Specification, Validation, and Enforcement of a Generalized Spatio-Temporal Role-Based Access Control Model

Ramadan Abdunabi, Mustafa Al-Lail, Indrakshi Ray, *Senior Member, IEEE,* and Robert B. France

***Abstract*—With the advent of wireless and mobile devices, many new applications are being developed that make use of the spatio-temporal information of a user to provide better functionality. Such applications also necessitate sophisticated authorization models where access to a resource depends on the credentials of the user and also on the location and time of access. Consequently, researchers have extended the traditional access control models, such as role-based access control, to provide spatio-temporal access control. We improve upon these models by providing additional features that allow us to express constraints that were not possible until now. We express our model using the unified modeling language (UML) and the object constraint language that are the *de facto* specification languages used by the industry. Our model has numerous features that interact in subtle ways. To this end, we show how the UML-based specification environment tool can be used to analyze the spatio-temporal access control model of an application. We propose an architecture for enforcing our model and provide a protocol that demonstrates how access control can be granted and revoked in our approach. We also develop a prototype of this architecture to demonstrate the feasibility of our approach.**

***Index Terms*—Access control.**

I. Introduction

WITH THE GROWTH of the sensor and wireless technology, new applications are being developed for mobile devices. Such applications typically have new authorization requirements where environmental conditions, such as location and time, are used together with the credentials of the user to determine access. An example will help illustrate this point. Consider a real-world example of a spatio-temporal policy for the telemedicine application iMediK [1]. The iMedik is a mobile application accessible by handheld devices that are integrated with a global positioning system (GPS) that identifies its physical location. With the help of mobile devices, doctors can access their patient information on the move. The security policy requires that doctors can use handheld devices to view complete patient medical record (PMR) information in the clinic during daytime, whereas the same doctors can view only partial PMR information outside the clinic during nighttime. Such a policy is needed to protect patient-sensitive information in the case of lost or stolen devices. Traditional access control models, such as role-based access control (RBAC), cannot be used for expressing such policies.

Researchers have addressed this problem by extending RBAC that allows it to do spatio-temporal access control [2]–[4]. Most of the work on spatio-temporal RBAC associate two entities, namely, location and time with users, roles, and permissions. The location and time associated with a user gives the current time and his present location. The location and time associated with a role designate when and where the role can be activated. The location and time associated with a permission signify when and where a permission can be invoked. In addition, researchers have also suggested how spatio-temporal constraints can be associated with inheritance and separation of duty (SoD) relations. Our current work extends the earlier works along several dimensions. First, we provide a more expressive spatio-temporal access control model that we refer to as generalized spatio-temporal role-based access control (GSTRBAC). We allow spatio-temporal constraints to be specified with various types of prerequisite constraints. We also introduce the concept of spatio-temporal zone that allows us to abstract location and time into one entity. This, in turn, reduces the number of entities that must be managed and also prevents the creation of new roles or permissions when spatio-temporal constraints associated with them change.

Second, we demonstrate how our model can be formally represented using the unified modeling language (UML) and object constraint language (OCL) that are the *de facto* specification languages used in the software industry.

Third, we illustrate how the model can be automatically analyzed using the UML-based specification environment (USE) tool for consistency and correctness. An application using our model can also be analyzed using USE to check for security property violations and see how the properties are affected when the security policy is changed.

Fourth, we provide an architecture for enforcing our spatio-temporal access control model. We also provide communication protocols that demonstrate how access can be granted and revoked in the context of our model and prove the security of this protocol. Finally, we implement a prototype and show how our spatio-temporal RBAC can be used in an Android mobile application.

Manuscript received October 9, 2011; revised April 12, 2012; accepted November 3, 2012.

The authors are with the Computer Science Department, Colorado State University, Fort Collins, CO 80523 USA (e-mail: rabdunab@cs.colostate.edu; mustafa@cs.colostate.edu; iray@cs.colostate.edu; france@cs.colostate.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/JSYST.2013.2242751

Treating location and time as separate entities often create problems. Let us illustrate this with an example. Suppose a doctor role can be activated at locations {hospital, clinic} from 8:00 a.m. to 5:00 p.m. This means that the doctor can activate his role either in the hospital or a clinic anytime from 8:00 a.m. to 5:00 p.m. Suppose that the medical board decides to change the spatio-temporal constraints such that the doctor can activate his role in the hospital from 8:00 a.m. to 1:00 p.m. and can activate his role in the clinic from 12:00 p.m. to 5:00 p.m. In order to specify such a constraint, we would have to split the doctor role into two roles, namely, hospital doctor and clinic doctor and associate the respective location and temporal constraints with each of them. Thus, a simple change to the spatio-temporal constraint requires the creation of new roles and changing all the relationships that are associated with the original role. Such a change is nontrivial. Treating location and time as distinct entities also causes a significant increase in the number of entities to be managed as location and time are associated with every object and relation in RBAC. This not only reduces ease of understanding for the user but also makes automated verification more challenging due to state-space explosion.

We introduce the concept of a spatio-temporal zone, henceforth, referred to as STZone. The STZone entity in our model is an abstract concept associated with each RBAC entity and relationship. STZone is represented as a set of pairs for locations and intervals that are of interest to the RBAC entities. In the previous example, the doctor role is initially associated with the following STZones: {(hospital, [8:00 a.m. to 5:00 p.m.]), (clinic, [8:00 a.m. to 5 p.m.])}. When the medical board decides to change the policy, this can be effectuated just by changing the STZones associated with the doctor role as follows: {(hospital, [8:00 a.m. to 1:00 p.m.]), (clinic, [12:00 p.m. to 5 p.m.])}. Abstracting location and time into a single STZone also reduces the number of entities in the model, making it easier to understand and verify.

We formalize GSTRBAC using UML [5] and OCL [6]. A number of reasons motivated our choice. First, UML is a general-propose language that has been considered the *de facto* standard in modeling software. Thus, applications are likely to be specified in UML. This will make it easier for one to integrate the access control policies with the application. Second, UML has a set of graphical notations for specifying static and dynamic aspects of software systems. The graphical diagrams of the UML make it easy to understand and use. Third, UML has supporting tools [7] that can be used for automated analysis. Fourth, UML can be used in all the phases of the software development process. Thus, it will be easy to check whether an access control implementation satisfies the policy if both are specified using the same language.

Once we have specified the GSTRBAC model, it must be analyzed for consistency and correctness. Moreover, for an application using GSTRBAC, we need to ensure that no access control breaches or problems occur. Toward this end, we need to do some automated analysis. Earlier works that use UML to specify access control requirements have typically resorted to the use of other formalisms for automated analysis. Such an approach typically involves a transformation process where the UML is converted into Alloy [8], Coloured Petri Nets [9], or UPPAAL [10] for the purpose of analysis. The results of the analysis depend on the correctness of the transformation procedure. We do not follow this approach but utilize USE [7] for the analysis that allows us to use the same language for specification and verification of GSTRBAC. The USE tool supports the automated generation of snapshot instances, which is used to validate GSTRBAC policies. The USE tool provides an interactive environment that facilitates the validation of properties of UML models specified in the form of OCL invariants, preconditions, and postconditions against some test scenarios. Such test scenarios are automatically generated by the USE tool that makes the verification process easier. The verification is carried out by an embedded constraint solver.

The rest of this paper is organized as follows. Section II enumerates some related work in this area. Section III introduces the concept of spatio-temporal zones. Section IV presents our GSTRBAC model using UML and OCL. Section V shows how USE can be used for analyzing the access control requirements of an application specified using GSTRBAC. In the presentation of the GSTRBAC enforcement mechanism, Section VI discusses the proposed implementation architecture model, Section VII introduces the resource usage protocols, and Section VIII describes an experimental evaluation of a prototype enforcing GSTRBAC in an Android mobile application. Section IX concludes this paper with pointers to future directions.

II. Related Work

RBAC [11] is the de-facto access control model used in the commercial sector. RBAC is policy neutral and can be used to express different types of policies. RBAC simplifies security management. In RBAC, users are assigned to roles, and roles are associated with permissions. In a session, a user can activate a subset of roles assigned to him. The operations that a user can perform in a session depend on the roles activated by the user and the permissions associated with those roles. To simplify role management, roles are organized in the form of a hierarchy. A senior role may inherit the permissions of a junior role, or a senior role may activate the junior role depending on whether the roles are connected by permission inheritance hierarchy or role activation hierarchy. In order to prevent fraud, RBAC allows one to specify separation of duty constraints. Static separation of duty constraints prevents a user from being assigned to conflicting roles or a role being associated with conflicting permissions. Dynamic separation of duty constraints prevent a user from activating two conflicting roles.

Researchers have worked on extending RBAC with time and location. Bertino *et al.* [12] proposed a temporal RBAC. The authors introduced the concept of role enabling and disabling. Roles can be enabled or disabled on the basis of temporal constraints. Roles can be activated if they are enabled. The model does not consider the impact of temporal constraints on user-role assignment or permission-role assignment. This also does not consider the effect of time on separation of duty constraints, cardinality constraints, and role–role hierarchy. Joshi *et al.* [13] proposed a generalized temporal RBAC model that associates temporal constraints with the entities and all the relationships in an RBAC model.

Researchers have also extended RBAC using location information. Hansen and Oleshchuk [14] proposed the spatial RBAC for specifying location-based access control policies for wireless networks. Bertino *et al.* [15] proposed the GEO-RBAC model that allows role activation based on users' locations. However, the model does not discuss the impact of spatial constraints on role hierarchy, separation of duty, user-role assignment, and permission-role assignment. Ray *et al.* [16] proposed location-aware RBAC (LRBAC) model incorporates location constraints in user-role activation, user-role, and permission-role assignments. LRBAC does not define spatial constraints on role hierarchy or separation of duty.

The use of both spatial and temporal information for doing access control has also been investigated by many researchers [2]–[4], [17]–[19]. In all these models, the role activation is constrained by spatio-temporal information. In some models [4], [17], [19], additional spatio-temporal constraints are imposed on user-role assignment and permission-role assignment. Some of these models [17]–[19] also consider the impact of spatio-temporal constraints on role hierarchy and separation of duty. Others [17], [19] also put additional spatio-temporal constraints on permissions. Our current model allows the specification of all the above types of spatio-temporal constraints. In addition, we allow spatio-temporal constraints to be specified with prerequisite constraints—the impact of location and time on prerequisite constraints has not been discussed in any of the previous works; we do this in this paper. We also introduce the concept of STZone that abstracts the location and time into one entity. This simplifies policy management and policy analysis.

Chen and Crampton develop the graph-based representation for the spatio-temporal RBAC in [17]. The RBAC entities are represented by vertices, while their relationships are represented by the edges of a directed graph. The authors propose three types of models: standard, strong, and weak. For 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. For 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 authors developed strong and clear semantics of these different models.

Our work differs from that of Chen and Crampton [17] in the following ways. First, we consider the spatio-temporal impact on separation of duty and prerequisite constraints which is missing from the work of Chen and Crampton. They also did not consider moving objects, which we do in this paper. Second, Chen and Crampton consider the spatial and temporal domains separately before developing the spatio-temporal point. Treating the two domains separately gives rise to the problems described in Section I. Third, Chen and Crampton add spatio-temporal constraints to the RBAC entities and relationships using λ and μ functions, respectively. This approach makes it harder to capture the number, types, and the relationship between the various spatio-temporal constraints. On the other hand, we consider STZone as a separate entity along with the other existing RBAC entities. This allows a more uniform treatment; the STZone pertinent to the application is enumerated and their relationships can easily be evaluated. Fourth, Chen and Crampton provide a graph-theoretic approach for visualizing problems with the specification, but do not focus on the analysis. We use UML and OCL for this purpose; UML and OCL are the *de facto* language for specifying the various requirements of the applications. This makes it easier to analyze the interactions among the access control constraints and also how these impact other application requirements.

In addition to specifying novel RBAC models, researchers have proposed approaches for verifying RBAC security policies. Some researchers [4], [20]–[22] have investigated the use of Alloy for verifying spatio-temporal RBAC policies. In order to make the analysis tractable, Alloy requires that the user scope the problem. The results of the analysis are, therefore, applicable only for the scope of the problem being verified. Modeling and analyzing concurrency in Alloy is nontrivial. Toward this end, researchers [19], [23], [24] have investigated alternative techniques based on Coloured Petri Nets [9] and timed automata [10] for verifying temporal, spatio-temporal, and real-time RBAC policies. The major challenge in these works is how to do the analysis without causing the problem of state explosion. The use of UML and OCL for specification and analysis of RBAC has been investigated by Sohr *et al.* [25]. Our work extends this for specifying and analyzing spatio-temporal policies.

Researchers have also proposed other temporal authorization models that are not based on RBAC. Bertino *et al.* [26] proposed a temporal authorization model that extends authorizations with temporal constraints. In this model, an authorization is associated with a temporal expression, identifying the periods of time in which the authorization applies. Furthermore, it also permits the specification of derivation rules for expressing temporal dependencies among authorizations. Gal and Atluri [27] proposed a temporal data authorization model (TDAM) that can be seen as a complementary model to the one in [26]. TDAM can express time-based policies based on the temporal attributes of data such as transaction time. However, these models are for non-RBAC policies; they cannot express temporal constraints on roles such as temporal role enabling and disabling constraints.

III. Location and Time Representation

In our model, each entity and relation is associated with spatio-temporal information. Before describing these associations in detail, we show how spatio-temporal information is represented in our model.

A. *Location Representation*

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 3-D geometric space. A physical location $ploc_i$ is a nonempty set of points $\{p_i, p_j, \ldots, p_n\}$, where a point p_k is represented by three coordinates. The granularity of each coordinate 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, CO. 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 corresponding physical one.

Definition 1: **[Mapping Function m]** m is a total function that converts a logical location into a physical one. Formally, $m : L \longrightarrow P$, where P is the set of all possible physical locations and L is the set of all logical locations.

We define the *containment* $\subseteq$ and *equality* = on physical locations. A physical location $ploc_j$ is said to be contained in another physical location $ploc_k$ if $ploc_j \subseteq ploc_k$. Two physical locations $ploc_r$ and $ploc_s$ are equal if $ploc_r \subseteq ploc_s$ and $ploc_s \subseteq ploc_r$.

Note that 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 *anywhere* that contains all other locations. Each application can describe logical locations at different granularity levels. For example, some permissions may be applicable on the entire state whereas other permissions are only applicable to people in the city. Let us denote the logical locations that are of interest to the application by the set **L**. Let the physical locations corresponding to these logical locations be denoted by **P**. The size of the smallest location in **P** corresponds to the minimal location granularity of the application. For example, in the organization Software Development Corporation, we may have **L** = {*MainBuilding, TestingOffice, DirectorOffice, DevelopmentOffice*}. The *MainBuilding* houses the three offices in separate floors of the building. In this case, the minimal location granularity is one floor.

B. Time Representation

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. A time interval is a set of consecutive time instants that can be represented in the form of $d = [t_s - t_e]$, where t_s, t_e represent time instants and t_s precedes t_e on the time line if $t_s \not= t_e$. We use the notation $t_i \in d$ to mean that t_i is a time instant in the time interval d. The exact granularity of a time instant is application dependent. Suppose the granularity of time instant in an application is 1 min. In this case, time interval [3:00 a.m. to 4:00 a.m.] consists of the set of time instants {3:00 a.m., 3:01 a.m., 3:02 a.m., ..., 3:59 a.m., 4:00 a.m.}.

Now, we define the containment $\subseteq$ and equality = on time intervals. A time interval d_j is said to be contained in another time interval d_k if $d_j \subseteq d_k$. 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. The set of all time intervals of interest to the application is defined by **I**. The minimal time granularity of an application refers to the size of the smallest time interval used by the application. For example, in the Software Development Corporation, we may have the following intervals that are of interest: $\mathbf{I} = \{i_1, i_2, i_3, i_4\}$, where i_1 = [8 a.m. to 5 p.m.], i_2 = [8 a.m. to 12 p.m.], i_3 = [12 p.m. to 1 p.m.], and i_4 = [1 p.m. to 5 p.m.]. The minimal time granularity pertaining to this application is 1 hr.

C. Spatio-Temporal Zone

One of the main contributions in our work is the formalization of the concept of a spatio-temporal zone. The concept of spatio-temporal zone abstracts location and time representation into a single entity. Now we give the formal definition of the notion spatio-temporal zone and the zones set.

Definition 2 (Spatio-Temporal Zone): A spatio-temporal zone STZone is a pair of the form (l, d), where l and d represent the logical location and the time interval, respectively.

An example of a spatio-temporal zone can be $z = (HomeOffice, [6\text{ p.m. to }8\text{ a.m.}])$.

A spatio-temporal zone set, i.e., $STZones = \{z_0, z_1, \ldots, z_n\}$, is a set of all spatio-temporal zones in an organization that defines where and when some entities are available. An example of a spatio-temporal zone set is {(HomeOffice, [6 p.m. to 8 a.m.]), (DeptOffice, [8 a.m. to 6 p.m.])}. A spatio-temporal zone (l, d) is specified at minimal granularity if l and d are specified at minimal location granularity and minimal temporal granularity, respectively.

Definition 3 (Spatio-Temporal Zone Containment): A spatio-temporal zone $z_1 = (l, d)$ is contained in another spatio-temporal zone $z_2 = (l', d')$, denoted by $z_1 \subseteq z_2$, if both zones have time intervals containment and locations containment, $d \subseteq d'$ and $m(l) \subseteq m(l')$, where m is the mapping function discussed in Definition 1.

Note that $\{(FortCollins, May2011)\} \subseteq \{(Colorado, Year2011)\}$ since $m(FortCollins) \subseteq m(Colorado)$ and $May2011 \subseteq Year2011$. However, $\{(FortCollins, May2011)\} \not\subseteq \{(Colorado, Year2010)\}$ since $May2011 \not\subseteq Year2010$. Similarly, $\{(FortCollins, May2011)\} \not\subseteq \{(Nevada, Year2011)\}$ because $m(FortCollins) \not\subseteq m(Nevada)$**.** We define a spatio-temporal zone {(anywhere, always)} that contains all other spatio-temporal zones. For spatio-temporal zones equality =, two zones z and z' are said to be equal, $z = z'$, iff $z \subseteq z'$ and $z' \subseteq z$.

IV. GSTRBAC Model

We now present our GSTRBAC model and formalize its specification using UML and OCL.

A. Effect of Spatio-Temporal Constraints on RBAC Entities

The RBAC entities users, roles, permissions, and objects are associated with spatio-temporal zones.

1) *Users:* We assume that each valid user, interested in doing some location-sensitive operations, carries a locating device that is able to track his location. The location of a user changes with time. The spatio-temporal zone associated with a user gives the user's current location and time.

Note that time and location can have different levels of granularity. For example, the current time can be expressed as 12:00:05 p.m. or 12:00 p.m. Similarly, a user's current location can be Fort Collins or it can be Colorado. The user's current location and time information will be used for making access decisions. Let us illustrate why the notion of minimality must be associated with the user's spatio-temporal zone. Suppose permission is valid in a certain zone. If we do not use the concept of minimal, then it is possible that the user zone may partially overlap with the permission zone. In such a case, should we give access or deny access? On the other hand, if we use the concept of minimal, then the user's zone will either be within the permission zone or outside it. In such cases, we know whether to give or deny access. Consequently, we require the minimal temporal and location be used to express the spatio-temporal zone associated with a user. We define the function $currentzone$ that returns the minimal spatio-temporal zone associated with a user. This function is formally defined as follows:

- $currentzone : Users \rightarrow STZones$.

2) *Objects:* Objects may also be mobile like the user. Here, again, we have locating devices that track the location of an object. Moreover, an object may not be accessible everywhere and anytime. For example, tellers can only review customer information at a teller office during working hours. The $ozones$ function returns the spatio-temporal zones that determine where and when every object is available

- $ozones : Objects \rightarrow 2^{STZones}$.

3) *Roles:* Role can be assigned or activated only in specific locations and time. The role of the on-campus student can only be assigned or activated inside the campus during the semester. The spatio-temporal zone associated with a role gives the location and time from which roles can be assigned or activated. The $rzones$ function gives the set of spatio-temporal zones associated with a given role

- $rzones : Roles \rightarrow 2^{STZones}$.

4) *Permissions:* Permissions are also associated with a spatio-temporal zone that indicate where and when a permission can be invoked. For example, a permission to perform a backup of servers can be executed only from the department after 10 p.m. on Friday nights. The function $pzones$ gives the zones in which a specific permission can be accessed

- $pzones : Permissions \rightarrow 2^{STZones}$.

Fig. 1 shows the class diagram of GSTRBAC. A security policy of a mobile application can be specified as one possible instance of this GSTRBAC class diagram. The GSTRBAC entities: *User*, *Role*, *Permission*, *Object*, *Activity*, and *STZone*, are represented by classes. *Permission* is represented in the GSTRBAC class diagram as an aggregation of the classes *Object* and *Activity*. The *STZone* class aggregates the location and time subclasses. In *STZone* class, *zcontainment* is a reflexive association specifying that a zone can contain other zones. Different relationships between entities, including *UserRoleAssignment*, *UserRoleActivation*, *PermissionRoleAssignment*, *RoleHierarchy*, and *SoD*, are modeled using association classes that are transformed to normal classes following the modeling guidelines in [5] and [6]. These association classes have binary relationships with *STZone* class to enforce the spatio-temporal constraints.

B. Effect of Spatio-Temporal Constraints on RBAC Operations

1) *User-Role Assignment:* A user-role assignment is location and time dependent. That is, a user can be assigned to a role, provided the user is in specific locations. For example, a person can be assigned the on-campus student role only when he is in the campus during the semester. This requirement is expressed using the zone concept

- $UserRoleAssignment \subseteq Users \times Roles \times STZones$.

This relationship is depicted in the GSTRBAC class diagram as association class *UserRoleAssignment*. The OCL operation *assignRole* assigns role r to user u in zone z if z is in the set of *rzones*, user u is present in zone z, and role r is not already assigned to user u in zone z. For the lack of space, we omit the descriptions of the OCL queries used in the *assignRole* operation

```
context User::assignRole(r: Role, z:STZone):
UserRoleAssignment
pre:  r.rzones->includes(z)
pre:  z.containedZones()->includes(self.
currentzone)
pre:  self.getAssignedRoles(z)->excludes(r)
post: self.getAssignedRoles(z)->includes(r).
```

2) *User-Role Activation:* A user can activate a role if the role can be activated on the specific zone and it is already assigned to that user. For example, the role of a doctor trainee can only be activated in a hospital during the training period. We define the *UserRoleActivation* relation to determine the current active roles based on zones

- $UserRoleActivation \subseteq Users \times Roles \times STZones$.

In the GSTRBAC class diagram, *UserRoleActivation* class is specified in a manner similar to *UserRoleAssignment*. The only difference is that the *activateRole* operation ensures that a user is already assigned to a role before the role is being activated.

3) *Check Access:* This operation checks whether a user is authorized to perform some operation on an object during a certain time and from a certain location. A user is allowed to fire a missile if he is assigned the role of a top secret commander and he is in the controller room of the missile during a severe crisis period. Thus, a user can access an object in a certain zone if that user has activated a role that has an appropriate permission for that object in that zone

```
context User::checkAccess(o:Object,a:
Activity,z:STZone):Boolean
post: result = getActivatedRoles(z)->
collect( r | r.getAuthorizedPermissions(z))
->asSet()->
exists( p | p.object=o and p.activity=a and
 o.ozones->includes(z)).
```

4) *Permission-Role Assignment:* Permissions can only be assigned to a role during specific time and locations. For example, the permission of opening a cashier drawer in a store should only be assigned to a salesman role during the daytime. The assignment of permissions to roles is specified based on zones

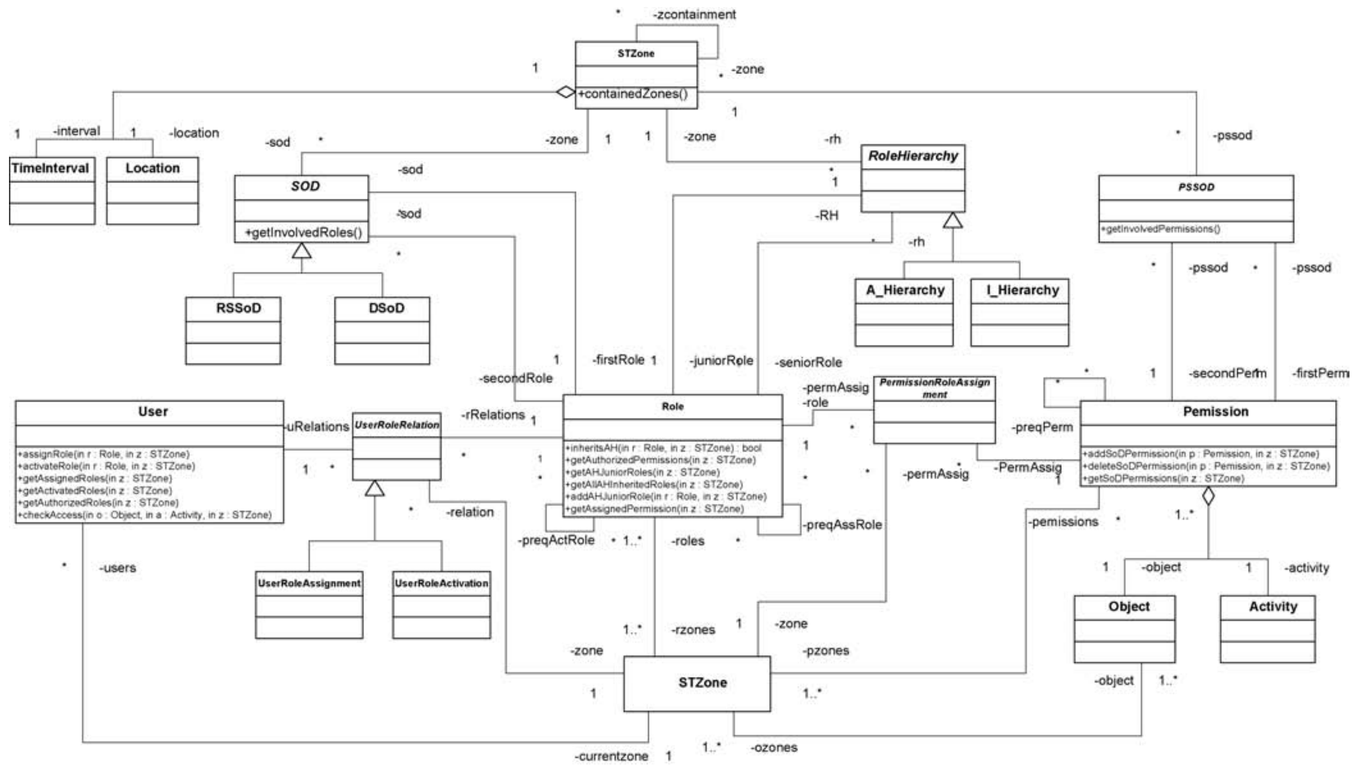

STZone
+containedZones()
-zcontainment
-zone
TimeInterval
Location
-interval
-location
SOD
+getInvolvedRoles()
-sod
RSSoD
DSoD
RoleHierarchy
-rh
-RH
A_Hierarchy
I_Hierarchy
PSSOD
+getInvolvedPermissions()
-pssod
User
+assignRole(in r : Role, in z : STZone)
+activateRole(in r : Role, in z : STZone)
+getAssignedRoles(in z : STZone)
+getActivatedRoles(in z : STZone)
+getAuthorizedRoles(in z : STZone)
+checkAccess(in o : Object, in a : Activity, in z : STZone)
-uRelations
UserRoleRelation
-rRelations
-relation
UserRoleAssignment
UserRoleActivation
-users
Role
+inheritsAH(in r : Role, in z : STZone) : bool
+getAuthorizedPermissions(in z : STZone)
+getAHJuniorRoles(in z : STZone)
+getAllAHInheritedRoles(in z : STZone)
+addAHJuniorRole(in r : Role, in z : STZone)
+getAssignedPermission(in z : STZone)
-secondRole
-firstRole
-juniorRole
-seniorRole
-preqActRole
-preqAssRole
-roles
PermissionRoleAssignment
-permAssig
-role
-PermAssig
Pemission
+addSoDPermission(in p : Pemission, in z : STZone)
+deleteSoDPermission(in p : Pemission, in z : STZone)
+getSoDPermissions(in z : STZone)
-preqPerm
-secondPerm
-firstPerm
-pemissions
Object
Activity
-object
-activity
-rzones
-pzones
-ozones
-currentzone

Fig. 1. UML class model for GSTRBAC.

- *PermissionRoleAssignment* $\subseteq$ *Permissions* $\times$ *Roles* $\times$ *STZones*.

The following OCL operation assigns permission p to role r in zone z if z is in the set of $pzones$ and $rzones$:

```
context Role::assignPermission(p:Permission,
z:STZone): PermissionAssignment
pre:   p.pzones->includes(z) and self.rzones->
includes(z)
pre:   self.getAssignedPermissions(z)->excludes(p)
post:  self.getAssignedPermissions(z)->includes(p).
```

C. Spatio-Temporal Role Hierarchy

The permission-inheritance hierarchy (*I-Hierarchy*) and the role-activation hierarchy (*A-Hierarchy*) are two variations of role hierarchy (*RoleHierarchy*) in RBAC [11], [13]. In our model, a senior role could have a subset of junior roles in a particular zone. The spatio-temporal role hierarchies are formally defined as follows.

1) *RoleHierarchy* $\subseteq$ *Roles* $\times$ *Roles* $\times$ *STZones*.
2) *I-Hierarchy* $\subseteq$ *RoleHierarchy*, *A-Hierarchy* $\subseteq$ *RoleHierarchy*, and *I-Hierarchy* $\cap$ *A-Hierarchy* = ϕ.

The subtypes of *RoleHierarchy* are represented in the GSTRBAC class diagram by the subclasses *I-Hierarchy* and *A-Hierarchy*, which are connected to *STZone* class to restrict the roles associations.

1) *Permission-Inheritance Hierarchy:* In a permission-inheritance hierarchy, a senior role r can only inherit junior role r' permissions in zone z if both roles are available in zone z. A project manager inherits the permissions of a developer when he is at the customer site giving a demo. The following OCL expression specifies the spatio-temporal constraint on I-Hierarchy for adding a new junior role:

```
context Role::addIHJuniorRole(r:Role,z:STZone):
I_Hierarchy
pre: self.rzones->includes(z) and r.rzones->
includes(z)
pre:  self.getIHJuniorRoles(z)->excludes(r)
post: self.getIHJuniorRoles(z)->includes(r).
```

The delete operation of a junior role in *I-Hierarchy* can be defined in the similar manner. The *I-Hierarchy* relationship is acyclic as shown by the following OCL constraint:

```
context r1,r2: Role
inv IHierarchy_Cycle_Constraint: not
STZone.allInstances-> exists(z|r1.inheritsIH(r2,z)
and r2.inheritsIH(r1,z)and r1<>r2).
```

The Boolean operation *inheritsIH(r,z)* returns true if role r is directly or indirectly a junior role of the context role in a particular zone; otherwise, it returns false.

```
inheritsIH(r:Role,z:STZone): Boolean =
if (self.getIHJuniorRoles(z)->includes(r))
then true
else self.getIHJuniorRoles(z)->
exists(j | j.inheritsIH(r,z)) endif.
```

We define the OCL query operation *getAuthorizedPermissions(z)* to get the authorized permissions for a given role at zone z through direct assignment or indirect *I-Hierarchy*

```
context Role::getAuthorizedPermissions(z:STZone):
Set(Permission)
Post: result= self.getAssignedPermissions(z)->
union(self.getAllIHInheritedRoles(z)->collect(r |
r.getAssignedPermissions(z)))->asSet().
```

2) *Role-Activation Hierarchy:* Restricted spatio-temporal *A-Hierarchy* allows members of senior roles to activate junior

roles in predefined spatio-temporal zones. For example, a department chair can activate a staff role during the semester inside the department building. The OCL operations of adding and deleting junior roles to the *A-Hierarchy* are defined in similar manner to *I-Hierarchy*. Furthermore, the acyclic constraints on *A-Hierarchy* are enforced in the same way as the *I-Hierarchy*.

The only differences are that the OCL query operation *getAHJuniorRoles(z)* returns all the junior role in *A-Hierarchy* of the context role in particular zone. Moreover, the OCL query operation *getAuthorizedRoles(z)* gives the authorized activation roles for the context user that are either explicitly assigned or implicitly obtained through *A-Hierarchy* in certain zones

```
context User:: getAuthorizedRoles(z:STZone):
Set(Role)
post: result= self.getAssignedRoles(z)->
union(self.getAssignedRoles(z)->collect(r|
r.getAllAHInheritedRoles(z))->
asSet()).
```

D. Spatio-Temporal Separation of Duty

The static SoD (SSoD) and dynamic SoD (DSoD) are two special classes of the SoD constraints in RBAC [28]. Furthermore, the role SSoD (RSSoD) constraints are defined on roles assignment, while the permission SSoD (PSSoD) constraints are defined on permissions assignments.

In our model, the conflicting roles and permissions in SoD are defined over some zones. The spatio-temporal RSSoD, PSSoD, and DSoD relations are formally defined as follows.

1) *RSSoD* $\subseteq$ *Roles* $\times$ *Roles* $\times$ *STZones*.
2) *DSoD* $\subseteq$ *Roles* $\times$ *Roles* $\times$ *STZones*, and *RSSoD* $\cap$ *DSoD* = ϕ.
3) *PSSoD* $\subseteq$ *Permissions* $\times$ *Permissions* $\times$ *STZones*.

The static and dynamic SoD relations are represented in the GSTRBAC class diagram using the associations classes RSSoD, PSSoD, and DSoD, which connect the conflicting entities with certain zones.

1) *Role SSoD:* The same individual should not be assigned to specific roles in a specific location for some duration. For example, the same user should not be assigned to billing clerk and account receivable clerk roles in the same time at a specific trade corporation. The following OCL invariant forbids the assignment of conflicting roles in a particular zone:

```
context User
inv RSSOD_Constaint: STZone.allInstances->forAll( z |
not self.getAssignedRoles(z)->
exists(r1,r2 | r1.getSSoDRoles(z)->includes(r2))).
```

However, the above constraint may be violated through a role hierarchy relation. For example, a billing supervisor may be a senior role of the two conflicting roles billing clerk and account clerk at the same time and in the same accounting department. The following OCL constraint prevents such a situation:

```
context User
 inv RSSOD_RH_Constraint: STZone.allInstances->
 forAll( z | not self.getAuthorizedRoles(z)->
 exists(r1,r2 | r1.getSSoDRoles(z)->includes(r2))).
```

2) *Permissions SSoD:* PSSoD prevents the assignment of conflicting permissions to a role. For example, a loan officer is not permissible to issue loan request and approve it in the bank building during the daytime. The following OCL invariant expresses the PSSoD requirement in our model:

```
context Role
inv PSSOD_Constaint: STZone.allInstances->
forAll( z | not self.getAssignedPermissions(z)->
exists(p1,p2 | p1.getPSSoDPermissions(z)->
includes(p2))).
```

However, this constraint may be violated through *I-Hierarchy*, in which a senior role inherits some junior roles that have mutually been assigned conflicting permissions. The following OCL invariant prevents the violation of PSSoD via *I-Hierarchy*:

```
context Role
inv PSSOD_RH_Constraint: STZone.allInstances->
forAll( z | not self.getAuthorizedPermissions(z)
exists(p1,p2 | p1.getPSSoDPermissions(z)->
includes(p2))).
```

3) *DSoD:* Two conflicting activation roles cannot be activated in some spatio-temporal zones by the same user. For example, the simultaneous activation of cashier and cashier supervisor is forbidden during the working hours in the same store to deter such a user from committing a fraud. The DSoD constraints are expressed in OCL invariants in a similar manner to the RSSoD constraints. The only difference is that the OCL invariants prevent the activations of conflicting roles that are connected by DSoD in some zones through either the explicit role assignment or the implicit *A-Hierarchy*.

E. Spatio-Temporal Prerequisite Constraints

In RBAC, the prerequisite constraints obligate that some actions to be taken prior to performing an operation [29]. Prerequisite constraints impose conditions that must be satisfied before certain assignments, such as user-role assignment and permission-role assignment can be executed. Our spatio-temporal prerequisite constraints can be expressed in first-order logic; consequently, we can represent them using OCL preconditions and invariants.

1) *Prerequisite Constraints on User-Role Assignment:* The prerequisite constraint on roles assignments imposes that a user must be assigned to some less critical roles in a given spatio-temporal zone before being assigned more critical roles in specific zones. For example, the role of emergency nurse can be assigned to John in the urgent care unit from 12:00 a.m. to 5:00 a.m. if he is assigned the role of nurse-on-night-duty at the hospital during those hours. The following OCL invariant expresses the prerequisite constraints on user-role assignment. The query operation *getPreqAssRoles()* returns all the assignment prerequisite roles needed for assigning a certain role

```
context User
inv Prerequiste_URAssign: STZone.allInstances->
forAll(z | Role.allInstances->
forAll(r1 | (self.getAssignedRoles(z)->
includes(r1)) implies (self.getAssignedRoles(z)->
includesAll(r1.getPreqAssRoles())))).
```

2) *Prerequisite Constraints on Permission-Role Assignment:* The prerequisite constraints on permissions assignments indicate that a role can be assigned a permission in a specific zone if some prerequisite permissions are already assigned to that role in the same zone. For example, a bank teller must have the permission of reading an account during working hours before he can be given the permission to update that account. The prerequisite constraint on permission-role assignment can be specified using OCL expression as follows:

```
context Role
inv Prerequist_PRAssign: STZone.allInstances->
forAll(z | Permission.allInstances->
forAll(p1 | (self.getAssignedPermissions(z)->
includes(p1)) implies
(self.getAssignedPermissions(z) ->
includesAll(p1.getPrerequisitePermissions()))))).
```

3) *Prerequisite User-Role Activation:* A role can be activated if some prerequisite roles are already activated in specific zones. For example, in a university the teaching assistant role can be activated during a semester in a department if the student role can be activated during the same time. This requirement is specified in our model in the same way of the prerequisite user-role assignment constraint except that the OCL query *getPreqAssRoles()* is substituted with *getPreqActRole()*. The query operation *getPreqActRole()* returns all activation prerequisite roles needed to activate a role.

V. Example Access Control Verification

The GSTRBAC model has many features that may interact with each other, causing conflicts, inconsistencies, and security breaches. In addition, constraints could be specified stronger than needed resulting in some roles, permissions, or objects being inaccessible. Consequently, it is important to analyze GSTRBAC policies before applying them. A manual analysis is tedious and error prone. Toward this end, we propose an automated verification approach based on the USE constraint solver tool. The USE tool accepts three inputs, the GSTRBAC class diagram, the OCL constraints, and the policy instances as object diagrams. When the policy instance does not conform to a GSTRBAC class diagram or it violates some property, the tool pictorially shows how the property has been violated. Specifically, it illustrates the entities and the relationships responsible and how their interactions have caused for the property violation. Once the security designer sees this graphical representation, he can fix the policy specification. For example, if the security designer finds that some prerequisite constraint has been violated, he can either change the prerequisite relation or change the constraint depending on the application requirement.

To illustrate our specification and verification approaches, consider a software development environment for producing military applications. The software project files are stored in computer machines inside a secure building, and the access to those files is location and time dependent. The access control policy of the software development system is specified as follows.

1) Users = {Bob, Ben, Alice, Rachael, Clare, Sam}.
2) Intervals = $\{i_1, i_2\}$. where i_1 = [8 a.m. to 6 p.m.] and i_2 = [6 p.m. to 8 a.m.].
3) Locations = {Home, DevelopmentOffice, TestingOffice, DirectorOffice, DepartmentBuilding}. DepartmentBuilding includes all other offices.
4) STZones = $\{z_0, z_1, z_2, z_3, z_4\}$, where z_0 = (DepartmentBuilding, i_1), z_1 = (Home, i_2), z_2 = (DevelopmentOffice, i_1), z_3 = (TestingOffice, i_1), and z_4 = (DirectorOffice, i_1).
5) Roles = { Software Engineer (SE), Software Programmer (SP), Test Engineers (TE), Programmer Supervisor (PS), Test Supervisor (TS), Project Lead (PL) }.
6) rzones = { (SE, z_0), (SE, z_2), (SP, z_1), (SP, z_2), (TE, z_1), (TE, z_3), (PS, z_2), (TS, z_3), (PL, z_4) }.
7) Objects = { Project Files (obj_1), Test Files (obj_2), Programmer Logs (obj_3), Test Logs (obj_4), Programmer Supervisor Report (obj_5), Test Supervisors Reports (obj_6) }.
8) ozones = { (obj_1, z_1), (obj_1, z_2), (obj_2, z_1), (obj_2, z_3), (obj_3, z_2), (obj_4, z_3), (obj_5, z_4), (obj_6, z_4)}.
9) Activities = $\{read, write, copy, run, review\}$.
10) Permissions = { P_1(read, obj_1), P_2(write, obj_1), P_3(copy, obj_1), P_4(write, obj_2), P_5(run, obj_2)), P_6(review, obj_4), P_7(review, obj_3)), P_8(read, obj_5) }.
11) pzones = { (P_1, z_1), (P_1, z_2), (P_2, z_1), (P_2, z_2), (P_3, z_2), (P_4, z_1), (P_4, z_3), (P_5, z_3), (P_6, z_3), (P_7, z_2), (P_8, z_4) }.
12) UserRoleAssignment = { (Ben, SP, z_1), (Ben, SP, z_2), (Bob, PS, z_2), (Alice, PL, z_4), (Clare, TS, z_3), (Rachael, TE, z_1), (Rachael, TE, z_3), (Sam, SE, z_0) }.
13) PermissionRoleAssignment = { (SP, P_1, z_1), (SP, P_1, z_2), (SP, P_2, z_1), (SP, P_2, z_2), (SP, P_3, z_2), (TE, P_4, z_1), (TE, P_4, z_3), (TE, P_5, z_3), (TS, P_6, z_3), (PS, P_7, z_2), (PL, P_8, z_4) }.
14) I-Hierarchy = { (PS, SP, z_2), (TS, TE, z_3), (PL, PS, z_0), (PL, TS, z_0) }.
15) RSSoD = { (SP, TE, z_0) }.
16) PSSoD = { (P_2, P_4, z_0) }.
17) Prerequisite constraints: The role of software programmer and the test engineer can be assigned if the user is already assigned the role of software engineer.

The policy is graphically represented in Fig. 2 based on the graph notations inspired by Chen and Crampton [17]. We present two scenarios illustrating our analysis approach for verifying security properties.

The first scenario shows how the *checkAccess* operation can be analyzed. Assume that *Ben* is in zone z_3 = $\{(TestingOffice, [8\text{ a.m. to }6\text{ p.m.}])\}$ and tries to *copy ProjectFiles* object. Based on the policy, *Ben* is only assigned to SP in z_1 and z_2. Thus, *Ben* should not be allowed to access that object because the SP role is not available for activation in zone z_3. Fig. 3 shows that no policy violation is found with this scenario.

The second scenario considers the verification of the RSSoD constraints. Assume that *Ben* is already assigned to the role SP in the zone z_0, which is a containing zone for z_1 and z_2.

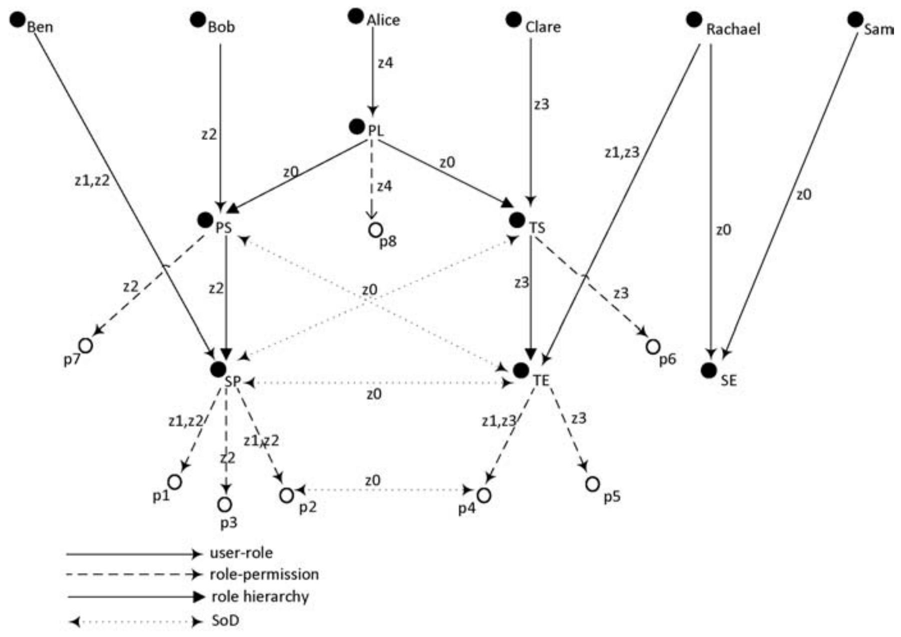
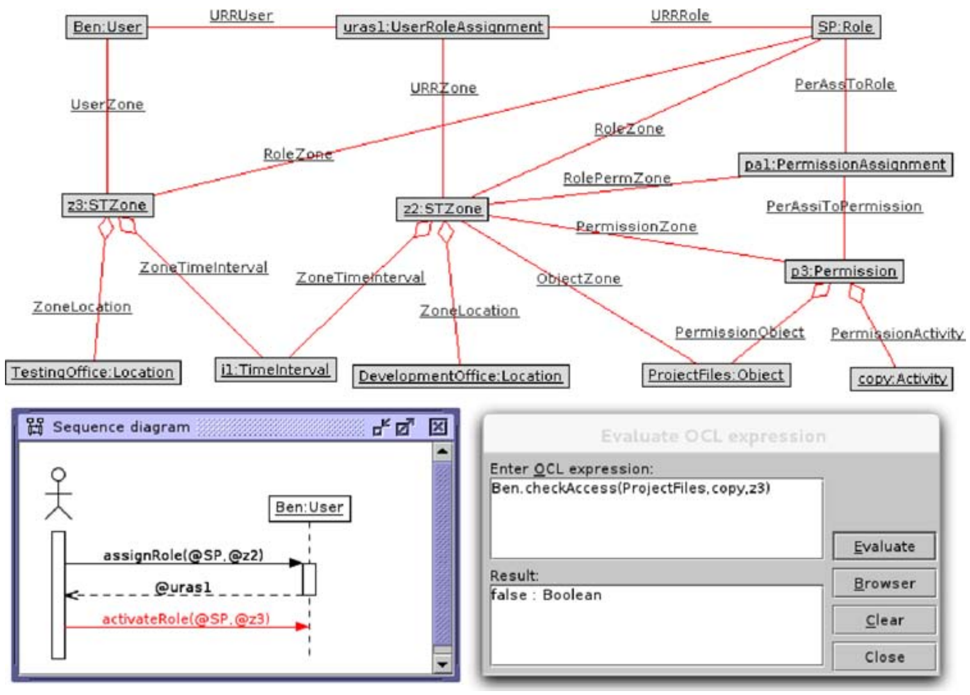


Fig. 2. Access control graph for the development system policy.



Enter OCL expression:
Ben.checkAccess(ProjectFiles,copy,z3)

Result:
false : Boolean

Fig. 3. Accessing object from an invalid STZone.

Later on, a security administrator tries to assign Ben to the conflicting role TE in the zone z_0. The assignment operation fails due to the RSSoD constraint, as shown in Fig. 4.

VI. Software Architecture

This section describes a platform-independent implementation architecture, which maps the high-level *GSTRBAC* policy definition to the enforcement mechanism in mobile applications. Later on, we provide an experimental evaluation of this architecture using Android mobile operating system (OS) [30].

Fig. 5 depicts the proposed implementation architecture for enforcing *GSTRBAC* in a mobile application. The architecture consists of three core components: request composition module (RCM), resource access module (RAM), and authorization control module (ACM). Each of these modules is a stand-alone program. The RCM is installed at the user mobile device, the RAM and ACM are installed in servers that may or may not be colocated.

1) RCM is responsible for forming a user access request and maintaining the access, while the rights are exercised. The *Request Builder* component in RCM creates a

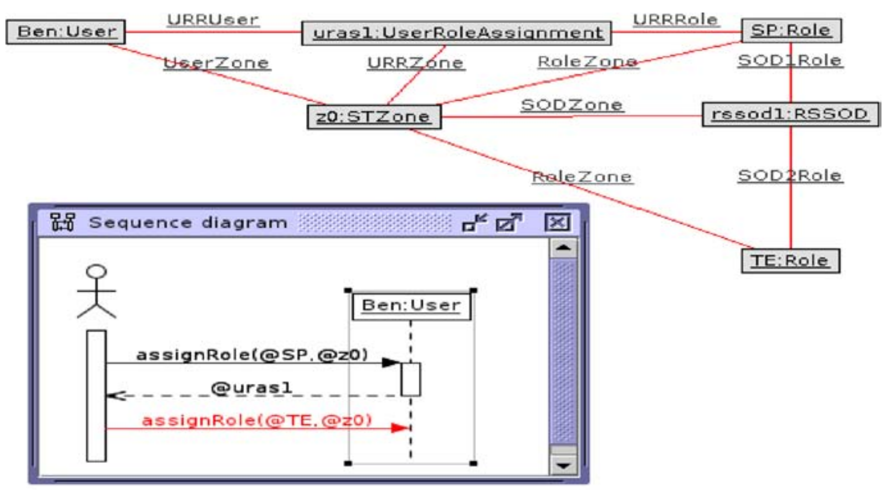


Fig. 4. Conflicting roles assignment.

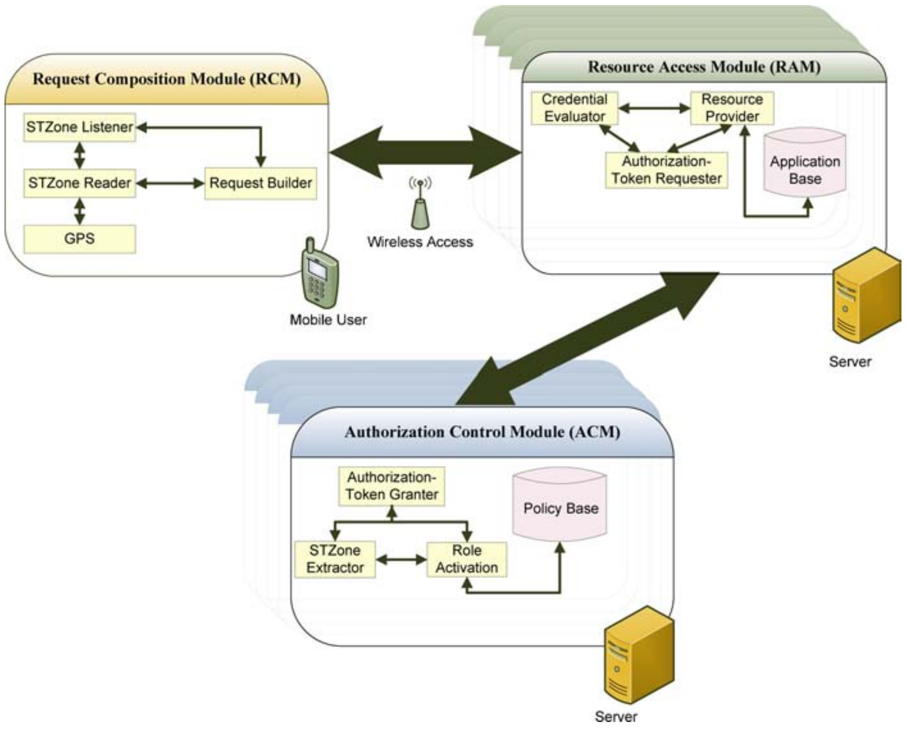


Fig. 5. Implementation architecture of GSTRBAC policy in mobile applications.

resource access request using information obtained from the other components. The *Request Builder* enquires the *STZone Reader* to form the current user *STZone* that encapsulates current user location and time. The *STZone Reader,* in turn, reads the current mobile device time and gets the location from GPS data component storing the location information. After creating the proper access request package, *Request Builder* sends that package to one of the available *RAM* servers. The *STZone Listener* gets the current spatio-temporal information from *STZone Reader* and ensures that the user is in the authorized spatio-temporal zone while the resource is being accessed. Once the user moves outside the authorized zone, *STZone Listener* requests service termination.

2) *RAM* is an intermediate server between the user and ACM server, which is primarily responsible for handling the application resources to the users. *RAM* receives the user request and consults with the ACM server about the user authorization. The user's credentials are stored in the *Application Base* when the user registers with the system. The *Credential Evaluator* evaluates the validity of the user credentials. If the user credentials are valid, the *Authorization-Token Requester* component requests an authorization token (AT) from one of the available ACM servers. Once the AT is granted, it is used by the *Resource Provider* component to provide access to resources needed by the authorized users to accomplish their tasks. *RAM* maintains a list of users' ATs, which

we refer to as *UATokens*, and the elements of this list are continuously updated. A user AT gets deleted from the list once it is expired, which might happen whenever a user deactivates his role or the STZone associated with the role becomes invalid.

3) ACM is accountable for the policy evaluations and tokens generations. Typically, ACM is responsible for issuing a new AT for every role that the user requests to be activated. In order for ACM to evaluate access requests, it consults with the *GSTRBAC Policy Base*. *GSTRBAC Policy Base* stores the access control policy and the authorizations granted to users. The *Role Activation* component in ACM gets the set of roles and permissions that can be activated based on the *STZone* it receives from the *STZone Extractor* component. The *Role Activation* component updates the policy state for each activated role. The *Authorization-Token Granter* component is responsible for granting the AT if a role can be activated. Note that a user's role can be activated only if all the following conditions are satisfied: 1) the role is not already active; 2) the role can be activated in the given *STZone*; and 3) no conflicting roles are already active. If the requested role can be activated, *Authorization-Token Granter* issues a new authorization token with the following format: $AT = (ID_u, ID_{ut}, r, P, STZone)$, where ID_u refers to the user identifier, ID_{ut} is the token's identifier, r is the requested role to be activated, P is the set of the authorized permissions associated with the active role r, and $STZone$ defines where and when these privileges are valid.

VII. Resource Usage Protocols

A. Assumptions

Each mobile device is associated with a user through which the user can access resources. We assume that the users' mobile phones (clients) and servers have tamper-proof storages where the ATs and keys are stored. We also have tamper-proof components that can only be accessed by authorized applications. These components house critical software, such as RCM that cannot be accessed or modified by the unauthorized user. In addition, the client has the needed software to extract the current time and location information. The clients and servers should have the capabilities of executing public key cryptography algorithms and hashing algorithms, such as MD5. We also assume that the existence of a certificate authority is responsible for providing public keys and private keys for the clients and servers. We use timestamps in the protocol, so the different servers and clients must be synchronized. The user identifier and his device identifier must be registered with the RAM servers, and the user must have a unique password.

Fig. 6 describes the communication exchanges of the resource usage protocols. Suffixes associated with the communication messages indicate the order of steps in the protocols. Table I enumerates the notations used in the description of the protocol. Message $\mathbf{M}_i$ indicates Step i of the protocol.

Now we describe the steps of the basic resource usage protocol for handling users' requests.

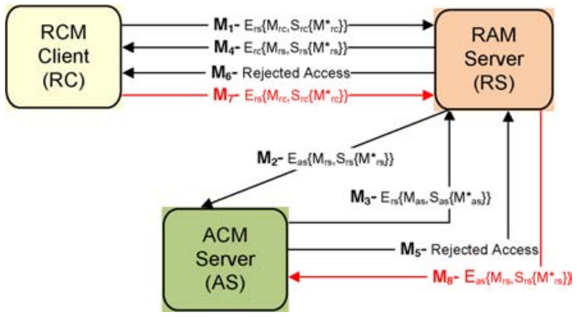


Fig. 6. Communication steps of the resource usage protocols.

TABLE I
The Notations of the Resource Usage Protocols

Symbol	Interpenetration
ID_x	Identifier of party x
ID_s	User's device identifier
PuK_x	Public key of party x
PrK_x	Private key of party x
Ts_x	Timestamp computed by party x
P_u	User password
P_u^*	One-time password
$H(P_u, Ts_x)$	Computes P_u^*
M_x	Package payload created by party x
$E_x\{S\}$	Encryption of sequence S by PuK_x
M_x^*	Message checksum generated by party x
$H(M_x, Ts_x)$	Computes M_x^*
$S_x\{M_x^*\}$	Signing M_x^* by PrK_x
$A \xrightarrow{M_i} B$	Party A sends package M_i to party B
Tw	Time window
AT	Authorization token
ID_{ut}	Authentication-token identifier
"Close"	Keyword indicates deletes user's AT
"Freeze"	Keyword indicates suspends user's AT

1) *Role Activation Request* $[RCM \xrightarrow{M_1} RAM]$: RCM creates an access request payload $M_{rc} = (ID_u, ID_s, P_u^*, STZone, r, Ts_{rc})$, where ID_u is the user identifier, ID_s is the device identifier, $P_u^* = H(P_u, Ts_{rc})$ is the user one-time password, $STZone$ is the current user zone, r is the requested role, and Ts_{rc} is the timestamp at which M_{rc} is created. RCM computes the hash value $M_{rc}^* = H(M_{rc}, Ts_{rc})$ and signs it using the user's private key PrK_u, i.e., $S_{rc}\{M_{rc}^*\}$, to be used as a nonrepudiation proof.
2) *User AT Request* $[RAM \xrightarrow{M_2} ACM]$: On receiving message $\mathbf{M}_1$, RAM decrypts it using its private key. If the message is validated and the user is authenticated, RAM sends message $\mathbf{M}_2$ to the ACM server in order to issue an AT. RAM forwards the AT request payload $M_{rs} = (ID_{rs}, ID_u, STZone, r, Ts_{rs})$, where Ts_{rs} is the timestamp at which M_{rs} payload request is created. RAM computes the hash value of M_{rs} and signs it using its private key PrK_{rs}. Then, RAM encrypts the M_{rs} along with a digitally signed signature $S_{rs}\{M_{rs}^*\}$ using ACM public key PuK_{as}.
If the user authentication fails, RAM sends a rejection message to RCM that is similar to message $\mathbf{M}_6$.

3) *User AT Response* [$ACM \xrightarrow{M_3} RAM$]: At the authorization server, ACM decrypts the package M_2. ACM recomputes the hash value and compares it with the one in package M_2, and verifies the digital signature using the public key of RAM PuK_{rs}.
   In the case the sender of $\mathbf{M}_2$ is authenticated and the user has the rights to activate the requested role, an AT is sent back as a message $\mathbf{M}_3$. $\mathbf{M}_3$ includes the payload M_{as} and a signed hash of M_{as}. The payload M_{as} = ($ID_{as}, ID_{ut}, ID_u, AT, Ts_{as}$) has the user identifier ID_u, user's authorization-token AT, token identifier ID_{ut}, and timesatmp Ts_{as} at which the payload is created.
   If the user's request cannot be granted, an access rejection response is sent to the RAM, as shown in message $\mathbf{M}_5$.
4) *Forwarding User AT* [$RAM \xrightarrow{M_4} RCM$]: After authenticating message $\mathbf{M}_3$ from ACM, RAM stores a copy of the user's AT in the *UATokens* list along with the token identifier ID_{ut}, user identifier ID_u, and device identifier ID_s. Subsequently, RAM forwards an encrypted and signed response message $\mathbf{M}_4$ to RCM. This message includes payload M_{rs} = ($ID_{rs}, ID_u, ID_s, ID_{ut}, AT, Ts_{rs}$) and $S_{rs}\{M^*_{rs}\}$ digital signature signed by the private key PrK_{rs}. $\mathbf{M}_4$ is encrypted using the user public key PuK_u. Note that each user AT is related to a particular user ID_u and a device ID_s.
   However, in the case the access request is rejected by ACM, RAM directly forwards the rejection response in message $\mathbf{M}_6$ to RCM.
   At the user side, if $\mathbf{M}_4$ message from RAM is authenticated, RCM stores the AT in its secure storage.

In our model, we need to revoke access whenever the user moves out of a valid *STZone*. The *STZone Listener* gets periodic updates by the *STZone Reader* about the spatio-temporal coordinates of the user. Whenever the current user location or time does not satisfy the information in a user's AT, the *STZone Listener* revokes the user's AT and requires *Request Builder* to request an access termination.

In Fig. 6, the messages $\mathbf{M}_7$ and $\mathbf{M}_8$ describe the additional exchanges needed to implement the ongoing access protocol.

1) *Terminating User Access* [$RCM \xrightarrow{M_7} RAM$]: The client software sends the termination access request $\mathbf{M}_7$ to RAM at the time the current user *STZone* becomes invalid. RCM creates a termination message M_{rc} = ($ID_u, ID_s, ID_{ut}, P_u, Close, Ts_{rc}$), where the keyword "Close" indicates the termination of access. RCM concatenates M_{rc} with the user digital signature $S_{rc}\{M^*_{rc}\}$ and encrypts them with the resource server public key PuK_{rs}. The user's AT should be deleted at the client side to terminate the user access.
2) *Revoking User Privileges* [$RAM \xrightarrow{M_8} ACM$]: After authenticating the sender of $\mathbf{M}_7$ message, RAM reads the keyword "Close" and then uses ID_u, ID_s, and ID_{ut} to lookup for the user AT in the *UATokens* list and removes it. Therefore, if a user subsequently requests an access to a resource via the same ID_{ut}, his request will be denied because that user does not have the AT for that resource.
   RAM forwards an encrypted and signed termination request $\mathbf{M}_8$ to ACM to delete the user's authorization. This request includes M_{rs} = ($ID_{rs}, ID_u, ID_{ut}, Close, Ts_{rs}$) and the digital signature $S_{rs}\{M^*_{rs}\}$ signed by the RAM private key.
   At the ACM server, after authenticating the sender of $\mathbf{M}_8$, the keyword "Close" indicates ACM to revoke the current user's active role and authorized permissions associated with the user AT ID_{ut}. It does this by updating the GSTRBAC policy state.

A user may temporarily depart the authorized location from which he is currently accessing some resources. To preserve our design efficiency, the user's privileges should not be permanently revoked, but these privileges must be frozen or suspended for the short period of time the user is off-site and they are returned when he is on-site.

We achieve this by modifying the communication steps of the ongoing resource usage protocol to define the suspending resource usage protocol. The client software sends a resource access deferral package $\mathbf{M}'_7$ at the time a user temporarily leaves his valid position.

This message has a similar format to $\mathbf{M}_7$ except that the keyword "Close" is changed to "Freeze" to indicate that user's privileges should be frozen for a certain period. The time window Tw is included in $\mathbf{M}'_7$ message. The time window determines the freezing time during which a user cannot exercise previously granted access rights. Once the user moves back to his previous valid location and prior to the Tw expiration, his roles can automatically be reactivated. If the user does not move to the valid position before the expiration of Tw, his privileges are revoked and message $\mathbf{M}_8$ is sent.

B. Security Analysis

Since our primary concern is protecting applications' resources from improper access, we need to provide a proof that our protocol does indeed protect the resources from unauthorized access. We assume that RAM and ACM are trusted. Furthermore, the RCM is installed in a tamper-proof component of the phone, and it can be only accessed by authorized applications.

In communication protocols, the message confidentiality, message integrity, message authentication, and identity authentication are important. We provide these features in our protocol. Message confidentiality is guaranteed because messages are encrypted by the public key of the recipient. Message integrity is maintained by concatenating each message with the message digest, so that if an intruder intercepts and alters a communication message, the receiver can detect that. The receiver validates the received message by reconstructing the checksum and comparing it to the one in the received message. Message authentication is protected by associating digital signatures with messages; the digital signature provides a non-repudiation proof of the message origin. Identity authentication is provided by using public key certificates and digital signatures. Using the public key certificates and digital signatures, the sender and the receiver are able to mutually authenticate their identities. We assume that private keys used to sign the message digests are only possessed by the signers, and the public key certificates have one-to-one mapping between the public key and the owner. Furthermore, the password-based authentication technique is utilized to provide a proof of identity. Note that the manner in which such techniques are used ultimately determines the security of the protocol.

In the following, we show how our resources are protected from improper access by unauthorized and authorized users. First, we consider the attacks that unauthorized users can carry out to gain an access to protected resources and show how these attacks are prevented in our approach.

1) *Eavesdropping:* An adversary may eavesdrop on the messages that one entity sends to another to breach confidentiality. In this case, even if an adversary eavesdrops on the communication across two entities, he cannot gain useful information as the messages are encrypted by the public key of the recipient.
2) *Modifications:* An adversary may intercept and modify data that one entity sends to another. In order to make this possible, the adversary must be able to read and modify the messages. Since the messages must be decrypted by the adversary before they can be modified, such an attack is not possible in our scheme.
3) *Replay:* In this attack, an attacker intercepts authentic messages coming from a legitimate entity and replays them to gain an access to resources. Even when messages are replayed by an adversary, the timestamps and one-time passwords prevent the replay attacks.
4) *Man-in-the-Middle (MITM)*: In an MITM attack, a malicious user eavesdropping on the communication channel between entities and masquerades as the legitimate entity to the other entity. It intercepts messages coming from a legitimate entity and retransmits them after possibly modifying them to the intended recipient. The use of a public key prevents the attacker from reading and modifying transmitted messages. Consequently, this attack does not occur in this protocol.
5) *Illegitimate Use*: An illegitimate use attack occurs when a malicious user tries to make an access via lost or stolen cell-phones. Since a malicious user will not be aware of the user's password, it cannot send new access requests.

We next consider the attacks that may be carried out by authorized users.

1) *Reusing Authorization-Tokens*: A user may want to access a resource using his past AT. However, when the *STZone* expires, the AT gets deleted from RCM and cannot be used.
2) *Modifying Authorization-Tokens*: A user may modify the AT stored at his site. However, ATs are stored in a secure tamper-proof storage at the user's mobile device and cannot be modified.
3) *Users Collusion*: In this attack, two or more users collude to commit a fraud. That is, user u_1 gets a valid AT for a role and sends it by some means to user u_2 to allow u_2 to violate conflicting constraints. However, this attack is not applicable to our protocols because each AT is exactly linked to a particular device and user.

VIII. Prototype Development

We have developed a proof-of-concept prototype enforcing GSTRBAC in an Android mobile application. Our Android application and servers are written using the Java programming language.

The RCM client is implemented in a mobile application that is developed using free and open source Google's Android (OS) [30]. An Android platform provides a flexible map display and a location service support. Our mobile application utilizes the Android location listener to keep track of the current location coordinates and captures the local time at which the location is fetched. We have used the Android *LocationManager* package [31] for developing an Android location-based services application. Our application retrieves the current position from an enabled GPS receiver, and uses the Google maps application programming interface to display the location on the screen.

The RAM and ACM components are written as traditional Java server programs. To implement the basic resource usage protocol, we have adopted some source code of the *FlexiProvider* Toolkit [32] that has Java-based cryptographic modules, including the public key, the digital signature, and the MD5 message digest. The toolkit implements fast and secure cryptographic algorithms and can be plugged into many Java applications. Prior to running the experiment, each entity should have its private key and stores the public key of the communicated endpoints in a local file. We use the *KeyPairGenerator* class to generate public and private key pairs. The public keys are securely distributed using the AES symmetric-key encryption. Furthermore, all communicating parties implement the same MD5 algorithm.

We used the relational database to represent the application and policy data. The relational database is implemented in the open source database MySQL server [33]. We have two relational database tables: the first one is the application table that stores login information, and the second one is the policy table that stores a GSTRBAC policy. The application table is only accessible by RAM via a MySQL server, while the policy table is connected to ACM.

We tested our mobile application through the Android emulator, and the test showed that our prototype works as expected. The Android handset emulator is depicted in Fig. 7 displaying our Android applications. The handset emulator prompts a user to select his role and enters the user's identifier and password. Once the user enters these information and hits the connect button, the Android application software fetches the last known user location and the local time, then it composes an access request and sends it to one of the available virtual RAM servers in a secure manner.

In order to use the location capabilities of Android, we used the *LocationManager* class to access to the Android location services. The *locationManager.requestLocationUpdates()* method updates the device's location every some fixed period of time through the location provider GPS. A class implement-

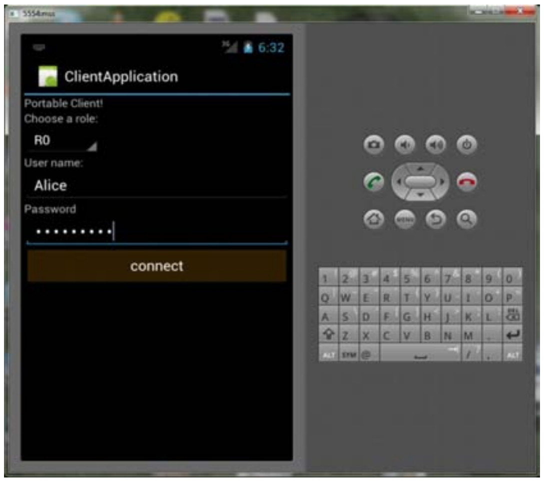


Fig. 7. Android handset emulator.

TABLE II
BACK-END AVERAGE RESPONSE DELAY

Response	Average Delay
0-Approved	73.66 ms
1-Rejected (improper login data)	29.56 ms
2-Rejected (improper role)	67.50 ms
3-Rejected (improper zone)	81.43 ms
Total Average Delay	63.04 ms

ing the *LocationListener* interface handles changes in the device location. The *locationManager.getLastKnownLocation()* method fetches the last known location object that has the altitude, latitude, and longitude information. When the Android emulator starts for the first time, it reports the current location *Null* because there is no last known location to fetch. Thus, we used *DDMS* view of Eclipse to manually feed mock location information to the Android emulator. Once the emulator's GPS device has the dummy data, the listener is triggered to retrieve the current location. Now, the Android SDK's emulator can emulate user's location changes. Then, the current user's coordinates is viewed on a Google map and the location name is manually fed into our application.

In our experiment, we instantiated three *RCM* handset Android emulators, three *RAM* virtual servers, and three *ACM* virtual servers. All of these execute on a single machine and communicate through sockets. We also created another virtual server running a local centralized MySQL database on the same machine. This database server is connected with the servers to process users' requests. The Android handset emulator sends an access request to one of the virtual *RAM* servers, and the *RAM* servers connect with *ACM* servers to get the access authorizations. For each request, the handset emulator opens a new connection with one of the virtual *RAM* servers' names stored in a local file, and it closes the connection at the time it receives a response. The *RAM* server opens a new connection with one of the *ACM* servers' names stored in a local file only if the user login information is correct, and it closes the connection at the time it gets a response form the endpoint *ACM* server.

To evaluate different spatio-temporal access scenarios, we have stored the logical locations and role names in two distinct local files. Thus, for each request, the handset emulator randomly selects a location name and a role name from these files and sends them along with other information in a request package. This approach allows us to test whether our application works as anticipated and validates the policy correctness. We measured the time from issuing an access request to the time when the response is received. The response delay is evaluated for 150 requests sent simultaneously from three Android handset emulators, each emulator sent 50 requests. The responses vary based on the information in the request packages. For example, a request gets approved only if the login information is correct, the requested role can be authorized, and the current user's zone is acceptable; otherwise, the request is rejected.

The experiments are carried out on Windows 7 platform running on Intel(R) Core(TM) 2 Due CPU at 2.20 GH with 4.00 GB *RAM*. The results in Table II exhibit the response average delay for each class of responses and the total average response delay in milliseconds. The overall average response delay yielded by the basic resource usage protocol is 63.04 ms. Furthermore, the rejected requests due to invalid login information yield 29.56 ms that is the smallest response delay because *RAM* servers send the responses immediately without consulting the *ACM* servers.

IX. CONCLUSION AND FUTURE WORK

In this paper, we proposed GSTRBAC that allows specification of role-based access control policies that are based on time and location. We introduced the concept of a spatio-temporal zone that encapsulated temporal and spatial constraints that facilitated understanding, analysis, and policy evolution. The model had many features that interacted with each other in subtle ways. Toward this end, we showed how an application using our model can be analyzed using the constraint solver embedded in USE. We proposed an architecture and developed a prototype for a mobile system enforcing GSTRBAC model. We developed a number of protocols that consider spatio-temporal information for initiating and maintaining access under different circumstances.

Much work remains to be done. In the current work, we presented arguments demonstrating that certain types of attacks that led to access control breaches did not occur in our protocol. Often times, formal analysis may reveal problems that are not apparent in the informal analysis. Consequently, to provide more assurance, we plan to formally analyze the security protocols using existing tools, such as Coloured Petri Nets [9] and Alloy [8]. We also plan to extend our spatio-temporal access control model for workflows that consist of a set of tasks that are coordinated by control-flow, data-flow, and temporal dependencies. It would be interesting to see how these various dependencies interact with the spatio-temporal constraints of the workflow. Our future work also includes deploying this model for a real-world dengue decision support application.

REFERENCES

[1] A. Maji, A. Mukhoty, A. Majumdar, J. Mukhopadhyay, S. Sural, S. Paul, and B. Majumdar, "Security analysis and implementation of web-based telemedicine services with a four-tier architecture," in *Proc. 2nd Int. Conf. Pervasive Comput. Technol. Healthcare*, 2008, pp. 46–54.

[2] I. Ray and M. Toahchoodee, "A spatio-temporal role-based access control model," in *Proc. 21st Annu. IFIP WG 11.3 Working Conf. Data Appl. Security (DBSec)*, 2007, pp. 211–226.

[3] S. Aich, S. Sural, and A. Majumdar, "STARBAC: Spatio temporal role based access control," in *Proc. On The Move to Meaningful Internet Syst. (OTM)*, 2007, pp. 1567–1582.

[4] A. Samuel, A. Ghafoor, and E. Bertino, "A framework for specification and verification of generalized spatio-temporal role based access control model," Purdue Univ., West Lafayette, IN, USA, Tech. Rep. CERIAS TR 2007-08, Feb. 2007.

[5] G. Booch, J. Rumbaugh, and I. Jacobson, *The Unified Modeling Language User Guide*, 2nd ed. Reading, MA, USA: Addison-Wesley Professional, 2005.

[6] L. Craig, *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development*, 3rd ed. Englewood Cliffa, NJ, USA: Prentice-Hall, 2004.

[7] M. Gogolla, F. Büttner, and M. Richters, "USE: A UML-based specification environment for validating UML and OCL," *Sci. Comput. Programming*, vol. 69, nos. 1–3, pp. 27–34, 2007.

[8] D. Jackson, "Alloy: A lightweight object modelling notation," *ACM Trans. Software Eng. Methodol.*, vol. 11, no. 2, pp. 256–290, 2002.

[9] K. Jensen, L. Kristensen, and L. Wells, "Coloured petri nets and CPN tools for modelling and validation of concurrent systems," *Int. J. Software Tools Technol. Transfer*, vol. 9, nos. 3–4, pp. 213–254, 2007.

[10] R. Alur and D. Dill, "A theory of timed automata," *Theor. Comput. Sci.*, vol. 126, no. 2, pp. 183–235, 1994.

[11] R. Sandhu, E. Coyne, H. Feinstein, and C. Youman, "Role-based access control models," *IEEE Comput.*, vol. 29, no. 2, pp. 38–47, Feb. 1996.

[12] E. Bertino, P. Bonatti, and E. Ferrari, "TRBAC: A temporal role-based access control model," *ACM Trans. Inform. Syst. Security*, vol. 4, no. 3, pp. 191–233, 2001.

[13] J. Joshi, E. Bertino, U. Latif, and A. Ghafoor, "A generalized temporal role-based access control model," *IEEE Trans. Knowledge Data Eng.*, vol. 17, no. 1, pp. 4–23, Jan. 2005.

[14] F. Hansen and V. Oleshchuk, "SRBAC: A spatial role-based access control model for mobile systems," in *Proc. Nordic Workshop Secure IT Syst.*, 2003, pp. 129–141.

[15] E. Bertino, B. Catania, M. L. Damiani, and P. Perlasca, "GEO-RBAC: A spatially aware RBAC," in *Proc. 10th ACM Symp. Access Control Models Technol. (SACMAT)*, 2005, pp. 29–37.

[16] I. Ray, M. Kumar, and L. Yu, "LRBAC: A location-aware role-based access control model," in *Proc. 2nd Int. Conf. Inform. Syst. Security*, 2006, pp. 147–161.

[17] L. Chen and J. Crampton, "On spatio-temporal constraints and inheritance in role-based access control," in *Proc. ACM Symp. Inform. Comput. Commun. Security (ASIACCS)*, 2008, pp. 205–216.

[18] S. Aich, S. Mondal, S. Sural, and A. Majumdar, "Role based access control with spatiotemporal context for mobile applications," *Trans. Comput. Sci.*, vol. 4, pp. 177–199, Apr. 2009.

[19] M. Toahchoodee and I. Ray, "On the formalization and analysis of a spatio-temporal role-based access control model," *J. Comput. Security*, vol. 19, no. 3, pp. 399–452, 2011.

[20] M. Toahchoodee and I. Ray, "On the formal analysis of a spatio-temporal role-based access control model," in *Proc. 22nd Annu. IFIP WG 11.3 Working Conf. Data Appl. Security (DBSec)*, 2008, pp. 17–32.

[21] M. Toahchoodee and I. Ray, "Using alloy to analyse a spatio-temporal access control model supporting delegation," *IET Inform. Security*, vol. 3, no. 3, pp. 75–113, Sep. 2009.

[22] M. Toahchoodee, I. Ray, K. Anastasakis, G. Georg, and B. Bordbar, "Ensuring spatio-temporal access control for real-world applications," in *Proc. 19th ACM Symp. Access Control Models Technol. (SACMAT)*, 2009, pp. 13–22.

[23] B. Shafiq, A. Masood, J. Joshi, and A. Ghafoor, "A role-based access control policy verification framework for real-time systems," in *Proc. 10th IEEE Int. Workshop Object-Oriented Real-Time Dependable Syst. (WORDS)*, 2005, pp. 13–20.

[24] S. Mondal and S. Sural, "Security analysis of temporal-RBAC using timed automata," in *Proc. 4th Int. Conf. Inform. Assurance Security (IAS)*, 2008, pp. 37–40.

[25] K. Sohr, G.-J. Ahn, M. Gogolla, and L. Migge, "Specification and validation of authorisation constraints using UML and OCL," in *Proc. 10th Eur. Symp. Res. Comput. Security (ESORICS)*, 2005, pp. 64–79.

[26] E. Bertino, C. Bettini, E. Ferrari, and P. Samarati, "An access control model supporting periodicity constraints and temporal reasoning," *ACM Trans. Database Syst.*, vol. 23, no. 3, pp. 231–285, 1998.

[27] A. Gal and V. Atluri, "An authorization model for temporal data," in *Proc. ACM Conf. Comput. Commun. Security*, 2000, pp. 144–153.

[28] G.-J. Ahn and R. Sandhu, "Role based authorization constraints specification," *ACM Trans. Inform. Syst. Security*, vol. 3, no. 4, pp. 207–226, 2000.

[29] G. Ahn and M. Shin, "Role based authorization constraints specification using object constraint language," in *WETICE*, pp. 157-162, 2001.

[30] Google, Inc. (2012). *The Android Mobile (OS)* [Online]. Available: www.android.com

[31] Google, Inc. (2012). *Android Developers* [Online]. Available: developer.android.com

[32] FlexiProvider Research Group, Technische Universität Darmstadt. (2012). *The FlexiProvider Toolkit for the Java Cryptography Architecture* [Online]. Available: http://www.flexiprovider.de/overview.html

[33] MySQL, Inc. (2012). *The World's Most Popular Open Source Database* [Online]. Available: http://www.mysql.com/

Ramadan Abdunabi received the M.S. degree from the University of Twente, Enschede, The Netherlands, in 2004, and the M.S. degree from Colorado State University, Fort Collins, USA, in 2010. He is currently pursuing the Ph.D. degree with the Computer Science Department, Colorado State University.

He has published papers on access control models and their analysis. His current research interests include security and software engineering.

Mr. Abdunabi is a member of Upsilon Pi Epsilon.

Mustafa Al-Lail received the B.S. degree in computer engineering from the King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia, in 2004, and the Masters degree in computer science from Colorado State University, Fort Collins, USA, in 2009. He is currently pursuing the Ph.D. degree with the Computer Science Department, Colorado State University.

He has been a Senior Network Engineer with the industry. His current research interests include the verification and validation of software and access control models.

Mr. Al-Lail is a member of Upsilon Pi Epsilon.

Indrakshi Ray (SM'13) received the Ph.D. degree from George Mason University, Fairfax County, VA, USA.

She is currently an Associate Professor with the Computer Science Department, Colorado State University, Fort Collins, USA. Prior to joining Colorado State University, she was a Faculty Member with the University of Michigan-Dearborn, Dearborn, USA. Her current research interests include security and privacy, database systems, e-commerce, and formal methods in software engineering.

Dr. Ray is a member of the ACM.

Robert B. France is currently a Full Professor with the Computer Science Department, Colorado State University, Fort Collins, USA. His current research interests include software engineering, in particular, formal specification, model-based software development, and domain-specific languages.

Prof. France was a recipient of the Ten Year Most Influential Paper Award at MODELS in 2008. He is a Founder and the Editor-in-Chief of the *Springer Journal on Software and System Modeling* and an editor for the *Journal on Software Testing, Verification and Reliability*. He was a member of the early UML 1.x Revision Task Forces.

Specification, Validation, and Enforcement of a Generalized Spatio-Temporal Role-Based Access Control Model

Ramadan Abdunabi, Mustafa Al-Lail, Indrakshi Ray, *Senior Member, IEEE,* and Robert B. France

***Abstract*—With the advent of wireless and mobile devices, many new applications are being developed that make use of the spatio-temporal information of a user to provide better functionality. Such applications also necessitate sophisticated authorization models where access to a resource depends on the credentials of the user and also on the location and time of access. Consequently, researchers have extended the traditional access control models, such as role-based access control, to provide spatio-temporal access control. We improve upon these models by providing additional features that allow us to express constraints that were not possible until now. We express our model using the unified modeling language (UML) and the object constraint language that are the *de facto* specification languages used by the industry. Our model has numerous features that interact in subtle ways. To this end, we show how the UML-based specification environment tool can be used to analyze the spatio-temporal access control model of an application. We propose an architecture for enforcing our model and provide a protocol that demonstrates how access control can be granted and revoked in our approach. We also develop a prototype of this architecture to demonstrate the feasibility of our approach.**

***Index Terms*—Access control.**

I. Introduction

With the growth of the sensor and wireless technology, new applications are being developed for mobile devices. Such applications typically have new authorization requirements where environmental conditions, such as location and time, are used together with the credentials of the user to determine access. An example will help illustrate this point. Consider a real-world example of a spatio-temporal policy for the telemedicine application iMediK [1]. The iMedik is a mobile application accessible by handheld devices that are integrated with a global positioning system (GPS) that identifies its physical location. With the help of mobile devices, doctors can access their patient information on the move. The security policy requires that doctors can use handheld devices to view complete patient medical record (PMR) information in the clinic during daytime, whereas the same doctors can view only partial PMR information outside the clinic during nighttime. Such a policy is needed to protect patient-sensitive information in the case of lost or stolen devices. Traditional access control models, such as role-based access control (RBAC), cannot be used for expressing such policies.

Researchers have addressed this problem by extending RBAC that allows it to do spatio-temporal access control [2]–[4]. Most of the work on spatio-temporal RBAC associate two entities, namely, location and time with users, roles, and permissions. The location and time associated with a user gives the current time and his present location. The location and time associated with a role designate when and where the role can be activated. The location and time associated with a permission signify when and where a permission can be invoked. In addition, researchers have also suggested how spatio-temporal constraints can be associated with inheritance and separation of duty (SoD) relations. Our current work extends the earlier works along several dimensions. First, we provide a more expressive spatio-temporal access control model that we refer to as generalized spatio-temporal role-based access control (GSTRBAC). We allow spatio-temporal constraints to be specified with various types of prerequisite constraints. We also introduce the concept of spatio-temporal zone that allows us to abstract location and time into one entity. This, in turn, reduces the number of entities that must be managed and also prevents the creation of new roles or permissions when spatio-temporal constraints associated with them change.

Second, we demonstrate how our model can be formally represented using the unified modeling language (UML) and object constraint language (OCL) that are the *de facto* specification languages used in the software industry.

Third, we illustrate how the model can be automatically analyzed using the UML-based specification environment (USE) tool for consistency and correctness. An application using our model can also be analyzed using USE to check for security property violations and see how the properties are affected when the security policy is changed.

Fourth, we provide an architecture for enforcing our spatio-temporal access control model. We also provide communication protocols that demonstrate how access can be granted and revoked in the context of our model and prove the security of this protocol. Finally, we implement a prototype and show how our spatio-temporal RBAC can be used in an Android mobile application.

Manuscript received October 9, 2011; revised April 12, 2012; accepted November 3, 2012.

The authors are with the Computer Science Department, Colorado State University, Fort Collins, CO 80523 USA (e-mail: rabdunab@cs.colostate.edu; mustafa@cs.colostate.edu; iray@cs.colostate.edu; france@cs.colostate.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/JSYST.2013.2242751

Treating location and time as separate entities often create problems. Let us illustrate this with an example. Suppose a doctor role can be activated at locations {hospital, clinic} from 8:00 a.m. to 5:00 p.m. This means that the doctor can activate his role either in the hospital or a clinic anytime from 8:00 a.m. to 5:00 p.m. Suppose that the medical board decides to change the spatio-temporal constraints such that the doctor can activate his role in the hospital from 8:00 a.m. to 1:00 p.m. and can activate his role in the clinic from 12:00 p.m. to 5:00 p.m. In order to specify such a constraint, we would have to split the doctor role into two roles, namely, hospital doctor and clinic doctor and associate the respective location and temporal constraints with each of them. Thus, a simple change to the spatio-temporal constraint requires the creation of new roles and changing all the relationships that are associated with the original role. Such a change is nontrivial. Treating location and time as distinct entities also causes a significant increase in the number of entities to be managed as location and time are associated with every object and relation in RBAC. This not only reduces ease of understanding for the user but also makes automated verification more challenging due to state-space explosion.

We introduce the concept of a spatio-temporal zone, henceforth, referred to as STZone. The STZone entity in our model is an abstract concept associated with each RBAC entity and relationship. STZone is represented as a set of pairs for locations and intervals that are of interest to the RBAC entities. In the previous example, the doctor role is initially associated with the following STZones: {(hospital, [8:00 a.m. to 5:00 p.m.]), (clinic, [8:00 a.m. to 5 p.m.])}. When the medical board decides to change the policy, this can be effectuated just by changing the STZones associated with the doctor role as follows: {(hospital, [8:00 a.m. to 1:00 p.m.]), (clinic, [12:00 p.m. to 5 p.m.])}. Abstracting location and time into a single STZone also reduces the number of entities in the model, making it easier to understand and verify.

We formalize GSTRBAC using UML [5] and OCL [6]. A number of reasons motivated our choice. First, UML is a general-propose language that has been considered the *de facto* standard in modeling software. Thus, applications are likely to be specified in UML. This will make it easier for one to integrate the access control policies with the application. Second, UML has a set of graphical notations for specifying static and dynamic aspects of software systems. The graphical diagrams of the UML make it easy to understand and use. Third, UML has supporting tools [7] that can be used for automated analysis. Fourth, UML can be used in all the phases of the software development process. Thus, it will be easy to check whether an access control implementation satisfies the policy if both are specified using the same language.

Once we have specified the GSTRBAC model, it must be analyzed for consistency and correctness. Moreover, for an application using GSTRBAC, we need to ensure that no access control breaches or problems occur. Toward this end, we need to do some automated analysis. Earlier works that use UML to specify access control requirements have typically resorted to the use of other formalisms for automated analysis. Such an approach typically involves a transformation process where the UML is converted into Alloy [8], Coloured Petri Nets [9], or UPPAAL [10] for the purpose of analysis. The results of the analysis depend on the correctness of the transformation procedure. We do not follow this approach but utilize USE [7] for the analysis that allows us to use the same language for specification and verification of GSTRBAC. The USE tool supports the automated generation of snapshot instances, which is used to validate GSTRBAC policies. The USE tool provides an interactive environment that facilitates the validation of properties of UML models specified in the form of OCL invariants, preconditions, and postconditions against some test scenarios. Such test scenarios are automatically generated by the USE tool that makes the verification process easier. The verification is carried out by an embedded constraint solver.

The rest of this paper is organized as follows. Section II enumerates some related work in this area. Section III introduces the concept of spatio-temporal zones. Section IV presents our GSTRBAC model using UML and OCL. Section V shows how USE can be used for analyzing the access control requirements of an application specified using GSTRBAC. In the presentation of the GSTRBAC enforcement mechanism, Section VI discusses the proposed implementation architecture model, Section VII introduces the resource usage protocols, and Section VIII describes an experimental evaluation of a prototype enforcing GSTRBAC in an Android mobile application. Section IX concludes this paper with pointers to future directions.

II. Related Work

RBAC [11] is the de-facto access control model used in the commercial sector. RBAC is policy neutral and can be used to express different types of policies. RBAC simplifies security management. In RBAC, users are assigned to roles, and roles are associated with permissions. In a session, a user can activate a subset of roles assigned to him. The operations that a user can perform in a session depend on the roles activated by the user and the permissions associated with those roles. To simplify role management, roles are organized in the form of a hierarchy. A senior role may inherit the permissions of a junior role, or a senior role may activate the junior role depending on whether the roles are connected by permission inheritance hierarchy or role activation hierarchy. In order to prevent fraud, RBAC allows one to specify separation of duty constraints. Static separation of duty constraints prevents a user from being assigned to conflicting roles or a role being associated with conflicting permissions. Dynamic separation of duty constraints prevent a user from activating two conflicting roles.

Researchers have worked on extending RBAC with time and location. Bertino *et al.* [12] proposed a temporal RBAC. The authors introduced the concept of role enabling and disabling. Roles can be enabled or disabled on the basis of temporal constraints. Roles can be activated if they are enabled. The model does not consider the impact of temporal constraints on user-role assignment or permission-role assignment. This also does not consider the effect of time on separation of duty constraints, cardinality constraints, and role–role hierarchy. Joshi *et al.* [13] proposed a generalized temporal RBAC model that associates temporal constraints with the entities and all the relationships in an RBAC model.

Researchers have also extended RBAC using location information. Hansen and Oleshchuk [14] proposed the spatial RBAC for specifying location-based access control policies for wireless networks. Bertino *et al.* [15] proposed the GEO-RBAC model that allows role activation based on users' locations. However, the model does not discuss the impact of spatial constraints on role hierarchy, separation of duty, user-role assignment, and permission-role assignment. Ray *et al.* [16] proposed location-aware RBAC (LRBAC) model incorporates location constraints in user-role activation, user-role, and permission-role assignments. LRBAC does not define spatial constraints on role hierarchy or separation of duty.

The use of both spatial and temporal information for doing access control has also been investigated by many researchers [2]–[4], [17]–[19]. In all these models, the role activation is constrained by spatio-temporal information. In some models [4], [17], [19], additional spatio-temporal constraints are imposed on user-role assignment and permission-role assignment. Some of these models [17]–[19] also consider the impact of spatio-temporal constraints on role hierarchy and separation of duty. Others [17], [19] also put additional spatio-temporal constraints on permissions. Our current model allows the specification of all the above types of spatio-temporal constraints. In addition, we allow spatio-temporal constraints to be specified with prerequisite constraints—the impact of location and time on prerequisite constraints has not been discussed in any of the previous works; we do this in this paper. We also introduce the concept of STZone that abstracts the location and time into one entity. This simplifies policy management and policy analysis.

Chen and Crampton develop the graph-based representation for the spatio-temporal RBAC in [17]. The RBAC entities are represented by vertices, while their relationships are represented by the edges of a directed graph. The authors propose three types of models: standard, strong, and weak. For 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. For 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 authors developed strong and clear semantics of these different models.

Our work differs from that of Chen and Crampton [17] in the following ways. First, we consider the spatio-temporal impact on separation of duty and prerequisite constraints which is missing from the work of Chen and Crampton. They also did not consider moving objects, which we do in this paper. Second, Chen and Crampton consider the spatial and temporal domains separately before developing the spatio-temporal point. Treating the two domains separately gives rise to the problems described in Section I. Third, Chen and Crampton add spatio-temporal constraints to the RBAC entities and relationships using λ and μ functions, respectively. This approach makes it harder to capture the number, types, and the relationship between the various spatio-temporal constraints. On the other hand, we consider STZone as a separate entity along with the other existing RBAC entities. This allows a more uniform treatment; the STZone pertinent to the application is enumerated and their relationships can easily be evaluated. Fourth, Chen and Crampton provide a graph-theoretic approach for visualizing problems with the specification, but do not focus on the analysis. We use UML and OCL for this purpose; UML and OCL are the *de facto* language for specifying the various requirements of the applications. This makes it easier to analyze the interactions among the access control constraints and also how these impact other application requirements.

In addition to specifying novel RBAC models, researchers have proposed approaches for verifying RBAC security policies. Some researchers [4], [20]–[22] have investigated the use of Alloy for verifying spatio-temporal RBAC policies. In order to make the analysis tractable, Alloy requires that the user scope the problem. The results of the analysis are, therefore, applicable only for the scope of the problem being verified. Modeling and analyzing concurrency in Alloy is nontrivial. Toward this end, researchers [19], [23], [24] have investigated alternative techniques based on Coloured Petri Nets [9] and timed automata [10] for verifying temporal, spatio-temporal, and real-time RBAC policies. The major challenge in these works is how to do the analysis without causing the problem of state explosion. The use of UML and OCL for specification and analysis of RBAC has been investigated by Sohr *et al.* [25]. Our work extends this for specifying and analyzing spatio-temporal policies.

Researchers have also proposed other temporal authorization models that are not based on RBAC. Bertino *et al.* [26] proposed a temporal authorization model that extends authorizations with temporal constraints. In this model, an authorization is associated with a temporal expression, identifying the periods of time in which the authorization applies. Furthermore, it also permits the specification of derivation rules for expressing temporal dependencies among authorizations. Gal and Atluri [27] proposed a temporal data authorization model (TDAM) that can be seen as a complementary model to the one in [26]. TDAM can express time-based policies based on the temporal attributes of data such as transaction time. However, these models are for non-RBAC policies; they cannot express temporal constraints on roles such as temporal role enabling and disabling constraints.

III. Location and Time Representation

In our model, each entity and relation is associated with spatio-temporal information. Before describing these associations in detail, we show how spatio-temporal information is represented in our model.

A. Location Representation

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 3-D geometric space. A physical location $ploc_i$ is a nonempty set of points $\{p_i, p_j, \ldots, p_n\}$, where a point p_k is represented by three coordinates. The granularity of each coordinate 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, CO. 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 corresponding physical one.

Definition 1: **[Mapping Function m]** m is a total function that converts a logical location into a physical one. Formally, $m : L \longrightarrow P$, where P is the set of all possible physical locations and L is the set of all logical locations.

We define the *containment* $\subseteq$ and *equality* = on physical locations. A physical location $ploc_j$ is said to be contained in another physical location $ploc_k$ if $ploc_j \subseteq ploc_k$. Two physical locations $ploc_r$ and $ploc_s$ are equal if $ploc_r \subseteq ploc_s$ and $ploc_s \subseteq ploc_r$.

Note that 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 *anywhere* that contains all other locations. Each application can describe logical locations at different granularity levels. For example, some permissions may be applicable on the entire state whereas other permissions are only applicable to people in the city. Let us denote the logical locations that are of interest to the application by the set **L**. Let the physical locations corresponding to these logical locations be denoted by **P**. The size of the smallest location in **P** corresponds to the minimal location granularity of the application. For example, in the organization Software Development Corporation, we may have **L** = {*MainBuilding, TestingOffice, DirectorOffice, DevelopmentOffice*}. The *MainBuilding* houses the three offices in separate floors of the building. In this case, the minimal location granularity is one floor.

B. Time Representation

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. A time interval is a set of consecutive time instants that can be represented in the form of $d = [t_s - t_e]$, where t_s, t_e represent time instants and t_s precedes t_e on the time line if $t_s \neq t_e$. We use the notation $t_i \in d$ to mean that t_i is a time instant in the time interval d. The exact granularity of a time instant is application dependent. Suppose the granularity of time instant in an application is 1 min. In this case, time interval [3:00 a.m. to 4:00 a.m.] consists of the set of time instants {3:00 a.m., 3:01 a.m., 3:02 a.m., ..., 3:59 a.m., 4:00 a.m.}.

Now, we define the containment $\subseteq$ and equality = on time intervals. A time interval d_j is said to be contained in another time interval d_k if $d_j \subseteq d_k$. 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. The set of all time intervals of interest to the application is defined by **I**. The minimal time granularity of an application refers to the size of the smallest time interval used by the application. For example, in the Software Development Corporation, we may have the following intervals that are of interest: $\mathbf{I} = \{i_1, i_2, i_3, i_4\}$, where i_1 = [8 a.m. to 5 p.m.], i_2 = [8 a.m. to 12 p.m.], i_3 = [12 p.m. to 1 p.m.], and i_4 = [1 p.m. to 5 p.m.]. The minimal time granularity pertaining to this application is 1 hr.

C. Spatio-Temporal Zone

One of the main contributions in our work is the formalization of the concept of a spatio-temporal zone. The concept of spatio-temporal zone abstracts location and time representation into a single entity. Now we give the formal definition of the notion spatio-temporal zone and the zones set.

Definition 2 (Spatio-Temporal Zone): A spatio-temporal zone STZone is a pair of the form (l, d), where l and d represent the logical location and the time interval, respectively.

An example of a spatio-temporal zone can be $z = (HomeOffice, [6\text{ p.m. to }8\text{ a.m.}])$.

A spatio-temporal zone set, i.e., $STZones = \{z_0, z_1, \ldots, z_n\}$, is a set of all spatio-temporal zones in an organization that defines where and when some entities are available. An example of a spatio-temporal zone set is {(HomeOffice, [6 p.m. to 8 a.m.]), (DeptOffice, [8 a.m. to 6 p.m.])}. A spatio-temporal zone (l, d) is specified at minimal granularity if l and d are specified at minimal location granularity and minimal temporal granularity, respectively.

Definition 3 (Spatio-Temporal Zone Containment): A spatio-temporal zone $z_1 = (l, d)$ is contained in another spatio-temporal zone $z_2 = (l', d')$, denoted by $z_1 \subseteq z_2$, if both zones have time intervals containment and locations containment, $d \subseteq d'$ and $m(l) \subseteq m(l')$, where m is the mapping function discussed in Definition 1.

Note that $\{(FortCollins, May2011)\} \subseteq \{(Colorado, Year2011)\}$ since $m(FortCollins) \subseteq m(Colorado)$ and $May2011 \subseteq Year2011$. However, $\{(FortCollins, May2011)\} \not\subseteq \{(Colorado, Year2010)\}$ since $May2011 \not\subseteq Year2010$. Similarly, $\{(FortCollins, May2011)\} \not\subseteq \{(Nevada, Year2011)\}$ because $m(FortCollins) \not\subseteq m(Nevada)$. We define a spatio-temporal zone {(anywhere, always)} that contains all other spatio-temporal zones. For spatio-temporal zones equality =, two zones z and z' are said to be equal, $z = z'$, iff $z \subseteq z'$ and $z' \subseteq z$.

IV. GSTRBAC Model

We now present our GSTRBAC model and formalize its specification using UML and OCL.

A. Effect of Spatio-Temporal Constraints on RBAC Entities

The RBAC entities users, roles, permissions, and objects are associated with spatio-temporal zones.

1) *Users:* We assume that each valid user, interested in doing some location-sensitive operations, carries a locating device that is able to track his location. The location of a user changes with time. The spatio-temporal zone associated with a user gives the user's current location and time.

Note that time and location can have different levels of granularity. For example, the current time can be expressed as 12:00:05 p.m. or 12:00 p.m. Similarly, a user's current location can be Fort Collins or it can be Colorado. The user's current location and time information will be used for making access decisions. Let us illustrate why the notion of minimality must be associated with the user's spatio-temporal zone. Suppose permission is valid in a certain zone. If we do not use the concept of minimal, then it is possible that the user zone may partially overlap with the permission zone. In such a case, should we give access or deny access? On the other hand, if we use the concept of minimal, then the user's zone will either be within the permission zone or outside it. In such cases, we know whether to give or deny access. Consequently, we require the minimal temporal and location be used to express the spatio-temporal zone associated with a user. We define the function $currentzone$ that returns the minimal spatio-temporal zone associated with a user. This function is formally defined as follows:

- $currentzone : Users \rightarrow STZones$.

2) *Objects:* Objects may also be mobile like the user. Here, again, we have locating devices that track the location of an object. Moreover, an object may not be accessible everywhere and anytime. For example, tellers can only review customer information at a teller office during working hours. The $ozones$ function returns the spatio-temporal zones that determine where and when every object is available

- $ozones : Objects \rightarrow 2^{STZones}$.

3) *Roles:* Role can be assigned or activated only in specific locations and time. The role of the on-campus student can only be assigned or activated inside the campus during the semester. The spatio-temporal zone associated with a role gives the location and time from which roles can be assigned or activated. The $rzones$ function gives the set of spatio-temporal zones associated with a given role

- $rzones : Roles \rightarrow 2^{STZones}$.

4) *Permissions:* Permissions are also associated with a spatio-temporal zone that indicate where and when a permission can be invoked. For example, a permission to perform a backup of servers can be executed only from the department after 10 p.m. on Friday nights. The function $pzones$ gives the zones in which a specific permission can be accessed

- $pzones : Permissions \rightarrow 2^{STZones}$.

Fig. 1 shows the class diagram of GSTRBAC. A security policy of a mobile application can be specified as one possible instance of this GSTRBAC class diagram. The GSTRBAC entities: *User*, *Role*, *Permission*, *Object*, *Activity*, and *STZone*, are represented by classes. *Permission* is represented in the GSTRBAC class diagram as an aggregation of the classes *Object* and *Activity*. The *STZone* class aggregates the location and time subclasses. In *STZone* class, *zcontainment* is a reflexive association specifying that a zone can contain other zones. Different relationships between entities, including *UserRoleAssignment*, *UserRoleActivation*, *PermissionRoleAssignment*, *RoleHierarchy*, and *SoD*, are modeled using association classes that are transformed to normal classes following the modeling guidelines in [5] and [6]. These association classes have binary relationships with *STZone* class to enforce the spatio-temporal constraints.

B. Effect of Spatio-Temporal Constraints on RBAC Operations

1) *User-Role Assignment:* A user-role assignment is location and time dependent. That is, a user can be assigned to a role, provided the user is in specific locations. For example, a person can be assigned the on-campus student role only when he is in the campus during the semester. This requirement is expressed using the zone concept

- $UserRoleAssignment \subseteq Users \times Roles \times STZones$.

This relationship is depicted in the GSTRBAC class diagram as association class *UserRoleAssignment*. The OCL operation *assignRole* assigns role r to user u in zone z if z is in the set of *rzones*, user u is present in zone z, and role r is not already assigned to user u in zone z. For the lack of space, we omit the descriptions of the OCL queries used in the *assignRole* operation

```
context User::assignRole(r: Role, z:STZone):
UserRoleAssignment
pre:  r.rzones->includes(z)
pre:  z.containedZones()->includes(self.
currentzone)
pre:  self.getAssignedRoles(z)->excludes(r)
post: self.getAssignedRoles(z)->includes(r).
```

2) *User-Role Activation:* A user can activate a role if the role can be activated on the specific zone and it is already assigned to that user. For example, the role of a doctor trainee can only be activated in a hospital during the training period. We define the *UserRoleActivation* relation to determine the current active roles based on zones

- $UserRoleActivation \subseteq Users \times Roles \times STZones$.

In the GSTRBAC class diagram, *UserRoleActivation* class is specified in a manner similar to *UserRoleAssignment*. The only difference is that the *activateRole* operation ensures that a user is already assigned to a role before the role is being activated.

3) *Check Access:* This operation checks whether a user is authorized to perform some operation on an object during a certain time and from a certain location. A user is allowed to fire a missile if he is assigned the role of a top secret commander and he is in the controller room of the missile during a severe crisis period. Thus, a user can access an object in a certain zone if that user has activated a role that has an appropriate permission for that object in that zone

```
context User::checkAccess(o:Object,a:
Activity,z:STZone):Boolean
post: result = getActivatedRoles(z)->
collect( r | r.getAuthorizedPermissions(z))
->asSet()->
exists( p | p.object=o and p.activity=a and
 o.ozones->includes(z)).
```

4) *Permission-Role Assignment:* Permissions can only be assigned to a role during specific time and locations. For example, the permission of opening a cashier drawer in a store should only be assigned to a salesman role during the daytime. The assignment of permissions to roles is specified based on zones

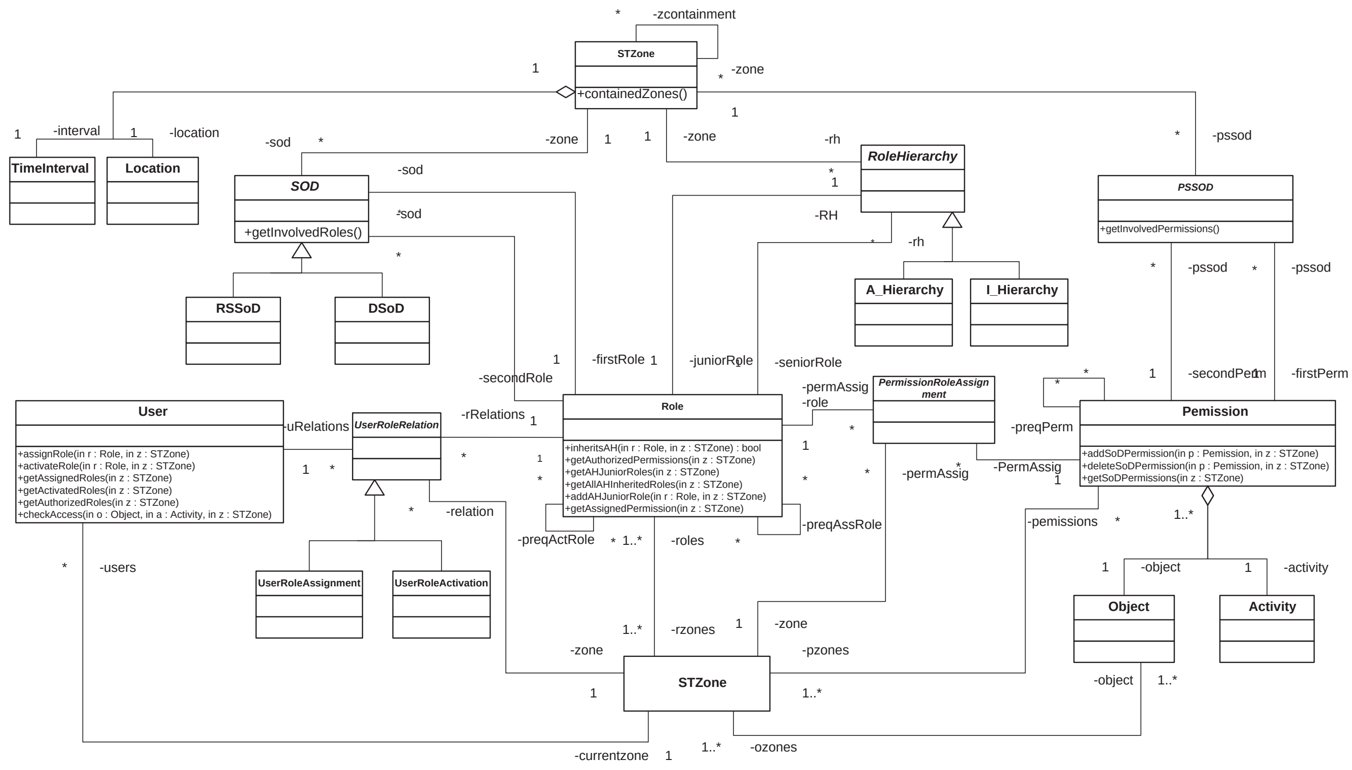

STZone
+containedZones()
-zcontainment
-zone
-interval
-location
TimeInterval
Location
-sod
SOD
+getInvolvedRoles()
RSSoD
DSoD
RoleHierarchy
-rh
-RH
A_Hierarchy
I_Hierarchy
PSSOD
+getInvolvedPermissions()
-pssod
-secondPerm
-firstPerm
User
+assignRole(in r : Role, in z : STZone)
+activateRole(in r : Role, in z : STZone)
+getAssignedRoles(in z : STZone)
+getActivatedRoles(in z : STZone)
+getAuthorizedRoles(in z : STZone)
+checkAccess(in o : Object, in a : Activity, in z : STZone)
-uRelations
UserRoleRelation
-rRelations
-relation
UserRoleAssignment
UserRoleActivation
-users
Role
+inheritsAH(in r : Role, in z : STZone) : bool
+getAuthorizedPermissions(in z : STZone)
+getAHJuniorRoles(in z : STZone)
+getAllAHInheritedRoles(in z : STZone)
+addAHJuniorRole(in r : Role, in z : STZone)
+getAssignedPermission(in z : STZone)
-firstRole
-secondRole
-juniorRole
-seniorRole
-preqActRole
-preqAssRole
-roles
PermissionRoleAssignment
-permAssig
-role
-PermAssig
Pemission
+addSoDPermission(in p : Pemission, in z : STZone)
+deleteSoDPermission(in p : Pemission, in z : STZone)
+getSoDPermissions(in z : STZone)
-preqPerm
-permissions
-object
-activity
Object
Activity
-rzones
-pzones
-currentzone
-ozones

Fig. 1. UML class model for GSTRBAC.

- *PermissionRoleAssignment* $\subseteq$ *Permissions* $\times$ *Roles* $\times$ *STZones*.

The following OCL operation assigns permission p to role r in zone z if z is in the set of $pzones$ and $rzones$:

```
context Role::assignPermission(p:Permission,
z:STZone): PermissionAssignment
pre:   p.pzones->includes(z) and self.rzones->
includes(z)
pre:   self.getAssignedPermissions(z)->excludes(p)
post:  self.getAssignedPermissions(z)->includes(p).
```

C. Spatio-Temporal Role Hierarchy

The permission-inheritance hierarchy (*I-Hierarchy*) and the role-activation hierarchy (*A-Hierarchy*) are two variations of role hierarchy (*RoleHierarchy*) in RBAC [11], [13]. In our model, a senior role could have a subset of junior roles in a particular zone. The spatio-temporal role hierarchies are formally defined as follows.

1) *RoleHierarchy* $\subseteq$ *Roles* $\times$ *Roles* $\times$ *STZones*.
2) *I-Hierarchy* $\subseteq$ *RoleHierarchy*, *A-Hierarchy* $\subseteq$ *RoleHierarchy*, and *I-Hierarchy* $\cap$ *A-Hierarchy* = ϕ.

The subtypes of *RoleHierarchy* are represented in the GSTRBAC class diagram by the subclasses *I-Hierarchy* and *A-Hierarchy*, which are connected to *STZone* class to restrict the roles associations.

1) *Permission-Inheritance Hierarchy:* In a permission-inheritance hierarchy, a senior role r can only inherit junior role r' permissions in zone z if both roles are available in zone z. A project manager inherits the permissions of a developer when he is at the customer site giving a demo. The following OCL expression specifies the spatio-temporal constraint on I-Hierarchy for adding a new junior role:

```
context Role::addIHJuniorRole(r:Role,z:STZone):
I_Hierarchy
pre: self.rzones->includes(z) and r.rzones->
includes(z)
pre:  self.getIHJuniorRoles(z)->excludes(r)
post: self.getIHJuniorRoles(z)->includes(r).
```

The delete operation of a junior role in *I-Hierarchy* can be defined in the similar manner. The *I-Hierarchy* relationship is acyclic as shown by the following OCL constraint:

```
context r1,r2: Role
inv IHierarchy_Cycle_Constraint: not
STZone.allInstances-> exists(z|r1.inheritsIH(r2,z)
and r2.inheritsIH(r1,z)and r1<>r2).
```

The Boolean operation *inheritsIH(r,z)* returns true if role r is directly or indirectly a junior role of the context role in a particular zone; otherwise, it returns false.

```
inheritsIH(r:Role,z:STZone): Boolean =
if (self.getIHJuniorRoles(z)->includes(r))
then true
else self.getIHJuniorRoles(z)->
exists(j | j.inheritsIH(r,z)) endif.
```

We define the OCL query operation *getAuthorizedPermissions(z)* to get the authorized permissions for a given role at zone z through direct assignment or indirect *I-Hierarchy*

```
context Role::getAuthorizedPermissions(z:STZone):
Set(Permission)
Post: result= self.getAssignedPermissions(z)->
union(self.getAllIHInheritedRoles(z)->collect(r |
r.getAssignedPermissions(z)))->asSet().
```

2) *Role-Activation Hierarchy:* Restricted spatio-temporal *A-Hierarchy* allows members of senior roles to activate junior

roles in predefined spatio-temporal zones. For example, a department chair can activate a staff role during the semester inside the department building. The OCL operations of adding and deleting junior roles to the *A-Hierarchy* are defined in similar manner to *I-Hierarchy*. Furthermore, the acyclic constraints on *A-Hierarchy* are enforced in the same way as the *I-Hierarchy*.

The only differences are that the OCL query operation *getAHJuniorRoles(z)* returns all the junior role in *A-Hierarchy* of the context role in particular zone. Moreover, the OCL query operation *getAuthorizedRoles(z)* gives the authorized activation roles for the context user that are either explicitly assigned or implicitly obtained through *A-Hierarchy* in certain zones

```
context User:: getAuthorizedRoles(z:STZone):
Set(Role)
post: result= self.getAssignedRoles(z)->
union(self.getAssignedRoles(z)->collect(r|
r.getAllAHInheritedRoles(z))->
asSet()).
```

D. Spatio-Temporal Separation of Duty

The static SoD (SSoD) and dynamic SoD (DSoD) are two special classes of the SoD constraints in RBAC [28]. Furthermore, the role SSoD (RSSoD) constraints are defined on roles assignment, while the permission SSoD (PSSoD) constraints are defined on permissions assignments.

In our model, the conflicting roles and permissions in SoD are defined over some zones. The spatio-temporal RSSoD, PSSoD, and DSoD relations are formally defined as follows.

1) *RSSoD* $\subseteq$ *Roles* $\times$ *Roles* $\times$ *STZones*.
2) *DSoD* $\subseteq$ *Roles* $\times$ *Roles* $\times$ *STZones*, and *RSSoD* $\cap$ *DSoD* = ϕ.
3) *PSSoD* $\subseteq$ *Permissions* $\times$ *Permissions* $\times$ *STZones*.

The static and dynamic SoD relations are represented in the GSTRBAC class diagram using the associations classes RSSoD, PSSoD, and DSoD, which connect the conflicting entities with certain zones.

1) *Role SSoD:* The same individual should not be assigned to specific roles in a specific location for some duration. For example, the same user should not be assigned to billing clerk and account receivable clerk roles in the same time at a specific trade corporation. The following OCL invariant forbids the assignment of conflicting roles in a particular zone:

```
context User
inv RSSOD_Constaint: STZone.allInstances->forAll( z |
not self.getAssignedRoles(z)->
exists(r1,r2 | r1.getSSoDRoles(z)->includes(r2))).
```

However, the above constraint may be violated through a role hierarchy relation. For example, a billing supervisor may be a senior role of the two conflicting roles billing clerk and account clerk at the same time and in the same accounting department. The following OCL constraint prevents such a situation:

```
context User
 inv RSSOD_RH_Constraint: STZone.allInstances->
 forAll( z | not self.getAuthorizedRoles(z)->
 exists(r1,r2 | r1.getSSoDRoles(z)->includes(r2))).
```

2) *Permissions SSoD:* PSSoD prevents the assignment of conflicting permissions to a role. For example, a loan officer is not permissible to issue loan request and approve it in the bank building during the daytime. The following OCL invariant expresses the PSSoD requirement in our model:

```
context Role
inv PSSOD_Constaint: STZone.allInstances->
forAll( z | not self.getAssignedPermissions(z)->
exists(p1,p2 | p1.getPSSoDPermissions(z)->
includes(p2))).
```

However, this constraint may be violated through *I-Hierarchy*, in which a senior role inherits some junior roles that have mutually been assigned conflicting permissions. The following OCL invariant prevents the violation of PSSoD via *I-Hierarchy*:

```
context Role
inv PSSOD_RH_Constraint: STZone.allInstances->
forAll( z | not self.getAuthorizedPermissions(z)
exists(p1,p2 | p1.getPSSoDPermissions(z)->
includes(p2))).
```

3) *DSoD:* Two conflicting activation roles cannot be activated in some spatio-temporal zones by the same user. For example, the simultaneous activation of cashier and cashier supervisor is forbidden during the working hours in the same store to deter such a user from committing a fraud. The DSoD constraints are expressed in OCL invariants in a similar manner to the RSSoD constraints. The only difference is that the OCL invariants prevent the activations of conflicting roles that are connected by DSoD in some zones through either the explicit role assignment or the implicit *A-Hierarchy*.

E. Spatio-Temporal Prerequisite Constraints

In RBAC, the prerequisite constraints obligate that some actions to be taken prior to performing an operation [29]. Prerequisite constraints impose conditions that must be satisfied before certain assignments, such as user-role assignment and permission-role assignment can be executed. Our spatio-temporal prerequisite constraints can be expressed in first-order logic; consequently, we can represent them using OCL preconditions and invariants.

1) *Prerequisite Constraints on User-Role Assignment:* The prerequisite constraint on roles assignments imposes that a user must be assigned to some less critical roles in a given spatio-temporal zone before being assigned more critical roles in specific zones. For example, the role of emergency nurse can be assigned to John in the urgent care unit from 12:00 a.m. to 5:00 a.m. if he is assigned the role of nurse-on-night-duty at the hospital during those hours. The following OCL invariant expresses the prerequisite constraints on user-role assignment. The query operation *getPreqAssRoles()* returns all the assignment prerequisite roles needed for assigning a certain role

```
context User
inv Prerequiste_URAssign: STZone.allInstances->
forAll(z | Role.allInstances->
forAll(r1 | (self.getAssignedRoles(z)->
includes(r1)) implies (self.getAssignedRoles(z)->
includesAll(r1.getPreqAssRoles())))).
```

2) *Prerequisite Constraints on Permission-Role Assignment:* The prerequisite constraints on permissions assignments indicate that a role can be assigned a permission in a specific zone if some prerequisite permissions are already assigned to that role in the same zone. For example, a bank teller must have the permission of reading an account during working hours before he can be given the permission to update that account. The prerequisite constraint on permission-role assignment can be specified using OCL expression as follows:

```
context Role
inv Prerequist_PRAssign: STZone.allInstances->
forAll(z | Permission.allInstances->
forAll(p1 | (self.getAssignedPermissions(z)->
includes(p1)) implies
(self.getAssignedPermissions(z) ->
includesAll(p1.getPrerequisitePermissions())))).
```

3) *Prerequisite User-Role Activation:* A role can be activated if some prerequisite roles are already activated in specific zones. For example, in a university the teaching assistant role can be activated during a semester in a department if the student role can be activated during the same time. This requirement is specified in our model in the same way of the prerequisite user-role assignment constraint except that the OCL query *getPreqAssRoles()* is substituted with *getPreqActRole()*. The query operation *getPreqActRole()* returns all activation prerequisite roles needed to activate a role.

V. Example Access Control Verification

The GSTRBAC model has many features that may interact with each other, causing conflicts, inconsistencies, and security breaches. In addition, constraints could be specified stronger than needed resulting in some roles, permissions, or objects being inaccessible. Consequently, it is important to analyze GSTRBAC policies before applying them. A manual analysis is tedious and error prone. Toward this end, we propose an automated verification approach based on the USE constraint solver tool. The USE tool accepts three inputs, the GSTRBAC class diagram, the OCL constraints, and the policy instances as object diagrams. When the policy instance does not conform to a GSTRBAC class diagram or it violates some property, the tool pictorially shows how the property has been violated. Specifically, it illustrates the entities and the relationships responsible and how their interactions have caused for the property violation. Once the security designer sees this graphical representation, he can fix the policy specification. For example, if the security designer finds that some prerequisite constraint has been violated, he can either change the prerequisite relation or change the constraint depending on the application requirement.

To illustrate our specification and verification approaches, consider a software development environment for producing military applications. The software project files are stored in computer machines inside a secure building, and the access to those files is location and time dependent. The access control policy of the software development system is specified as follows.

1) Users = {Bob, Ben, Alice, Rachael, Clare, Sam}.
2) Intervals = $\{i_1, i_2\}$. where i_1 = [8 a.m. to 6 p.m.] and i_2 = [6 p.m. to 8 a.m.].
3) Locations = {Home, DevelopmentOffice, TestingOffice, DirectorOffice, DepartmentBuilding}. DepartmentBuilding includes all other offices.
4) STZones = $\{z_0, z_1, z_2, z_3, z_4\}$, where z_0 = (DepartmentBuilding, i_1), z_1 = (Home, i_2), z_2 = (DevelopmentOffice, i_1), z_3 = (TestingOffice, i_1), and z_4 = (DirectorOffice, i_1).
5) Roles = { Software Engineer (SE), Software Programmer (SP), Test Engineers (TE), Programmer Supervisor (PS), Test Supervisor (TS), Project Lead (PL) }.
6) rzones = { (SE, z_0), (SE, z_2), (SP, z_1), (SP, z_2), (TE, z_1), (TE, z_3), (PS, z_2), (TS, z_3), (PL, z_4) }.
7) Objects = { Project Files (obj_1), Test Files (obj_2), Programmer Logs (obj_3), Test Logs (obj_4), Programmer Supervisor Report (obj_5), Test Supervisors Reports (obj_6) }.
8) ozones = { (obj_1, z_1), (obj_1, z_2), (obj_2, z_1), (obj_2, z_3), (obj_3, z_2), (obj_4, z_3), (obj_5, z_4), (obj_6, z_4)}.
9) Activities = $\{read, write, copy, run, review\}$.
10) Permissions = { P_1(read, obj_1), P_2(write, obj_1), P_3(copy, obj_1), P_4(write, obj_2), P_5(run, obj_2)), P_6(review, obj_4), P_7(review, obj_3)), P_8(read, obj_5) }.
11) pzones = { (P_1, z_1), (P_1, z_2), (P_2, z_1), (P_2, z_2), (P_3, z_2), (P_4, z_1), (P_4, z_3), (P_5, z_3), (P_6, z_3), (P_7, z_2), (P_8, z_4) }.
12) UserRoleAssignment = { (Ben, SP, z_1), (Ben, SP, z_2), (Bob, PS, z_2), (Alice, PL, z_4), (Clare, TS, z_3), (Rachael, TE, z_1), (Rachael, TE, z_3), (Sam, SE, z_0) }.
13) PermissionRoleAssignment = { (SP, P_1, z_1), (SP, P_1, z_2), (SP, P_2, z_1), (SP, P_2, z_2), (SP, P_3, z_2), (TE, P_4, z_1), (TE, P_4, z_3), (TE, P_5, z_3), (TS, P_6, z_3), (PS, P_7, z_2), (PL, P_8, z_4) }.
14) I-Hierarchy = { (PS, SP, z_2), (TS, TE, z_3), (PL, PS, z_0), (PL, TS, z_0) }.
15) RSSoD = { (SP, TE, z_0) }.
16) PSSoD = { (P_2, P_4, z_0) }.
17) Prerequisite constraints: The role of software programmer and the test engineer can be assigned if the user is already assigned the role of software engineer.

The policy is graphically represented in Fig. 2 based on the graph notations inspired by Chen and Crampton [17]. We present two scenarios illustrating our analysis approach for verifying security properties.

The first scenario shows how the *checkAccess* operation can be analyzed. Assume that *Ben* is in zone z_3 = $\{(TestingOffice, [8\,\text{a.m. to } 6\,\text{p.m.}])\}$ and tries to *copy ProjectFiles* object. Based on the policy, *Ben* is only assigned to SP in z_1 and z_2. Thus, *Ben* should not be allowed to access that object because the *SP* role is not available for activation in zone z_3. Fig. 3 shows that no policy violation is found with this scenario.

The second scenario considers the verification of the RSSoD constraints. Assume that *Ben* is already assigned to the role SP in the zone z_0, which is a containing zone for z_1 and z_2.

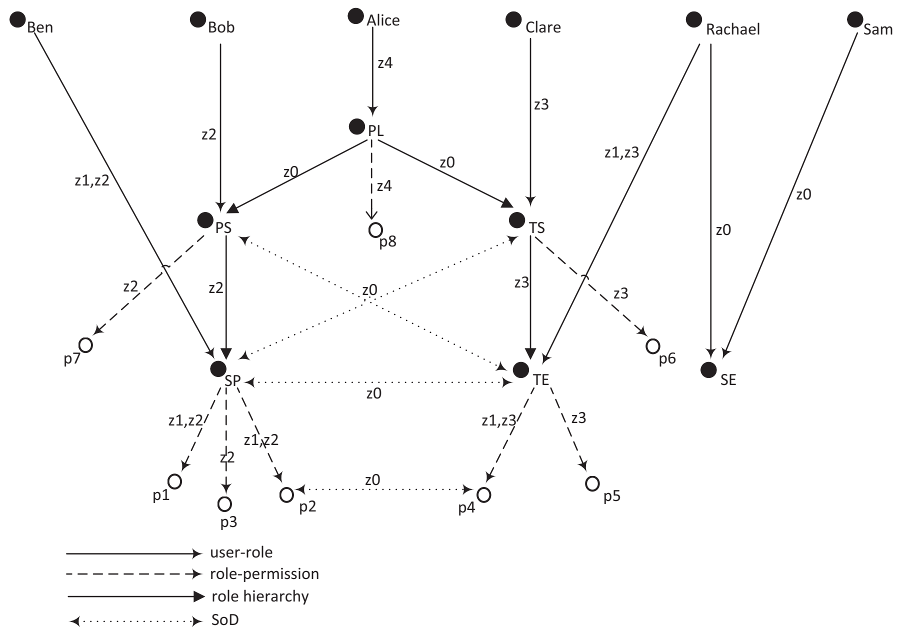
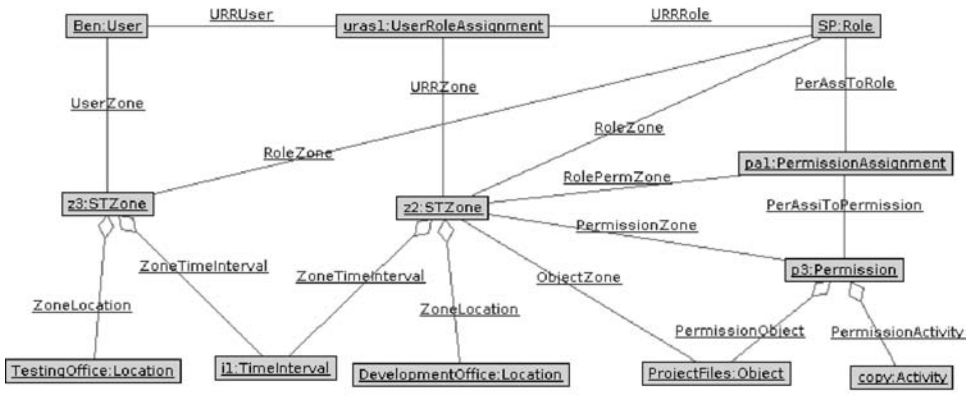
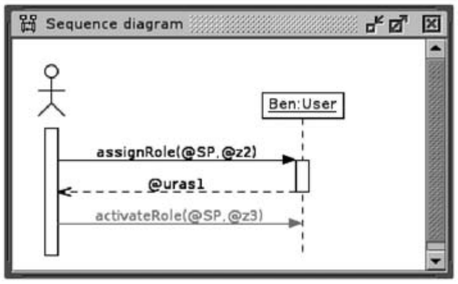
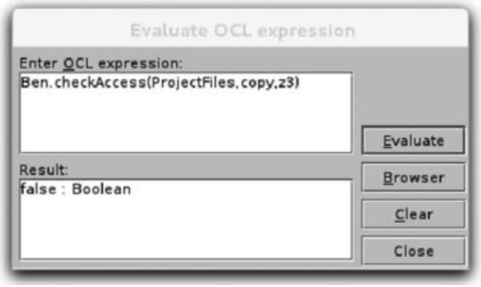


Fig. 2. Access control graph for the development system policy.






Evaluate OCL expression

Enter OCL expression:

Ben.checkAccess(ProjectFiles,copy,z3)

Result:

false : Boolean

Fig. 3. Accessing object from an invalid STZone.

Later on, a security administrator tries to assign Ben to the conflicting role TE in the zone z_0. The assignment operation fails due to the RSSoD constraint, as shown in Fig. 4.

VI. Software Architecture

This section describes a platform-independent implementation architecture, which maps the high-level *GSTRBAC* policy definition to the enforcement mechanism in mobile applications. Later on, we provide an experimental evaluation of this architecture using Android mobile operating system (OS) [30].

Fig. 5 depicts the proposed implementation architecture for enforcing *GSTRBAC* in a mobile application. The architecture consists of three core components: request composition module (RCM), resource access module (*RAM*), and authorization control module (ACM). Each of these modules is a standalone program. The RCM is installed at the user mobile device, the *RAM* and ACM are installed in servers that may or may not be colocated.

1) *RCM* is responsible for forming a user access request and maintaining the access, while the rights are exercised. The *Request Builder* component in RCM creates a

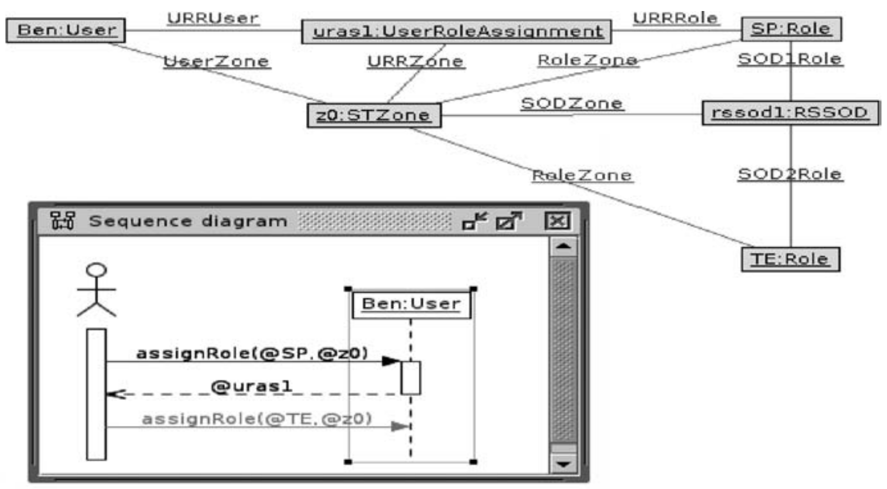


Fig. 4. Conflicting roles assignment.

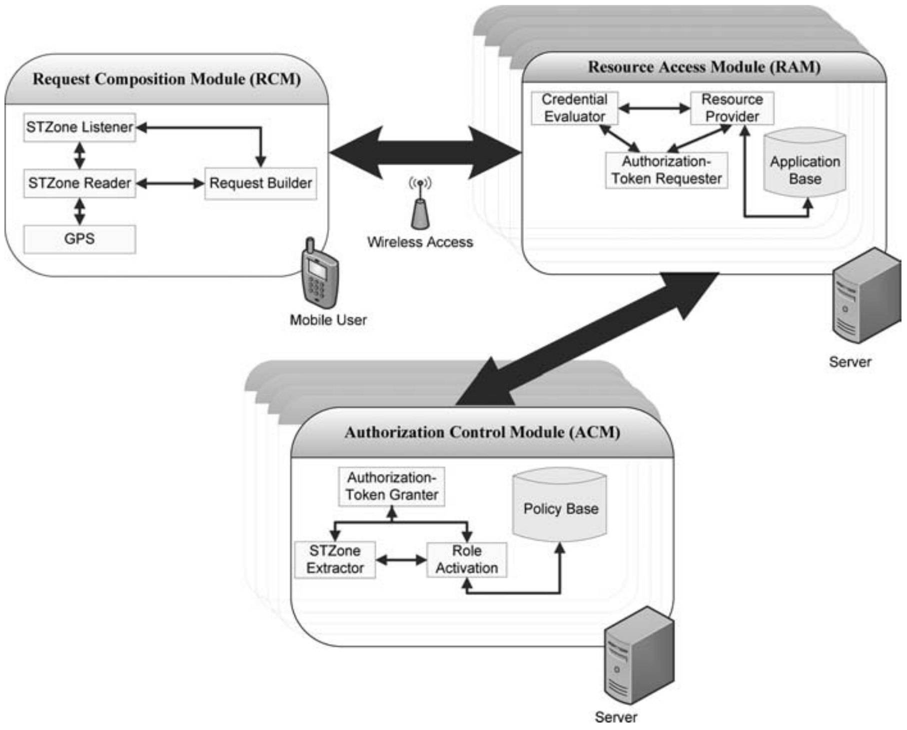


Fig. 5. Implementation architecture of GSTRBAC policy in mobile applications.

resource access request using information obtained from the other components. The *Request Builder* enquires the *STZone Reader* to form the current user *STZone* that encapsulates current user location and time. The *STZone Reader*, in turn, reads the current mobile device time and gets the location from GPS data component storing the location information. After creating the proper access request package, *Request Builder* sends that package to one of the available *RAM* servers. The *STZone Listener* gets the current spatio-temporal information from *STZone Reader* and ensures that the user is in the authorized spatio-temporal zone while the resource is being accessed. Once the user moves outside the authorized zone, *STZone Listener* requests service termination.

2) *RAM* is an intermediate server between the user and ACM server, which is primarily responsible for handling the application resources to the users. *RAM* receives the user request and consults with the ACM server about the user authorization. The user's credentials are stored in the *Application Base* when the user registers with the system. The *Credential Evaluator* evaluates the validity of the user credentials. If the user credentials are valid, the *Authorization-Token Requester* component requests an authorization token (AT) from one of the available ACM servers. Once the AT is granted, it is used by the *Resource Provider* component to provide access to resources needed by the authorized users to accomplish their tasks. *RAM* maintains a list of users' ATs, which

we refer to as *UATokens*, and the elements of this list are continuously updated. A user AT gets deleted from the list once it is expired, which might happen whenever a user deactivates his role or the STZone associated with the role becomes invalid.

3) ACM is accountable for the policy evaluations and tokens generations. Typically, ACM is responsible for issuing a new AT for every role that the user requests to be activated. In order for ACM to evaluate access requests, it consults with the *GSTRBAC Policy Base*. *GSTRBAC Policy Base* stores the access control policy and the authorizations granted to users. The *Role Activation* component in ACM gets the set of roles and permissions that can be activated based on the *STZone* it receives from the *STZone Extractor* component. The *Role Activation* component updates the policy state for each activated role. The *Authorization-Token Granter* component is responsible for granting the AT if a role can be activated. Note that a user's role can be activated only if all the following conditions are satisfied: 1) the role is not already active; 2) the role can be activated in the given *STZone*; and 3) no conflicting roles are already active. If the requested role can be activated, *Authorization-Token Granter* issues a new authorization token with the following format: $AT = (ID_u, ID_{ut}, r, P, STZone)$, where ID_u refers to the user identifier, ID_{ut} is the token's identifier, r is the requested role to be activated, P is the set of the authorized permissions associated with the active role r, and $STZone$ defines where and when these privileges are valid.

VII. Resource Usage Protocols

A. Assumptions

Each mobile device is associated with a user through which the user can access resources. We assume that the users' mobile phones (clients) and servers have tamper-proof storages where the ATs and keys are stored. We also have tamper-proof components that can only be accessed by authorized applications. These components house critical software, such as RCM that cannot be accessed or modified by the unauthorized user. In addition, the client has the needed software to extract the current time and location information. The clients and servers should have the capabilities of executing public key cryptography algorithms and hashing algorithms, such as MD5. We also assume that the existence of a certificate authority is responsible for providing public keys and private keys for the clients and servers. We use timestamps in the protocol, so the different servers and clients must be synchronized. The user identifier and his device identifier must be registered with the RAM servers, and the user must have a unique password.

Fig. 6 describes the communication exchanges of the resource usage protocols. Suffixes associated with the communication messages indicate the order of steps in the protocols. Table I enumerates the notations used in the description of the protocol. Message $\mathbf{M}_i$ indicates Step i of the protocol.

Now we describe the steps of the basic resource usage protocol for handling users' requests.

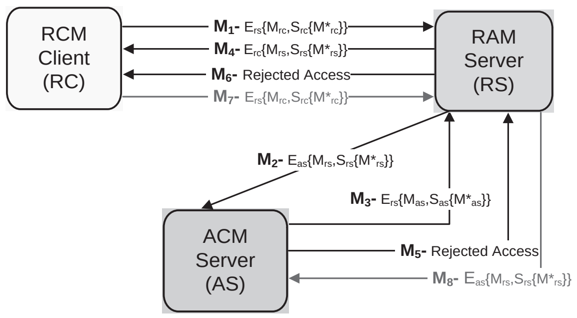


Fig. 6. Communication steps of the resource usage protocols.

TABLE I
The Notations of the Resource Usage Protocols

Symbol	Interpenetration
ID_x	Identifier of party x
ID_s	User's device identifier
PuK_x	Public key of party x
PrK_x	Private key of party x
Ts_x	Timestamp computed by party x
P_u	User password
P_u^*	One-time password
$H(P_u, Ts_x)$	Computes P_u^*
M_x	Package payload created by party x
$E_x\{S\}$	Encryption of sequence S by PuK_x
M_x^*	Message checksum generated by party x
$H(M_x, Ts_x)$	Computes M_x^*
$S_x\{M_x^*\}$	Signing M_x^* by PrK_x
$A \xrightarrow{M_i} B$	Party A sends package M_i to party B
Tw	Time window
AT	Authorization token
ID_{ut}	Authentication-token identifier
"Close"	Keyword indicates deletes user's AT
"Freeze"	Keyword indicates suspends user's AT

1) *Role Activation Request* $[RCM \xrightarrow{M_1} RAM]$: RCM creates an access request payload $M_{rc} = (ID_u, ID_s, P_u^*, STZone, r, Ts_{rc})$, where ID_u is the user identifier, ID_s is the device identifier, $P_u^* = H(P_u, Ts_{rc})$ is the user one-time password, $STZone$ is the current user zone, r is the requested role, and Ts_{rc} is the timestamp at which M_{rc} is created. RCM computes the hash value $M_{rc}^* = H(M_{rc}, Ts_{rc})$ and signs it using the user's private key PrK_u, i.e., $S_{rc}\{M_{rc}^*\}$, to be used as a nonrepudiation proof.

2) *User AT Request* $[RAM \xrightarrow{M_2} ACM]$: On receiving message $\mathbf{M}_1$, RAM decrypts it using its private key. If the message is validated and the user is authenticated, RAM sends message $\mathbf{M}_2$ to the ACM server in order to issue an AT. RAM forwards the AT request payload $M_{rs} = (ID_{rs}, ID_u, STZone, r, Ts_{rs})$, where Ts_{rs} is the timestamp at which M_{rs} payload request is created. RAM computes the hash value of M_{rs} and signs it using its private key PrK_{rs}. Then, RAM encrypts the M_{rs} along with a digitally signed signature $S_{rs}\{M_{rs}^*\}$ using ACM public key PuK_{as}.
If the user authentication fails, RAM sends a rejection message to RCM that is similar to message $\mathbf{M}_6$.

3) *User AT Response* [$ACM \xrightarrow{M_3} RAM$]: At the authorization server, ACM decrypts the package M_2. ACM recomputes the hash value and compares it with the one in package M_2, and verifies the digital signature using the public key of RAM PuK_{rs}.
   In the case the sender of $\mathbf{M}_2$ is authenticated and the user has the rights to activate the requested role, an AT is sent back as a message $\mathbf{M}_3$. $\mathbf{M}_3$ includes the payload M_{as} and a signed hash of M_{as}. The payload $M_{as} = (ID_{as}, ID_{ut}, ID_u, AT, Ts_{as})$ has the user identifier ID_u, user's authorization-token AT, token identifier ID_{ut}, and timesatmp Ts_{as} at which the payload is created.
   If the user's request cannot be granted, an access rejection response is sent to the RAM, as shown in message $\mathbf{M}_5$.
4) *Forwarding User AT* [$RAM \xrightarrow{M_4} RCM$]: After authenticating message $\mathbf{M}_3$ from ACM, RAM stores a copy of the user's AT in the *UATokens* list along with the token identifier ID_{ut}, user identifier ID_u, and device identifier ID_s. Subsequently, RAM forwards an encrypted and signed response message $\mathbf{M}_4$ to RCM. This message includes payload $M_{rs} = (ID_{rs}, ID_u, ID_s, ID_{ut}, AT, Ts_{rs})$ and $S_{rs}\{M^*_{rs}\}$ digital signature signed by the private key PrK_{rs}. $\mathbf{M}_4$ is encrypted using the user public key PuK_u. Note that each user AT is related to a particular user ID_u and a device ID_s.
   However, in the case the access request is rejected by ACM, RAM directly forwards the rejection response in message $\mathbf{M}_6$ to RCM.
   At the user side, if $\mathbf{M}_4$ message from RAM is authenticated, RCM stores the AT in its secure storage.

In our model, we need to revoke access whenever the user moves out of a valid *STZone*. The *STZone Listener* gets periodic updates by the *STZone Reader* about the spatio-temporal coordinates of the user. Whenever the current user location or time does not satisfy the information in a user's AT, the *STZone Listener* revokes the user's AT and requires *Request Builder* to request an access termination.

In Fig. 6, the messages $\mathbf{M}_7$ and $\mathbf{M}_8$ describe the additional exchanges needed to implement the ongoing access protocol.

1) *Terminating User Access* [$RCM \xrightarrow{M_7} RAM$]: The client software sends the termination access request $\mathbf{M}_7$ to RAM at the time the current user *STZone* becomes invalid. RCM creates a termination message $M_{rc} = (ID_u, ID_s, ID_{ut}, P_u, Close, Ts_{rc})$, where the keyword "Close" indicates the termination of access. RCM concatenates M_{rc} with the user digital signature $S_{rc}\{M^*_{rc}\}$ and encrypts them with the resource server public key PuK_{rs}. The user's AT should be deleted at the client side to terminate the user access.
2) *Revoking User Privileges* [$RAM \xrightarrow{M_8} ACM$]: After authenticating the sender of $\mathbf{M}_7$ message, RAM reads the keyword "Close" and then uses ID_u, ID_s, and ID_{ut} to lookup for the user AT in the *UATokens* list and removes it. Therefore, if a user subsequently requests an access to a resource via the same ID_{ut}, his request will be denied because that user does not have the AT for that resource.
   RAM forwards an encrypted and signed termination request $\mathbf{M}_8$ to ACM to delete the user's authorization. This request includes $M_{rs} = (ID_{rs}, ID_u, ID_{ut}, Close, Ts_{rs})$ and the digital signature $S_{rs}\{M^*_{rs}\}$ signed by the RAM private key.
   At the ACM server, after authenticating the sender of $\mathbf{M}_8$, the keyword "Close" indicates ACM to revoke the current user's active role and authorized permissions associated with the user AT ID_{ut}. It does this by updating the GSTRBAC policy state.

A user may temporarily depart the authorized location from which he is currently accessing some resources. To preserve our design efficiency, the user's privileges should not be permanently revoked, but these privileges must be frozen or suspended for the short period of time the user is off-site and they are returned when he is on-site.

We achieve this by modifying the communication steps of the ongoing resource usage protocol to define the suspending resource usage protocol. The client software sends a resource access deferral package $\mathbf{M}'_7$ at the time a user temporarily leaves his valid position.

This message has a similar format to $\mathbf{M}_7$ except that the keyword "Close" is changed to "Freeze" to indicate that user's privileges should be frozen for a certain period. The time window Tw is included in $\mathbf{M}'_7$ message. The time window determines the freezing time during which a user cannot exercise previously granted access rights. Once the user moves back to his previous valid location and prior to the Tw expiration, his roles can automatically be reactivated. If the user does not move to the valid position before the expiration of Tw, his privileges are revoked and message $\mathbf{M}_8$ is sent.

B. Security Analysis

Since our primary concern is protecting applications' resources from improper access, we need to provide a proof that our protocol does indeed protect the resources from unauthorized access. We assume that RAM and ACM are trusted. Furthermore, the RCM is installed in a tamper-proof component of the phone, and it can be only accessed by authorized applications.

In communication protocols, the message confidentiality, message integrity, message authentication, and identity authentication are important. We provide these features in our protocol. Message confidentiality is guaranteed because messages are encrypted by the public key of the recipient. Message integrity is maintained by concatenating each message with the message digest, so that if an intruder intercepts and alters a communication message, the receiver can detect that. The receiver validates the received message by reconstructing the checksum and comparing it to the one in the received message. Message authentication is protected by associating digital signatures with messages; the digital signature provides a non-repudiation proof of the message origin. Identity authentication is provided by using public key certificates and digital signatures. Using the public key certificates and digital signatures, the sender and the receiver are able to mutually authenticate their identities. We assume that private keys used to sign the message digests are only possessed by the signers, and the public key certificates have one-to-one mapping between the public key and the owner. Furthermore, the password-based authentication technique is utilized to provide a proof of identity. Note that the manner in which such techniques are used ultimately determines the security of the protocol.

In the following, we show how our resources are protected from improper access by unauthorized and authorized users. First, we consider the attacks that unauthorized users can carry out to gain an access to protected resources and show how these attacks are prevented in our approach.

1) *Eavesdropping:* An adversary may eavesdrop on the messages that one entity sends to another to breach confidentiality. In this case, even if an adversary eavesdrops on the communication across two entities, he cannot gain useful information as the messages are encrypted by the public key of the recipient.
2) *Modifications:* An adversary may intercept and modify data that one entity sends to another. In order to make this possible, the adversary must be able to read and modify the messages. Since the messages must be decrypted by the adversary before they can be modified, such an attack is not possible in our scheme.
3) *Replay:* In this attack, an attacker intercepts authentic messages coming from a legitimate entity and replays them to gain an access to resources. Even when messages are replayed by an adversary, the timestamps and one-time passwords prevent the replay attacks.
4) *Man-in-the-Middle (MITM)*: In an MITM attack, a malicious user eavesdropping on the communication channel between entities and masquerades as the legitimate entity to the other entity. It intercepts messages coming from a legitimate entity and retransmits them after possibly modifying them to the intended recipient. The use of a public key prevents the attacker from reading and modifying transmitted messages. Consequently, this attack does not occur in this protocol.
5) *Illegitimate Use*: An illegitimate use attack occurs when a malicious user tries to make an access via lost or stolen cell-phones. Since a malicious user will not be aware of the user's password, it cannot send new access requests.

We next consider the attacks that may be carried out by authorized users.

1) *Reusing Authorization-Tokens*: A user may want to access a resource using his past AT. However, when the *STZone* expires, the AT gets deleted from RCM and cannot be used.
2) *Modifying Authorization-Tokens*: A user may modify the AT stored at his site. However, ATs are stored in a secure tamper-proof storage at the user's mobile device and cannot be modified.
3) *Users Collusion*: In this attack, two or more users collude to commit a fraud. That is, user u_1 gets a valid AT for a role and sends it by some means to user u_2 to allow u_2 to violate conflicting constraints. However, this attack is not applicable to our protocols because each AT is exactly linked to a particular device and user.

VIII. Prototype Development

We have developed a proof-of-concept prototype enforcing GSTRBAC in an Android mobile application. Our Android application and servers are written using the Java programming language.

The RCM client is implemented in a mobile application that is developed using free and open source Google's Android (OS) [30]. An Android platform provides a flexible map display and a location service support. Our mobile application utilizes the Android location listener to keep track of the current location coordinates and captures the local time at which the location is fetched. We have used the Android *LocationManager* package [31] for developing an Android location-based services application. Our application retrieves the current position from an enabled GPS receiver, and uses the Google maps application programming interface to display the location on the screen.

The RAM and ACM components are written as traditional Java server programs. To implement the basic resource usage protocol, we have adopted some source code of the *FlexiProvider* Toolkit [32] that has Java-based cryptographic modules, including the public key, the digital signature, and the MD5 message digest. The toolkit implements fast and secure cryptographic algorithms and can be plugged into many Java applications. Prior to running the experiment, each entity should have its private key and stores the public key of the communicated endpoints in a local file. We use the *KeyPairGenerator* class to generate public and private key pairs. The public keys are securely distributed using the AES symmetric-key encryption. Furthermore, all communicating parties implement the same MD5 algorithm.

We used the relational database to represent the application and policy data. The relational database is implemented in the open source database MySQL server [33]. We have two relational database tables: the first one is the application table that stores login information, and the second one is the policy table that stores a GSTRBAC policy. The application table is only accessible by RAM via a MySQL server, while the policy table is connected to ACM.

We tested our mobile application through the Android emulator, and the test showed that our prototype works as expected. The Android handset emulator is depicted in Fig. 7 displaying our Android applications. The handset emulator prompts a user to select his role and enters the user's identifier and password. Once the user enters these information and hits the connect button, the Android application software fetches the last known user location and the local time, then it composes an access request and sends it to one of the available virtual RAM servers in a secure manner.

In order to use the location capabilities of Android, we used the *LocationManager* class to access to the Android location services. The *locationManager.requestLocationUpdates()* method updates the device's location every some fixed period of time through the location provider GPS. A class implement-

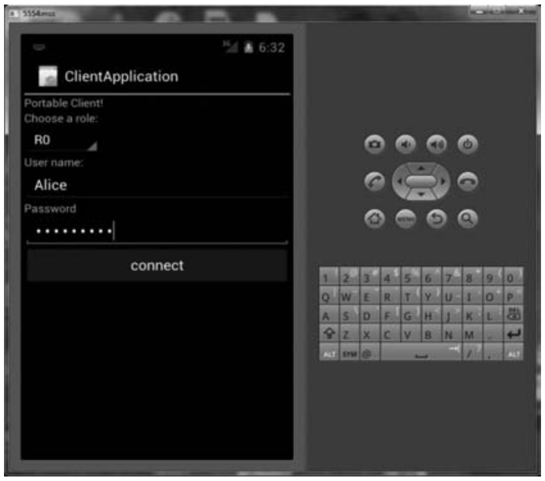


Fig. 7. Android handset emulator.

TABLE II
BACK-END AVERAGE RESPONSE DELAY

Response	Average Delay
0-Approved	73.66 ms
1-Rejected (improper login data)	29.56 ms
2-Rejected (improper role)	67.50 ms
3-Rejected (improper zone)	81.43 ms
Total Average Delay	63.04 ms

ing the *LocationListener* interface handles changes in the device location. The *locationManager.getLastKnownLocation()* method fetches the last known location object that has the altitude, latitude, and longitude information. When the Android emulator starts for the first time, it reports the current location *Null* because there is no last known location to fetch. Thus, we used *DDMS* view of Eclipse to manually feed mock location information to the Android emulator. Once the emulator's GPS device has the dummy data, the listener is triggered to retrieve the current location. Now, the Android SDK's emulator can emulate user's location changes. Then, the current user's coordinates is viewed on a Google map and the location name is manually fed into our application.

In our experiment, we instantiated three *RCM* handset Android emulators, three *RAM* virtual servers, and three *ACM* virtual servers. All of these execute on a single machine and communicate through sockets. We also created another virtual server running a local centralized MySQL database on the same machine. This database server is connected with the servers to process users' requests. The Android handset emulator sends an access request to one of the virtual *RAM* servers, and the *RAM* servers connect with *ACM* servers to get the access authorizations. For each request, the handset emulator opens a new connection with one of the virtual *RAM* servers' names stored in a local file, and it closes the connection at the time it receives a response. The *RAM* server opens a new connection with one of the *ACM* servers' names stored in a local file only if the user login information is correct, and it closes the connection at the time it gets a response form the endpoint *ACM* server.

To evaluate different spatio-temporal access scenarios, we have stored the logical locations and role names in two distinct local files. Thus, for each request, the handset emulator randomly selects a location name and a role name from these files and sends them along with other information in a request package. This approach allows us to test whether our application works as anticipated and validates the policy correctness. We measured the time from issuing an access request to the time when the response is received. The response delay is evaluated for 150 requests sent simultaneously from three Android handset emulators, each emulator sent 50 requests. The responses vary based on the information in the request packages. For example, a request gets approved only if the login information is correct, the requested role can be authorized, and the current user's zone is acceptable; otherwise, the request is rejected.

The experiments are carried out on Windows 7 platform running on Intel(R) Core(TM) 2 Due CPU at 2.20 GH with 4.00 GB *RAM*. The results in Table II exhibit the response average delay for each class of responses and the total average response delay in milliseconds. The overall average response delay yielded by the basic resource usage protocol is 63.04 ms. Furthermore, the rejected requests due to invalid login information yield 29.56 ms that is the smallest response delay because *RAM* servers send the responses immediately without consulting the *ACM* servers.

IX. CONCLUSION AND FUTURE WORK

In this paper, we proposed GSTRBAC that allows specification of role-based access control policies that are based on time and location. We introduced the concept of a spatio-temporal zone that encapsulated temporal and spatial constraints that facilitated understanding, analysis, and policy evolution. The model had many features that interacted with each other in subtle ways. Toward this end, we showed how an application using our model can be analyzed using the constraint solver embedded in USE. We proposed an architecture and developed a prototype for a mobile system enforcing GSTRBAC model. We developed a number of protocols that consider spatio-temporal information for initiating and maintaining access under different circumstances.

Much work remains to be done. In the current work, we presented arguments demonstrating that certain types of attacks that led to access control breaches did not occur in our protocol. Often times, formal analysis may reveal problems that are not apparent in the informal analysis. Consequently, to provide more assurance, we plan to formally analyze the security protocols using existing tools, such as Coloured Petri Nets [9] and Alloy [8]. We also plan to extend our spatio-temporal access control model for workflows that consist of a set of tasks that are coordinated by control-flow, data-flow, and temporal dependencies. It would be interesting to see how these various dependencies interact with the spatio-temporal constraints of the workflow. Our future work also includes deploying this model for a real-world dengue decision support application.

REFERENCES

[1] A. Maji, A. Mukhoty, A. Majumdar, J. Mukhopadhyay, S. Sural, S. Paul, and B. Majumdar, "Security analysis and implementation of web-based telemedicine services with a four-tier architecture," in *Proc. 2nd Int. Conf. Pervasive Comput. Technol. Healthcare*, 2008, pp. 46–54.

[2] I. Ray and M. Toahchoodee, "A spatio-temporal role-based access control model," in *Proc. 21st Annu. IFIP WG 11.3 Working Conf. Data Appl. Security (DBSec)*, 2007, pp. 211–226.

[3] S. Aich, S. Sural, and A. Majumdar, "STARBAC: Spatio temporal role based access control," in *Proc. On The Move to Meaningful Internet Syst. (OTM)*, 2007, pp. 1567–1582.

[4] A. Samuel, A. Ghafoor, and E. Bertino, "A framework for specification and verification of generalized spatio-temporal role based access control model," Purdue Univ., West Lafayette, IN, USA, Tech. Rep. CERIAS TR 2007-08, Feb. 2007.

[5] G. Booch, J. Rumbaugh, and I. Jacobson, *The Unified Modeling Language User Guide*, 2nd ed. Reading, MA, USA: Addison-Wesley Professional, 2005.

[6] L. Craig, *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development*, 3rd ed. Englewood Cliffa, NJ, USA: Prentice-Hall, 2004.

[7] M. Gogolla, F. Büttner, and M. Richters, "USE: A UML-based specification environment for validating UML and OCL," *Sci. Comput. Programming*, vol. 69, nos. 1–3, pp. 27–34, 2007.

[8] D. Jackson, "Alloy: A lightweight object modelling notation," *ACM Trans. Software Eng. Methodol.*, vol. 11, no. 2, pp. 256–290, 2002.

[9] K. Jensen, L. Kristensen, and L. Wells, "Coloured petri nets and CPN tools for modelling and validation of concurrent systems," *Int. J. Software Tools Technol. Transfer*, vol. 9, nos. 3–4, pp. 213–254, 2007.

[10] R. Alur and D. Dill, "A theory of timed automata," *Theor. Comput. Sci.*, vol. 126, no. 2, pp. 183–235, 1994.

[11] R. Sandhu, E. Coyne, H. Feinstein, and C. Youman, "Role-based access control models," *IEEE Comput.*, vol. 29, no. 2, pp. 38–47, Feb. 1996.

[12] E. Bertino, P. Bonatti, and E. Ferrari, "TRBAC: A temporal role-based access control model," *ACM Trans. Inform. Syst. Security*, vol. 4, no. 3, pp. 191–233, 2001.

[13] J. Joshi, E. Bertino, U. Latif, and A. Ghafoor, "A generalized temporal role-based access control model," *IEEE Trans. Knowledge Data Eng.*, vol. 17, no. 1, pp. 4–23, Jan. 2005.

[14] F. Hansen and V. Oleshchuk, "SRBAC: A spatial role-based access control model for mobile systems," in *Proc. Nordic Workshop Secure IT Syst.*, 2003, pp. 129–141.

[15] E. Bertino, B. Catania, M. L. Damiani, and P. Perlasca, "GEO-RBAC: A spatially aware RBAC," in *Proc. 10th ACM Symp. Access Control Models Technol. (SACMAT)*, 2005, pp. 29–37.

[16] I. Ray, M. Kumar, and L. Yu, "LRBAC: A location-aware role-based access control model," in *Proc. 2nd Int. Conf. Inform. Syst. Security*, 2006, pp. 147–161.

[17] L. Chen and J. Crampton, "On spatio-temporal constraints and inheritance in role-based access control," in *Proc. ACM Symp. Inform. Comput. Commun. Security (ASIACCS)*, 2008, pp. 205–216.

[18] S. Aich, S. Mondal, S. Sural, and A. Majumdar, "Role based access control with spatiotemporal context for mobile applications," *Trans. Comput. Sci.*, vol. 4, pp. 177–199, Apr. 2009.

[19] M. Toahchoodee and I. Ray, "On the formalization and analysis of a spatio-temporal role-based access control model," *J. Comput. Security*, vol. 19, no. 3, pp. 399–452, 2011.

[20] M. Toahchoodee and I. Ray, "On the formal analysis of a spatio-temporal role-based access control model," in *Proc. 22nd Annu. IFIP WG 11.3 Working Conf. Data Appl. Security (DBSec)*, 2008, pp. 17–32.

[21] M. Toahchoodee and I. Ray, "Using alloy to analyse a spatio-temporal access control model supporting delegation," *IET Inform. Security*, vol. 3, no. 3, pp. 75–113, Sep. 2009.

[22] M. Toahchoodee, I. Ray, K. Anastasakis, G. Georg, and B. Bordbar, "Ensuring spatio-temporal access control for real-world applications," in *Proc. 19th ACM Symp. Access Control Models Technol. (SACMAT)*, 2009, pp. 13–22.

[23] B. Shafiq, A. Masood, J. Joshi, and A. Ghafoor, "A role-based access control policy verification framework for real-time systems," in *Proc. 10th IEEE Int. Workshop Object-Oriented Real-Time Dependable Syst. (WORDS)*, 2005, pp. 13–20.

[24] S. Mondal and S. Sural, "Security analysis of temporal-RBAC using timed automata," in *Proc. 4th Int. Conf. Inform. Assurance Security (IAS)*, 2008, pp. 37–40.

[25] K. Sohr, G.-J. Ahn, M. Gogolla, and L. Migge, "Specification and validation of authorisation constraints using UML and OCL," in *Proc. 10th Eur. Symp. Res. Comput. Security (ESORICS)*, 2005, pp. 64–79.

[26] E. Bertino, C. Bettini, E. Ferrari, and P. Samarati, "An access control model supporting periodicity constraints and temporal reasoning," *ACM Trans. Database Syst.*, vol. 23, no. 3, pp. 231–285, 1998.

[27] A. Gal and V. Atluri, "An authorization model for temporal data," in *Proc. ACM Conf. Comput. Commun. Security*, 2000, pp. 144–153.

[28] G.-J. Ahn and R. Sandhu, "Role based authorization constraints specification," *ACM Trans. Inform. Syst. Security*, vol. 3, no. 4, pp. 207–226, 2000.

[29] G. Ahn and M. Shin, "Role based authorization constraints specification using object constraint language," in *WETICE*, pp. 157-162, 2001.

[30] Google, Inc. (2012). *The Android Mobile (OS)* [Online]. Available: www.android.com

[31] Google, Inc. (2012). *Android Developers* [Online]. Available: developer.android.com

[32] FlexiProvider Research Group, Technische Universität Darmstadt. (2012). *The FlexiProvider Toolkit for the Java Cryptography Architecture* [Online]. Available: http://www.flexiprovider.de/overview.html

[33] MySQL, Inc. (2012). *The World's Most Popular Open Source Database* [Online]. Available: http://www.mysql.com/

Ramadan Abdunabi received the M.S. degree from the University of Twente, Enschede, The Netherlands, in 2004, and the M.S. degree from Colorado State University, Fort Collins, USA, in 2010. He is currently pursuing the Ph.D. degree with the Computer Science Department, Colorado State University.

He has published papers on access control models and their analysis. His current research interests include security and software engineering.

Mr. Abdunabi is a member of Upsilon Pi Epsilon.

Mustafa Al-Lail received the B.S. degree in computer engineering from the King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia, in 2004, and the Masters degree in computer science from Colorado State University, Fort Collins, USA, in 2009. He is currently pursuing the Ph.D. degree with the Computer Science Department, Colorado State University.

He has been a Senior Network Engineer with the industry. His current research interests include the verification and validation of software and access control models.

Mr. Al-Lail is a member of Upsilon Pi Epsilon.

Indrakshi Ray (SM'13) received the Ph.D. degree from George Mason University, Fairfax County, VA, USA.

She is currently an Associate Professor with the Computer Science Department, Colorado State University, Fort Collins, USA. Prior to joining Colorado State University, she was a Faculty Member with the University of Michigan-Dearborn, Dearborn, USA. Her current research interests include security and privacy, database systems, e-commerce, and formal methods in software engineering.

Dr. Ray is a member of the ACM.

Robert B. France is currently a Full Professor with the Computer Science Department, Colorado State University, Fort Collins, USA. His current research interests include software engineering, in particular, formal specification, model-based software development, and domain-specific languages.

Prof. France was a recipient of the Ten Year Most Influential Paper Award at MODELS in 2008. He is a Founder and the Editor-in-Chief of the *Springer Journal on Software and System Modeling* and an editor for the *Journal on Software Testing, Verification and Reliability*. He was a member of the early UML 1.x Revision Task Forces.