for Verification of RBAC Policies. Technical report, Purdue University, 2002.

[43] H. Shin and V. Atluri. Spatiotemporal Access Control Enforcement under Uncertain Location Estimates. In *Proceedings of the 23rd Annual IFIP WG 11.3 Working Conference on Data and Applications Security*, pages 159–174, Montreal, P.Q., Canada, 2009.

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

[45] M. Toahchoodee and I. Ray. 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*, pages 17–32, London, UK, July 2008.

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

[47] M. Toahchoodee, I. Ray, K. Anastasakis, G. Georg, and B. Bordbar. Ensuring Spatio-Temporal Access Control for Real-World Applications. In *Proceedings of the 14th ACM Symposium on Access control Models and Technologies*, pages 13–22, Stresa, Italy, June 2009.

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

[49] C. Yuan, Y. He, J. He, and Z. Zhou. 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*, pages 196–210, Beijing, China, November 2006.

[50] J. Zao, H. Wee, J. Chu, and D. Jackson. RBAC Schema Verification Using Lightweight Formal Model and Constraint Analysis, 2002. http://alloy.mit.edu/publications.php.

[51] L. Zhang, G.-J. Ahn, and B.-T. Chu. A Rule-Based Framework for Role-Based Delegation and Revocation. *ACM Transactions on Information and System Security*, 6(3):404–441, August 2003.

[52] X. Zhang, S. Oh, and R. Sandhu. PBDM: A Flexible Delegation Model in RBAC. In *Proceedings of the 8th ACM Symposium on Access Control Models and Technologies*, pages 149–157, Como, Italy, June 2003.

NAME	DESCRIPTION	CONSTRAINTS	
		μ	ρ
$(Alice, State\ Epi)$	User-Role Assignment	$[b, A \cup B]$	
$(Bob, Clinic\ Epi)$	User-Role Assignment	$[b, C]$	
$(Ben, Clinician)$	User-Role Assignment	$[a, C]$	
$(Charlie, State\ VC)$	User-Role Assignment	$[a, A \cup B]$	
$(State\ Epi, Juris\ Epi)$	Permission Inheritance Hierarchy	$[b, B]$	
$(State\ VC, Juris\ VC)$	Permission Inheritance Hierarchy	$[a, B]$	
$(Juris\ VC, Local\ VC\ Team)$	Permission Inheritance Hierarchy	$[a \cup c, E]$	
$(State\ Epi, p_{16})$	Permission-Role Assignment	$[a, A \cup B]$	
$(Juris\ Epi, p_1)$	Permission-Role Assignment	$[a, B]$	
$(Juris\ Epi, p_3)$	Permission-Role Assignment	$[a, B]$	
$(Juris\ Epi, p_{17})$	Permission-Role Assignment	$[b, B]$	
$(Clinic\ Epi, p_{17})$	Permission-Role Assignment	$[b, D]$	
$(Clinician, p_1)$	Permission-Role Assignment	$[a, C]$	
$(Clinician, p_2)$	Permission-Role Assignment	$[a, C]$	
$(State\ VC, p_{11})$	Permission-Role Assignment	$[a, A]$	
$(State\ VC, p_{15})$	Permission-Role Assignment	$[a, A]$	
$(Juris\ VC, p_1)$	Permission-Role Assignment	$[a, B]$	
$(Juris\ VC, p_8)$	Permission-Role Assignment	$[a, B]$	
$(Local\ VC\ Team, p_7)$	Permission-Role Assignment	$[a \cup c, E]$	
$(Clinician, p_{17})$	R2R Permission Delegation	$[c, C]$	[*Clinic Epi*, 1]
$(State\ Epi, State\ VC)$	Role Static SoD	$[b, D]$	
$(State\ Epi, Juris\ VC)$	Role Static SoD	$[b, D]$	
$(Juris\ Epi, State\ VC)$	Role Static SoD	$[b, D]$	
$(Juris\ Epi, Juris\ VC)$	Role Static SoD	$[b, D]$	
$(Clinic\ Epi, State\ VC)$	Role Static SoD	$[b, D]$	
$(Clinic\ Epi, Juris\ VC)$	Role Static SoD	$[b, D]$	
(p_{11}, p_{15})	Permission Static SoD	$[a, D]$	
(p_{16}, p_{17})	Permission Static SoD	$[a, D]$	

Table 3: DDS Relationships and Constraints

NAME	DESCRIPTION	SPATIO-TEMPORAL DOMAIN (μ)
$(Alice, State\ Epi)$	User-Role Authorization	$[b, A \cup B]$
$(Bob, Clinic\ Epi)$	User-Role Authorization	$[b, C]$
$(Ben, Clinician)$	User-Role Authorization	$[a, C]$
$(Charlie, State\ VC)$	User-Role Authorization	$[a, A \cup B]$
$(State\ Epi, p_1)$	Permission-Role Authorization	$[a, B]$
$(State\ Epi, p_3)$	Permission-Role Authorization	$[a, B]$
$(State\ Epi, p_{16})$	Permission-Role Authorization	$[a, A \cup B]$
$(State\ Epi, p_{17})$	Permission-Role Authorization	$[b, B]$
$(Juris\ Epi, p_1)$	Permission-Role Authorization	$[a, B]$
$(Juris\ Epi, p_3)$	Permission-Role Authorization	$[a, B]$
$(Juris\ Epi, p_{17})$	Permission-Role Authorization	$[b, B]$
$(Clinic\ Epi, p_{17})$	Permission-Role Authorization	$[b, D]$
$(Clinician, p_1)$	Permission-Role Authorization	$[a, C]$
$(Clinician, p_2)$	Permission-Role Authorization	$[a, C]$
$(Clinician, p_{17})$	Permission-Role Authorization	$[c, C]$
$(State\ VC, p_1)$	Permission-Role Authorization	$[a, B]$
$(State\ VC, p_7)$	Permission-Role Authorization	$[a, \emptyset]$
$(State\ VC, p_8)$	Permission-Role Authorization	$[a, B]$
$(State\ VC, p_{11})$	Permission-Role Authorization	$[a, A]$
$(State\ VC, p_{15})$	Permission-Role Authorization	$[a, A]$
$(Juris\ VC, p_1)$	Permission-Role Authorization	$[a, B]$
$(Juris\ VC, p_7)$	Permission-Role Authorization	$[a \cup c, E]$
$(Juris\ VC, p_8)$	Permission-Role Authorization	$[a, B]$
$(Local\ VC\ Team, p_7)$	Permission-Role Authorization	$[a \cup c, E]$
$(State\ Epi, State\ VC)$	Role Static SoD	$[b, D]$
$(State\ Epi, Juris\ VC)$	Role Static SoD	$[b, D]$
$(Juris\ Epi, State\ VC)$	Role Static SoD	$[b, D]$
$(Juris\ Epi, Juris\ VC)$	Role Static SoD	$[b, D]$
$(Clinic\ Epi, State\ VC)$	Role Static SoD	$[b, D]$
$(Clinic\ Epi, Juris\ VC)$	Role Static SoD	$[b, D]$
(p_{11}, p_{15})	Permission Static SoD	$[a, D]$
(p_{16}, p_{17})	Permission Static SoD	$[a, D]$

Table 4: New Relationships and Constraints