ERBAC: Event-Driven RBAC*

Piero Bonatti Clemente Galdi Davide Torres

May 26, 2022

Abstract

Context-aware access control systems should reactively adapt access control decisions to dynamic environmental conditions. In this paper we present ERBAC, an event-driven extension of the TRBAC model that allows the specification and enforcement of general reactive policies. A variety of examples illustrate ERBAC's expressive power, and its ability of handling exceptional situations in a flexible way, while keeping policies compact and manageable. Then we extend XACML to support the new model, and illustrate a prototype implementation of the PDP. Experiments show that the computational cost of policy rule evaluation is compatible with real-world applications.

1 Introduction

Role Based Access Control (RBAC) models [39, 37, 38, 30] have received considerable attention in the last couple of decades. Classical authorisation frameworks assign permissions directly to users. Such a direct allocation is difficult to manage. For example, administrators must ensure that all users that are meant to perform the same tasks are given the same set of permissions; moreover every change in the policy associated to a task must be reflected on a multitude of users and objects.

The RBAC paradigm solves the above issues by interposing roles between users and permissions. In other words, permissions are assigned to roles and roles are assigned to users. This greatly simplifies the tasks of adding and removing users, as well as changing their role in the organization. The success of RBAC models is also due to the fact that roles are a natural way of representing job functions within an organization [39, 16].

With the advent of pervasive/ubiquitous/mobile systems, the access control problem has become more complex. In such systems, the decision of whether a permission should be granted or not may depend on a number of dynamic environmental conditions. First of all, such a decision might depend on *where* the user is, i.e., the current user position. Depending on the application scenario, the space might be divided into several logical areas. On one hand the access control system might be used to control the access of users to restricted areas, e.g., *accessing an operating room in a hospital should be allowed only to authorized personnel and patients*. On the other hand, the user location might be used as a condition for granting access to some resources, e.g., *an employee can read her work email marked "confidential" only whenever she is within specific offices in one of the company buildings*. A second important variable is *when* access is requested. An example is the case in which the policy needs to limit the access to some resource during specific time intervals, e.g., *the bank vault should be open every working day between 8:00 a.m. and 4:00 p.m.*

Fortunately, the ability of interacting with the environment is a characterizing feature of pervasive and ubiquitous systems. Such systems are able to obtain information from

*Preliminary version appears in [10]. This version of the article has been accepted for publication, after peer review, and is subject to IOS Press terms of use, but is not the Version of Record and does not reflect post-acceptance improvements, or any corrections. The Version of Record is available online at: https://doi.org/10.3233/JCS-150539

the environment and possibly use them to influence it. The events detected by monitoring the invironment may be exploited to adapt access control policies to particular—possibly exceptional—situations. Consider the following simple example: *Fire extinguishers should be, under normal circumstances, inaccessible; In case of fire, everybody who is close to the fire should be able to access fire extinguishers for 30 minutes after the alarm.* Such a scenario clearly describes the interplay between the environment and spatio-temporal conditions. If the environment does not sense any fire, nobody is able to access fire extinguishers. In order to express this policy, a language is needed where policy rules can be triggered by a rich range of environmental conditions and events. Furthermore, in the example, access permission is limited only to a category of resources (extinguishers) that are spatially and temporally bounded (*close* to the fire and *for the first 30 minutes from the alarm*).

Our Contributions In this paper we extend the RBAC model by presenting a framework that supports access control policies in which *environmental conditions* are used to drive role enabling/disabling. In order to keep the maximum possible flexibility, an environmental condition is seen as a triple consisting of a temporal, a spatial, and an *event* condition. The first two conditions address the above discussion. The *event condition* will be used to model *every measurable context variable that can influence the access decision.* Such a generic definition covers event conditions ranging from *a user enters a new location* (related to user mobility) to alarms like *fire in room X* or *patient Y having a heart attack.* In order to help the reader, we collected the glossary of main terms and concepts in Table 4.

The expressiveness of ERBAC and its ability of handling exceptional situations will be amply illustrated in Sec. 7, by means of a variety of examples, mainly focussed on medical scenarios.

Then we will show how XACML should be extended to cover the ERBAC model, and describe a prototype implementation that extends Sun Microsystems' implementation of XACML [42]. An extensive set of experimental results shows that rule evaluation overhead is moderate, and compatible with many application scenarios.

Organization of the paper. In Section 2 we review previous relevant works in the field. In Section 3 we briefly report the formal definition of the basic RBAC model. In Section 4 we present the basic components that we use in order to represent context conditions. In Section 5 we report the basic terminology and syntax that we use to specify access control policies. In Section 6 we define the formal semantics of the model, and in Section 7 we give a number of examples demonstrating how our system can be used to write context-dependent policies. We further provide a classification of the ERBAC model in the UCON context defined in [31]. In Section 8 we describe the prototype we have implemented and report the results of our experimental analysis. Finally Section 9 reports some concluding remarks and open issues.

2 Previous Works

In role based access control models [36], permission are assigned to users indirectly, through roles. Each permission can be assigned to different roles. Similarly each role can be assigned to different users. This model makes permission-to-users assignment extremely flexible for a number of reasons. First of all, roles can be immediately identified within an organization as they typically match specific jobs, e.g., an accountant, a driver, a branch manager, etc. Given a role, the set of permissions that should be assigned to it can be determined with the need-to-know principle (i.e., by identifying a minimal set of permissions that enables the role's tasks to be performed). Whenever a user's responsibility is modified, e.g., she is promoted, it is sufficient to update the set of roles that the user can enable. Furthermore, whenever the organization's policy changes, it is sufficient to modify the association of permissions to

roles. In order to reflect the (typically non-flat) organization chart, roles can be organized *hierarchically*, i.e, each role inherits all permissions that are beneath it in the hierarchy.

The classical RBAC model may be too coarse-grained for some applications. For example, if role *Doctor* is authorized to read patient records, then every user playing this role can read the medical records of all patients; it is not possible to restrict the permissions of any specific doctor to the data of her own patients only. This limitation has been addressed in [18] by introducing the concept of *parametrized privilege*, that is a privilege that can be granted to a particular user only if some condition holds. A restricted privilege can be seen a triple $(op, o, exp(v_1, \ldots v_n))$, where op is an operation, o is a set of elements and $exp(v_1, \ldots v_n)$ is a logical expression, $v_1, \ldots v_n$ is a set of variables. Whenever a parametric privilege is instantiated, i.e., a value is assigned to every variable $v_1, \ldots v_n$, the boolean expression identifies a subset of the objects in o on which the user is permitted to execute the operation op. Parametric privileges support the formulation of generic access control policies that are instantiated for specific users and objects by suitably binding their variables. For example $(rw, documents, document.branch = x)$ corresponds to the privilege of executing *read/write* operations on the *documents* belonging to the branch x, where x is the free variable. Such a (general, parametrized) privilege can be defined once for every branch in the organization. At the same time, it will guarantee that every manager will have the proper access to all the documents in her branch. Finally *role templates* are roles defined by using parametrized privileges.

This model has been enriched by introducing user location [15, 32] and temporal conditions [7, 8, 19]. Further context-aware RBAC models can be found in [5, 33, 34, 6, 14, 11, 17, 19, 41, 35, 20, 21]. These models are briefly discussed in the following.

In [14], the authors introduce the concept of environmental role in the Generalized RBAC model, (GRBAC) as a mean for reducing the complexity of access control systems in ubiquitous computing. An environmental role is an abstract role that is activated whenever a given set of environmental variables assume specific values, e.g. *temperature in room X in the morning below 32* $^\circ F$. In this model a permission, that can be associated with a set of environmental roles, becomes available to users whenever environmental conditions allow the activation of one of the roles in the set. In [13] the authors provide an implementation of GRBAC where access control is managed by means of environmental roles.

In [20, 21] the authors present Context Aware RBAC (CA-RBAC) and a framework that allows the specification of context aware policies. The system is characterized by *applications*, each of which can define its own set of roles. Each role is defined by considering context-information like temporal and spatial conditions, pre-conditions that have to be met before the role can be activated and/or context events that describe specific environmental conditions. It is possible to associate to each event a *context guard*, i.e., a condition that has to be verified whenever the environmental condition changes.

In [22, 23] the authors consider a specific application scenario for time-critical applications, namely context-aware access control for patient monitoring systems. In time-critical applications, it should be possible to trade the privacy of the information for the time needed to obtain an access control decision, i.e., against the criticality of current context. In these papers the authors compose different access control policies ranging from quick but coarse decisions in critical conditions to slow but fine-grained ones in normal conditions. Clearly, context-aware decisions are typically hard to achieve if many sensors need to be queried each time an access request is made.

In [5, 4, 33, 34, 11] the authors present models in which both a spatial condition s and temporal condition t are managed as a single spatio-temporal pair (s, t). Recently, in [3], a new model has been introduced in which the spatial component and the temporal component are merged into a single *spatio-temporal zone*. This modification on the one hand simplifies the human interpretation of policies and their automatic verification, but, on the other hand, makes policy specification longer.

Our work is not subsumed by any of the above contributions. We cleanly separate policies from the application logic, while [17, 35, 20, 21] don't. In our reactive framework, the environment is monitored continuously (through an event-based mechanism) while [5, 33, 34, 17, 41, 35] check policy conditions only once before granting access. Joshi [19] deals only with time and [35] only with space. Neumann [41] does not provide a formal model. Finally, [5, 33, 34, 6, 14, 17, 19] do not illustrate any implementation, and [35] provides no experimental evaluation.

3 Preliminaries

The RBAC model we are going to extend in this paper has been introduced in [36] and consists, basically, of the following components: A set of users U, a set of roles R, a set of permission P and a set of sessions S. A user in U is a human or artificial agent, that can execute operations in the systems, each of which requires a specific set of permissions. A non-empty set of operations defines a job within an organisation. A role can be seen as a set of permissions that are needed to execute a given job within an organisation. Finally, sessions associate users to roles. After the authentication phase, the system creates a new session during which the user can require the activation of the roles she is allowed to play. Role activation is successful only if the required role is enabled and the user is entitled to activate it. In this case, the user is granted all the permissions associated to the activated role. The model comprises a number of functions defined below. The user assignment (UA) and the permission assignment (PA) functions are used to associate users and permissions to roles, respectively. As stated above, a user can be associated with many roles and every role can be played by many users. Similarly, a permission can be associated with many roles and a role can include many permissions. The user function maps each session to a single user, whereas the function role establishes a mapping between a session and a set of roles. It is possible to define a hierarchy on the set of roles by means of the RH relation. If $(r_i, r_j) \in RH$, then role r_i inherits the permissions of role r_j. Formally the set of functions is defined as follows.

Definition 1 *The RBAC model consists of the following components:*

- U, R, P *and* S*, Users, Roles, Permissions and Sessions;*
- $PA \subseteq P \times R$*, A function associating permissions to roles;*
- $UA \subseteq U \times R$*, A function associating users to roles;*
- $RH \subseteq R \times R$*, a partially ordered role hierarchy;*
- $user : S \rightarrow U$*, a function associating each session to the user who started it;*
- $roles : S \rightarrow 2^R$*, a function that, for each session, computes the set of active sessions; for all sessions* s_i*,* $roles(s_i) \subseteq \{r \mid (user(s_i), r) \in UA\}$*. The set of privileges granted in a session* s_i *is given by* $\cup_{r \in roles(s_i)}\{p \in P \mid (p, r) \in PA\}$*.*

In this paper we will extend one of RBAC successful extensions, the TRBAC [8] whose components will be detailed in the next Section.

4 The Model

In this section we present the formal model for ERBAC. We will first introduce context representation and, then, we will use such a definition for extending the classical TRBAC model.

4.1 Context Representation

The context in which an action takes place is identified by three major features: *location* that describes where the action takes place; *time* which specifies when the action takes place; and *events*, that describe other relevant measurable features of the context that may influence the access control decision process.

4.1.1 Time Representation

For the representation of temporal assertions we will use the classical formalism introduced by TRBAC [8], which we recall briefly to make the paper self-contained.

Informally a temporal expression is represented as a pair (I, P), where $I = [begin, end]$ identifies a time interval and P is a *periodic expression* that is used to specify repetitions of the interval I. Periodic expression P are specified by using the concept of *calendar*, that is a countable set of contiguous intervals, numbered by integers called *indexes* of the intervals. We say that C_1 is a sub-calendar of C_2, $C_1 \sqsubseteq C_2$ in symbols, if each interval in C_2 is exactly covered by a finite number of intervals in C_1. In other words, for each interval I in C_2, there exist a set of intervals in C_1 whose union covers I. As in [8] we assume the existence of calendars, *hours, days, weeks, months and years* such that $hours \sqsubseteq days \sqsubseteq \ldots \sqsubseteq years$.

Let C be a calendar and $O \in 2^{\mathbb{N}} \cup \{all\}$ be its set of indices. We will denote by "$O \cdot C$" the set of intervals, defined in C whose indices correspond to the ones specified in O. If $O = all$, $all \cdot C$ denotes all the intervals in C. Formally we can define the following:

Definition 2 *Given calendars $C_1, \ldots, C_n$ e C_d, a* periodic expression *P is defined as:*

$$P = \sum_{i=1}^{n} O_i \cdot C_i \rhd r \cdot C_d$$

where $O_1 = all, O_i \in 2^{\mathbb{N}}, C_i \sqsubseteq C_{i-1}$ for $i \in [2, n], C_d \sqsubseteq C_n$ e $r \in \mathbb{N}$.

The symbol $\rhd$ separates the set of starting points of the intervals from the specification of the duration of each interval. For example, $all \cdot Years + \{1, 3\} \cdot Months \rhd 2 \cdot Weeks$ represents the set of all intervals starting at the beginning of the first and third months of every year, and having a duration of two weeks.

Given the above definition we can define:

Definition 3 (Temporal Expression) *A temporal expression identifies a set of periodical intervals and is represented by a pair (I, P) where:*

- *I=[begin,end] is an interval starting at time begin and ending at time end.*
- *P is a periodic expression.*

From TRBAC we further inherit the following functions:

Definition 4 *In the time-domain we define the following functions:*

- *$Sol(I, P)$: Given a temporal expression (I, P), $Sol(I, P)$ denotes the (explicit) set of time intervals identified by the succinct representation.*
- *time(): Returns the current time.*

4.1.2 Space Representation

A position in space can be described with three, orthogonal, representations. The first two have already been used in the context of time-space access control ([33, 34, 5]). The location of a device can be specified by its position within a Cartesian coordinate system. In this case, the *physical location* of the device is represented, say, by a triple of coordinates (x, y, z). We will denote by PL the set of physical locations. A second type of representation is strictly application dependent. The *logical location* of a device is an abstract notion like "Building A" or "Cardiology". The set of logical locations will be denoted by LL. We further define the logical locations "AnyPlace" and "NoPlace" that represent, respectively, the whole space (managed by an application) and the "empty" space. We assume the existence of a space hierarchy that characterizes containment relation among logical locations. We write $l_1 \subseteq_{LL} l_2$ whenever logical location l_1 is considered as a subspace of l_2.

Given the above, a physical location can be mapped to different logical locations since, for example, a point in space can belong to different logical locations (e.g., room, floor, building, etc). Similarly, a logical location can describe different physical locations. Notice that a logical location can describe different elements in different logical locations. For example the logical location "room x" can be part[1] of "floor y" and, at the same time, be part of "department z". Clearly the same argument can be extended to sets of physical locations. This means that, from the point of view of an application, the same (logical or physical) location is characterized by some extra information, its *type* that will be used to fine-tune context-dependent access control. Following [5], we will define the *type* of a logical location as an abstract notion characterizing its features. We will denote by LT the set of location types.

Definition 5 *In the location domain we define the following functions:*

- $PtoL : PL \rightarrow 2^{LL} \cup \{\perp\}$. *A partial function associating to each physical location the set of logical locations that contain it.*
- $LtoT : LL \rightarrow 2^{LT}$. *This function associates to each logical locations a set of type of locations that it represents.*
- $LofT : LT \rightarrow 2^{LL} \cup \{\perp\}$. *Partial function associating to each type of location the set of logical locations matching the given type.*
- $UserPos : U \rightarrow PL$. *This function returns the physical location of a given user.*

It is important to notice that, if there exists some sort of hierarchy over physical and logical space descriptions, then there should be a way of inheriting such a hierarchy also in the space of location types. Furthermore, such a hierarchy on location types should not merely reflect geometrical containment since there might be cases in which priorities need to be assigned to different types of locations. As an example we can consider the types "operating room" and "patient room". (Typically) there exists no containment relation between rooms of these types, neither at the physical nor at the logical level, but clearly, every logical location that is an "operating room" should be subject to more restrictive access control policies w.r.t. patient rooms.

Thus there are cases in which there exists a hierarchy in the logical organization of space. We say that a location type T_1 is *more specific* than another type T_2 whenever either T_1 is a finer-grained than T_2 in logical or physical terms or T_1 has higher "priority" w.r.t. T_2. In the previous examples, location types "Operating room" and "patient room" are more specific than "room", and "floor" is more specific than "building". To formalize the hierarchy over location types, we introduce a partial order ($\preceq_{LT}$) defined over the set LT. We further introduce a top element (TOP_{LT}) and a bottom element (BOT_{LT}).

[1] We will define containment operations shortly.

Definition 6 *Location type ordering is defined as a partial order over the set of location types, $(LT, \preceq_{LT})$ such that:*

1. *The relation $\preceq_{LT}$ is a partial order over LT. If $lt_1, lt_2 \in LT$ and $lt_1 \preceq_{LT} lt_2$ then lt_1 is more specific than lt_2;*
2. *LT contains the elements TOP_{LT} e BOT_{LT} such that:*
   - $\forall lt \in LT : BOT_{LT} \preceq_{LT} lt \preceq_{LT} TOP_{LT}$.
   - $LofT(TOP_{LT}) = \{AnyPlace\}$
   - $LofT(BOT_{LT}) = \{NoPlace\}$

Since each logical location can be associated with a number of location types, the relation $\preceq_{LT}$ might not be sufficient for determining which of two given logical locations l_1 and l_2 is more specific. The reason for this impossibility is, intuitively, that there might exist *some* location types for l_1 that are more specific than *some* location types for l_2 and viceversa, i.e., there exist *some* location types for l_2 that are more specific than *some* location types for l_1. This apparent inconsistency can be easily explained if we consider the following example. The entrance of a building has a high priority for the "building security management" (who are meant to control the building access doors) while it has essentially no priority for everyone else working in the building (since they know that security is taking care of it). On the other hand, offices within the building have high priority for white collars while they have low priority for security staff.

Given the above discussion we define a *containment* relation between logical locations as follows:

Definition 7 *Let $l_1, l_2 \in LL$. We say that l_1 is* more specific *in l_2, ($l_1 \subset_{LL} l_2$ in symbols) if the following hold:*

- $l_1 \subseteq_{LL} l_2$.
- *There exists some location type lt_i for l_1 that is more specific of some location type lt_j for l_2:*
  $\exists lt_i \in LtoT(l_1) \wedge \exists lt_j \in LtoT(l_2) : lt_i \prec_{LT} lt_j$;
- *No location type lt_j for l_2 is more specific of any location type for l_1:*
  $\forall lt_i \in LtoT(l_1) \wedge \forall lt_j \in LtoT(l_2) : lt_j \not\prec_{LT} lt_i$.

We further define a notion of proximity based on the space representation just discussed. We notice that a possible definition of proximity for two physical locations p_1, p_2 might be the one based on the (Euclidean) distance between the two locations. Clearly such a definition does not consider constraints posed by the environment. For example, consider the case in which two "close" (w.r.t. the Euclidean distance) physical locations are separated by a wall. The existence of such an obstacle does not allow to move "directly" from one location to another.

Definition 8 *Let $pl_1, pl_2 \in PL$. We say that pl_1 and pl_2 are* close *if the following holds:$\exists ll_1 \in PtoL(pl_1) \cap PtoL(pl_2)$ such that $\forall ll_2 \in PtoL(pl_1) \cup PtoL(pl_2)$ it holds that $ll_2 \not\subset_{LL} ll_1$*

Intuitively, two physical locations are close if they both belong to the same logical location whose type is the most specific, i.e., the one that guarantees the maximum possible granularity for describing both locations.

4.1.3 Event Representation

With the term *event* we denote all the aspects of the context that are different from time and space and that are monitored and/or measured. An event can describe, for example "surgery in progress" or "open door". Events can be classified as *localized* or *global*. An event is localized if it is associated somehow to a specific location, e.g., a "surgery in progress" has an impact only in the "operating room" in which the surgery is performed. An event is *global* otherwise, e.g., a blackout is, in principle, a global event since it can affect the whole space.

We further classify the events based on their generation. In particular an event can be *system generated* or *user generated*. An example of the former is the automatic opening of a door while, for the second one, we can think to a medical emergency request by some patient.

Finally we can classify the events based on the set of users it affects. In particular, an event is *personalized* if it affects *some* users while it is *general* if it affects *all* the users.

We will denote by E the set of all possible events. We associate to each event a priority, that is a natural number in the set $Prios$. We will denote by Min_P and Max_P the minimum and maximum priority in $Prios$, respectively.

To formalize the above discussion, we will use the following:

Definition 9 *Let U be the set of users. In the event domain we define the following functions:*

- *$generatedBy : E \rightarrow U \cup \{\perp\}$. Partial function associating to each event the user who generated it.*
- *$generatedFor : E \rightarrow 2^U \cup \{\perp\}$. Partial function associating to each event the set of users to whom it is addressed.*
- *$Eloc : E \rightarrow PL \cup \{\perp\}$. Partial function associating to each event the physical location where it has been generated.*
- *$EVisible : E \rightarrow LL \cup \{\perp\}$. Partial function associating to each event the set of logical locations where the event is visible.*
- *$Eprios : E \rightarrow Prios$. Partial function associating to each event its priority.*

The following table contains the taxonomy of events as described above.

Table 1: Event Taxonomy

Property	**Name**	**Condition**
Visibility	Global	$Eloc = \perp$
	Localized	$Eloc \neq \perp$
Generator	System Generated	$GeneratedBy = \perp$
	User Generated	$GeneratedBy = u \in U$
Target Users	General	$generatedFor = \perp$
	Personalized	$generatedFor \subseteq U$

From the above discussion, it is clear that every event is, in principle, visible only to a subset of user. We will consider two types of visibility. In the first one, the generation of an event in a specific physical location needs to be "propagated" to some extent in the spatial hierarchy. Consider the case of a fire in an office. Although the event is localized, it is reasonable to assume that it will became visible to all the users to other levels in the hierarchy, say in the whole floor or in the whole building. Thus, on the one hand, we have a spatial hierarchy that maps a physical location p into a set of logical locations, $PtoL(p)$,

containing p. On the other hand, every localized event ev, generated in a physical location p is associated with a set of logical locations in which it should be visible, i.e., $EVisible(ev)$. We assume a sort of consistency property: $EVisible(ev) \subseteq PtoL(Eloc(ev))$. Such a condition simply states that each event can only affect locations that contain the physical location in which the event has been generated.

A second type of visibility is related to the "presence" of a user in a specific place. Consider, for example, the case of a medical emergency. It is reasonable to assume that such an event is visible to doctors that are "close" to the event generation location. Similarly, it does not make much sense to alert a doctor that is miles away from the event location. We will discuss events of this type in Section 7.8.

Notice that a non-localized event is, by definition, visible everywhere. Finally, personalized events should be visible only to user to which it is addressed.

4.1.4 Overall context description

Given the above discussions and definitions we are now ready to formally define a context state in terms of time, space and events. Temporal conditions can be expressed either in the form of periodic intervals (I, P) or by means of the element "AnyTime". More formally:

Definition 10 *The set of temporal condition TCond is such that:*

- $(I, P) \in TCond$
- $AnyTime \in TCond$
- $tcond \in TCond \Rightarrow \neg tcond \in TCond$

A time instant t satisfies a temporal condition $tcond \in TCond$, denote by $t \models tcond$, if (a) $tcond = AnyTime$ or (b) $tcond = (I, P)$ and $t \in Sol(I, P)$ or (c) $tcond = \neg(I, P)$ and $t \notin Sol(I, P)$.

Similarly we can define spatial conditions as follows.

Definition 11 *The set of spatial conditions SCond is such that:*

- $\forall ll \in LL, ll \in SCond$
- $\forall tl \in LT, tl \in SCond$
- $AnyPlace \in SCond$
- $scond \in SCond \Rightarrow \neg scond \in SCond$

Given a physical location pos, let $Locs(pos) = PtoL(pos) \cup \{LtoT(loc) \mid loc \in PtoL(pos)\}$. We say that pos satisfies $scond \in Scond$ (written, $pos \models scond$) if (a) $scond = AnyPlace$ or (b) $scond \in Locs$ if $scond \in LL \cup LT$ or (c) $scond = \neg A$ and, for all $l \in Locs(pos)$, $l \neq A$.

Using the same strategy we can define conditions on events as follows:

Definition 12 *The set of event conditions ECond is such that:*

- $\forall ev \in E, ev \in ECond$
- $AnyEvent \in ECond$
- $econd \in ECond \Rightarrow \neg econd \in ECond$

A set of events evs satisfies an event condition $econd \in ECond$, in symbols $evs \models econd$, if one of the following holds: (a) econd=AnyEvent or (b) $econd \in evs$ if $econd \in E$ or (c) $econd \notin evs$ if $econd = \neg e$ for some $e \in E$.

A condition on the context can be expressed as the Cartesian product of $TCond$, $SCond$ and $ECond$, i.e., $STECond \subseteq TCond \times SCond \times ECond$. A conditional expression will be evaluated as function of the current values of the attributes. Thus a context satisfies an $STECond$ triple (s, t, e) if the current space, time and event satisfies the conditions expressed by (s, t, e), respectively. More formally,

Definition 13 *Let $c = (s, t, e)$ be a triple identifying a physical location s, a time instant t and a set of events. Let $cond = (sc, tc, ec) \in STECond$, where $sc \in SCond, tc \in TCond$ and $ec \in ECond$. We say that c satisfies $cond$ ($c \models cond$) if and only if $s \models sc$, $t \models tc$, and $e \models ec$.*

5 Syntax

In this paper we will use role templates and parametrized privileges in order to make privileges context-dependent. For this reason, we need to redefine the sets P (privileges) and R (roles) of the $RBAC$ model.

We will denote by OBJ the set of objects/resources managed by the access control system. A category is defined as a set of objects and the set of all categories will be denoted by OC.

Since objects and categories may be mapped arbitrarily, we define the following functions:

Definition 14 *In the object domain we define:*

- *$ObjInCat : OC \rightarrow 2^{OBJ}$. Function mapping each category to the set of objects it containes.*
- *$ObjCat : OBJ \rightarrow 2^{OC}$. Function mapping an object to the set of categories to which it belongs to.*

Since we will use parametrized privileges, we need to define a privilege in terms of an operation, a category and an expression.

Definition 15 *Let OPS be the set of operations. A* privilege *is represented by the triple $(op, oc, exp(v_1, \ldots, v_n))$, where $op \in OPS$, $oc \in OC$ and $exp(v_1, \ldots, v_n)$ is a logical expression over unbound variables $v_1, \ldots, v_n$. We will denote by P the set of privileges.*

An example of the expressiveness induced by the above definition is the following. It is possible to restrict read access to all the objects in the category "Medical Records" that contain a specific "department" attribute by using the parameterized privilege as follows:

$$(read, PatientRecords, record.department = \text{“}x\text{”})$$

Similarly, we will use role templates as a means for defining generic, parameterized roles in our model. We will denote by R_T the set of role templates and by R_I the set of role instances (obtained by instantiating the templates). The specification of role templates by means of parametrized privileges provides a flexible way of defining families of different roles with similar behavior.

Example 1 (Role Definition) *Let $PatientRecord$ be the object category that contains all the Patient Records for a given hospital. Each such record contains a number of fields such as patient's name, birthdate, SSN, department and so forth. Furthermore each record can be either read or written by some operator.*

We can use role templates to write access policies in a compact way. For example, we may allow each doctor to read each patient record in his own department using the following definition.

$$
\begin{array}{l}
Doctor\langle x\rangle = role(\\
\quad (read, PatientRecord, record.department = \text{“}x\text{”}) \\
)
\end{array}
$$

The above template defines an access policy that is inherently replicable for every department. Given such a template, the role instance that will be instantiated for a specific doctor in a specific department will be, for example, $Doctor\langle cardiology\rangle$ will have the privilege:

$$(read, PatientRecord, record.department = \text{“}cardiology\text{”})$$

■

In the classical RBAC model, roles are activated whenever an entitled user requests their activation. According to [8, 5], a role's life-cycle is partitioned into two subsequent steps. Initially all roles are *disabled*. Role enabling modifies the role state from disabled to *enabled*. Enabled roles are ready for the actual activation that occurs automatically at the first permitted access request. More generally, a context-aware system needs a way to control role enabling and disabling based on context-dependent conditions. We will use the following approach:

Definition 16 (Event Expression, Role Status Expression)
Let $(Prios, \leq_P)$ be a totally ordered set of priorities.

- *(Simple) Event Expression: “*enable *r” or “*disable *r” are used to enable or disable a role $r \in R$; the syntax “*enable *r* for *u”, “*disable *r* for *u” is used to enable or disable a role $r \in R$ for the specific user u. The set of simple event expressions will be denoted by $SEXP$.*

- *Prioritized Event Expression: These expressions have syntax $p : ev$, where $p \in Prios$, with $p \leq_P Max_P$, and ev is a simple event expression. The set of prioritized event expressions will be denoted by $PEXP$.*

- *Role Status Expression: In the form “*enabled *r” or “*¬enabled *r” denote, respectively, the enabled or disabled state for role r. The set of role status expressions will be denoted by $REXP$.*

Given the above definition we can define the conflicts between expressions.

Definition 17 *Let exp be an event expression in $SEXP$. The conflicting event $conf(exp)$ is defined as:*

- $conf$(enable r) = disable r
- $conf$(disable r) = enable r
- $conf$(enable r for u) = disable r for u
- $conf$(disable r for u) = enable r for u

Finally, access control policies can be encoded by coupling an event expression with a context-condition in what we call the *Conditional Event Expression*. From the TRBAC model we inherit the concept of *role trigger* in order to support the cascading enabling/disabling of roles as an effect of the enabling/disabling of other roles. For the sake of self-containment, we report the original definition.

Definition 18 *A* Conditional Event Expression *is an expression* $\langle cond, ev \rangle$*, where* $cond \in STECond$ *and* ev *is a simple or prioritized event expression. A simple event expression is interpreted as prioritized event expression with minimal priority. The set of conditional event expression is denoted by* $CEXP$*.*
A Role Trigger *is an element in the form:*

$$E_1, \ldots, E_n, C_1, \ldots, C_k \rightarrow p : E \text{ after } \Delta t$$

where $E_1, \ldots, E_n \in SEXP$*,* $C_1, \ldots, C_k \in REXP$*,* $p : E \in PEXP$ $(p <_P Max_P)$ *and* Δt *denotes the offset after which it is possible to evaluate* $p : E$*. The set of role trigger is denoted by* $TRIGGERS$*.*

A Role Enabling Base *(REB) is a set of conditional event expressions and role triggers.*

Example 2 (Conditional Event Expression) *Let us assume the following temporal expression:*

$$WorkingHours = All.Years + All.Months + All.weeks + \{1-5\}.Days + \{8\}.Hours \triangleright 8.Hours$$

Furthermore, let Cardiology *be a department in a given hospital and let* $Doctor\langle\cdot\rangle$ *be the template defined in Example 1. The following conditional event expression can be used to describe the enabling of the role instance in the cardiology department.*

$$\langle (WorkingHours, Cardiology, AnyEvent), enable\ Doctor\langle Cardiology\rangle\rangle$$

Such enabling rule will be executed whenever an employee, that has been authenticated using the credentials of a doctor of the Cardiology department, enters any physical location of such a department, during working hours. We assume that user authentication is guaranteed by standard tools, and do not discuss this issues here. ■

The above example defines the enabling rule for a specific department within a hospital. A trivial way of enforcing the same policy over all the departments would be replicating the rule for all departments. On the other hand, if the space is properly classified within the space hierarchy, we are able to write more compact and general policies.

Example 3 *Let us assume that the hospital is logically partitioned into different departments, e.g., Cardiology, Neurology, etc. Furthermore, let us assume that Department is a location type, i.e.,* $Department \in LT$ *with the property that each of the above logical locations is of type Department, i.e.,* $Cardiology \in LL$*,* $Department \in LtoT(Cardiology)$*. Then the access policy described in Example 2 can be extended to every department in the hospital by using the following rule:*

$$\langle (WorkingHours, Department, AnyEvent), enable\ Doctor\langle PtoL(UserPos) \cap LtoT(Department)\rangle\rangle$$

■

Security managers may ask the access control system to enforce some specific one-time behavior, e.g. enabling/disabling a specific role, independently from the current context. Such possibility was provided in TRBAC by *run-time requests.*

Definition 19 *A* Run-time Request Expression *is an expression* $p : ev$ *after* Δt*, where* $p \in Prios$*,* $ev \in SEXP$ *and* Δt *denoted the temporal offset after which the request* $p : ev$ *must be executed. A simple event expressions* ev *is intended to have maximum priority, i.e,.* $Max_P : ev$*.*

A run-time request expression can assume maximum priority. This means that, in principle, this type of expression can override other context-dependent prioritized event expressions. The sequence of run-time requests (labelled with their arrival time) is called *Request Stream* and is formalized as follows:

Definition 20 *A* Request Stream *(RQ) is an infinite sequence:*

$$RQ = \langle RQ(1), \ldots, RQ(t), \ldots \rangle$$

where $\forall t > 0, RQ(t)$ *is the set of run-time request received at time* t*.*

6 Semantics

6.1 Priorities and conflict resolution

In complex Role Enabling Bases, a context may trigger conflicting event expressions. Before defining the semantics of the ERBAC model, we must specify how to resolve such conflicts. Recall that each event expression is associated with a priority and, thus, we stipulate that the event expressions with higher priority take precedence. However the system may activate two (or more) conflicting event expressions with the same priority simultaneously. In this case, we adopt the strategy *denials take precedence*. This strategy can be restated as follows: a role is enabled only if there exists no disabling expression for the same role whose priority is greater than or equal to one of the enabling expression.

Definition 21 *Let* S *be a set of event expressions,* $r \in R$ *and* $u \in U$*. An expression* $p : ev \in S$ *is said to be* blocked *by* S *if there exists* $q \in Prios$ *such that* $q : conf(ev) \in S$ *and one of the following holds:*

1. $ev = enable\ r\ [for\ u] \wedge (p \leq_P q)$
2. $ev = disable\ r\ [for\ u] \wedge (p <_P q)$

The set of event expressions in S *that are* not *blocked by* S *will be denoted by* $NonBlocked(S)$*.*

Thus, given a set of event expressions S, the system actually processes at every time t, the set of $NonBlocked(S)$. Such set can depend on three distinct elements: the conditional event expressions, the role triggers in the role enabling base and the run-time requests received at time t. The only elements that depend on the context are the conditional event expression.

Priorities, in general, do not suffice to remove all conflicts. The *specificity* of event expressions provides an additional criterion for conflict resolution that refines explicit priorities. A first, natural specificity criterion is based on spatial conditions.

Definition 22 (Specificity in SCond)
Let $sc_1, sc_2 \in SCond$*. Spatial condition* sc_1 *is said to be* more specific *than* sc_2*, denoted by* $(sc_1 <_{SCond} s_2)$ *is it holds that:*

- $sc_1 \subset_{LL} s_2$*, if* $sc_1, sc_2 \in LL$
- $sc_1 \preceq_{LT} sc_2$*, if* $sc_1, sc_2 \in LT$
- $\begin{array}{c} \forall tl_1 \in LtoT(sc_1) : sc_2 \not\preceq_{LT} tl_1 \\ \wedge \\ \exists tl_1 \in LtoT(sc_1) : tl_1 \preceq_{LT} sc_2 \end{array}$ *, if* $sc_1 \in LL$ *e* $sc_2 \in LT$

It is now possible to extend the concept of specificity to the conditional event expressions as follows:

Definition 23 *Let $ce_1 = \langle (sc_1, tc_1, ec_1), p : ev_1 \rangle$ and $ce_2 = \langle (sc_2, tc_2, ec_2), q : ev_2 \rangle$ be conditional event expressions for which $\exists (s,t,e)$ such that $(s,t,e) \models (sc_1, tc_1, ec_1)$ and $(s,t,e) \models (sc_2, tc_2, ec_2)$. Let $ep1 = Eprios(ec_1)$, $ep2 = Eprios(ec_2)$. We say that ce_1 is more specific than ce_2, denote by $ce_1 < ce_2$, if one of the following holds:*

1. *$p > q$. The priority of event expression ev_1 is greater than the priority of event expression ev_2;*
2. *$p = q \wedge ep1 > ep2$. The priority of the event ec_1 is greater than the priority of event ec_1;*
3. *$p = q \wedge ep1 = ep2 \wedge sc_1 <_{SCond} sc_2$. The events have the same priority but the spatial conditions sc_1 is more specific than sc_2.*

This criterion defines a partial order over conditional event expressions that might imply, depending on the context, the exclusion of some rules because their context is less specific than others. It is thus possible to define the set *MostSpecific* that, given a set S of rules, defines the subset of them that are most specific in S.

Definition 24 (MostSpecific) *Let S be a set of conditional event expressions. We define the set $MostSpecific(S) = \{c \in S \mid \neg \exists c' \in S : c' < c\}$.*

Let us see an immediate application of the ordering just defined.

Example 4 *Assume the space hierarchy consists of the logical locations $OperatingRoom1$ and $SurgeryDepartment$. Each location can be of one of the following types $OperatingRooms$, $Department$, where $OperatingRooms \preceq_{LT} Department$. More specifically, $OperatingRoom1$ is of type $OperatingRooms$ and $SurgeryDepartment$ is of type $Department$. The system can generate a localized event $SurgeryInProgress$ whose priority is greater than Min_P. Let us consider the following set of rules:*

1. *$\langle (WorkingHours, AnyPlace, AnyEvent)$, disable $Surgeon\langle OperatingRoom1 \rangle \rangle$*
2. *$\langle (WorkingHours, SurgeryDepartment, AnyEvent)$, enable $Doctor\langle SurgeryDepartment \rangle \rangle$*
3. *$\langle (WorkingHours, OperatingRoom1, AnyEvent)$, enable $Surgeon\langle OperatingRoom1 \rangle \rangle$*
4. *$\langle (\neg WorkingHours, OperatingRoom1, AnyEvent)$, disable $Surgeon\langle OperatingRoom1 \rangle \rangle$*
5. *$\langle (AnyTime, OperatingRoom1, SurgeryInProgress)$, enable $Surgeon\langle OperatingRoom1 \rangle \rangle$*

It is interesting to observe the effects of ordering over the above rules in two specific contexts. Let us consider the triple $\sigma = (s,t,e)$ where $t \in WorkingHours$ and $PtoL(s) = \{OperatingRoom1, SurgeryDepartment\}$, i.e., the user is physically inside the operating room 1 (that is reasonable to assume to be part of the surgery department).
In state σ there are no visible events, i.e, $e = \emptyset$. The set S of rules satisfied by the triple (s,t,e) is $A = \{1,2,3\}$, two of which, rules 1 and 3, are conflicting. However, since $OperatingRooms \preceq_{LT} Department \preceq_{LT} AnyPlace$, it follows that $OperatingRoom1$ is more specific than $SurgeryDepartment$ and thus

$$
\begin{aligned}
&MostSpecific(S) = \{ \\
&\quad \langle (WorkingHours, OperatingRoom1, AnyEvent), \text{enable } Surgeon\langle OperatingRoom1 \rangle \rangle \\
&\}
\end{aligned}
$$

Let us now consider the case in which $\sigma' = (s,t',e)$, where $t' \notin WorkingHours$ In this case it is immediate that $MostSpecific(S) = \{4\}$. Such a rule models the policy that does not allow a surgeon to enable the Surgeon *role in the OperatingRoom1 outside her working hours. On the other hand, let us assume that, under the same space and time conditions, the event $SurgeryInProgress$ becomes visible, i.e., $\sigma'' = (s,t',e')$ and $SurgeryInProgress \in e'$. This*

case models the typical medical emergency in which the surgeon, independently from the working hours, should be able to run the surgery. In this case, both the conflicting rules 4 and 5 in S meet the triple σ''. However, since $Eprios(SurgeryInProgress) >_P Eprios(AnyEvent) = Min_P$, it follows that MostSpecific(S) becomes:

$$MostSpecific(S) = \{$$
$$\langle (AnyTime, OperatingRoom1, SurgeryInProgress), enable\ Surgeon\langle OperatingRoom1\rangle$$
$$\}$$

■

6.2 System behavior: Traces

Now the overall behavior of the system can be formalized as follows: in every instant a subset of the conditional event expressions are satisfied by the current context. The most specific rules in this set identify the set of roles and run-time requests that generate the role enabling/disabling requests that the system will process to determine the set of roles that are enabled at the next time instant. Such behavior can be formalized by using the concept of *system trace* that describes the state of the system at each time point.

Definition 25 *A* system trace *is a pair (EV, ST) of infinite sequences of sets defined as follows. For every $t \geq 0$:*

- *$EV(t)$ is the set of prioritized event expressions received at time t.*
- *$ST(t)$ is the set of roles that are globally enabled at time t and the set of pairs (r, u) that represent exceptions, that is, $r \in R$ is disabled for the given user $u \in U$.*

The sequence ST evolves as a function of event expressions evaluated at time t as follows:

$$\begin{aligned} ST(t+1) = \Big(& ST(t) \cup \\ & \{(r,u) \mid \text{disable } r \text{ for } u \in Nonblocked(EV(t))\} \cup \\ & \{r \mid \text{enable } r \in Nonblocked(EV(t))\}\Big) \setminus \\ & \Big(\{(r,u) \mid \text{enable } r \text{ for } u \in Nonblocked(EV(t))\} \cup \\ & \{r \mid \text{disable } r \in Nonblocked(EV(t))\}\Big) \end{aligned}$$

The sequence EV should include, for every time t, the event expressions generated by the role enabling base and the run-time request received at t. In particular, for event expressions, it is necessary to consider the type of contextual condition. Indeed, some conditions express constraints that can be satisfied only by using the information carried by a specific set of users and, in this case, only such users should be affected by such a condition. Other conditions, instead, might affect all the user in the system. For example, a contextual condition based on a spatial requirement restricts the effect of the role enabling/disabling only to users that posses a specific information, e.g., their physical position meets the spatial requirement. On the other hand, a contextual condition consisting only of a temporal requirement affects all the users in that specific time interval, independently from the specific information every user carries. We can formalize such an observation as follows:

Definition 26 *Let (EV, ST) be a system trace, $\mathcal{R}$ be a role enabling base and RQ be a request stream. We define $Caused(t, EV, ST, \mathcal{R}, RQ)$ to be the set of prioritized event expressions generated by the elements that satisfy the following conditions:*

1. *If $(p : ev\ after\ \Delta t) \in RQ(t - \Delta t)$ and $\Delta t \leq t$,*
   then $p : ev \in Caused(t, EV, ST, \mathcal{R}, RQ)$

2. *If* $[E_1, \ldots, E_n, C_1, \ldots, C_k \rightarrow p : ev \text{ after } \Delta t] \in c$ *and the following hold:*

   - $\Delta t \leq t$
   - *for each state expression* C_i *in the form enabled* R, $R \in ST(t - \Delta t)$
   - *for each state expression* C_i *in the form* $\neg$*enabled* R, $R \notin ST(t - \Delta t)$
   - *for each event expression* E_i *there exists* $p : E_i \in Caused(t - \Delta t, EV, ST, \mathcal{R}, RQ)$ *not blocked by* $EV(t - \Delta t)$

   then $p : ev \in Caused(t, EV, ST, \mathcal{R}, RQ)$

3. *Let* $Q(t) = \{\langle (sc, tc, ec), p : ev\rangle \in \mathcal{R} \mid \exists e \in ActiveEvents(t) \wedge (t, EVisible(e), e) \models (sc, tc, ec)\}$. *For each conditional event expression* $ce = \langle (tc, sc, ec), p : ev\rangle \in MostSpecific(Q(t))$ *we have* $p : ev$ *for* $u \in Caused(t, EV, ST, \mathcal{R}, RQ)$ *when:*

   (a) Global, General Events: $(Eloc(ec) = \perp \wedge generatedFor(ec) = \perp)$ $\forall u \in \{v \in U \mid UserPos(v) \models sc\}$

   (b) Global, Personalized Events: $(Eloc(ec) = \perp \wedge generatedFor(ec) \neq \perp)$ $\forall u \in \{v \in generatedFor(ec) \mid UserPos(v) \models sc\}$.

   (c) Localized, General Events: $(Eloc(ec) \neq \perp \wedge generatedFor(ec) = \perp)$ $\forall u \in \{v \in U \mid UserPos(v) \models sc \wedge EVisible(ec) \cap PtoL(UserPos(v)) \cap \emptyset\}$.

   (d) Localized, Personalized Events: $(Eloc(ec) \neq \perp \wedge generatedFor(ec) \neq \perp)$ $\forall u \in \{v \in generatedFor(ec) \mid UserPos(v) \models sc \wedge EVisible(ec) \cap PtoL(UserPos(v)) \neq \emptyset\}$.

The rationale behind rules 3a to 3d is the following. The effect of a conditional event expression can either be bound by the features of the event *ec* or by the spatial constraints *sc* (or by both of them). Rule 3a is used to model the case in which the event in the conditional expression is neither a localized events nor a personalized one. In this case the prioritized event expression is global in the sense that it affects all the user that satisfy the space condition *sc*.
Rule 3b is used to model personalized non-localized events. In this case the constraints posed by the personalized nature of the event, allows the prioritized event expression to affect only the users to which the event is addressed to. However, the spatial condition might further reduce such set of affected users by restricting it to contain only the ones that satisfy *sc* at time t.

Rule 3c models localized and general events. In this case, the set of users that are affected by the prioritized event expression is the one that meets the spatial conditions. Finally, the rule 3d manages personalized and localized events. In this case, the set of affected users are the ones for which the event has been generated for, whose position falls within the visibility range of the event and that satisfy the spatial condition.

At this point it is possible to define the behavior of the system introducing the concept of *execution model*.

Definition 27 *A system trace* (EV, ST) *is said to be a* execution model *for a role enabling base* $\mathcal{R}$ *and a request stream* RQ *if, for each* $t \geq 0$*, it holds that:*

$$EV(t) = Caused(t, EV, ST, \mathcal{R}, RQ)$$

6.3 Extending the RBAC model

Although most of the logic that binds the context to the system state has been limited to the concept of role enabling and disabling, the presence of role templates, role instances and parametrized privileges requires that in order to obtain the desired behavior we need to

redefine the *RBAC* model as defined in Definition 1. In particular we need to (a) make the assignment relation a function of the new elements that have been introduced or restated in Section 5, and (b) redefine the relation within a session so as to keep the "transparency" goal we had. Indeed, if we want the system to automatically guarantee the "required" privileges as a function of the context, we need to redefine the concept of role activation (represented in RBAC by the function *roles*) so that active roles are those activated by the system using its rule base.

Definition 28 *The RBAC model is modified to include:*

- U, S*: sets of users and sessions as defined in RBAC*
- P*: set of parametrized privileges as defined in Definition 15*
- $R = R_T \cup R_I$*: set of roles composed by the union of the role templates* (R_T) *and role instances* (R_I)
- $PA \subseteq P \times R_T$*: relation associating permission to role templates*
- $UA \subseteq U \times R_I$*: relation associating role instances to users*
- $RH \subseteq R \times R$*, a partially ordered role hierarchy;*
- $user : S \rightarrow U$*. Function associating every session to the user that generated it (as in RBAC)*
- $eSR(s,t) = enabledSessionRoles(s,t)$*: function associating to each session the enabled roles at time* t *such that:*

$$eSR(s,t) = \{r \in R_I \mid (user(s), r) \in UA \wedge (r \in ST(t) \wedge (r, user(s)) \notin ST(t)\}$$

  where (EV, ST) *is the execution model (Definition 27) that describes the system.*

- $roles : S \rightarrow 2^R$*. Function associating to each session the active roles. Given a session* $s \in S$*, it holds that:*

$$\forall t \geq 0,\ r \in roles(s) \Leftrightarrow r \in eSR(s,t)$$

Given the model just defined, an access request should be allowed only if there exists a parametrized privilege such that (a) its conditional expression is satisfied, and (b) the privilege is associated with a currently enabled role for the user who made the request. More formally, let $ar = \langle s, op, o \rangle$ be an access request generated at time t, where s is a session, op is an operation and o is an object; the model allows the right to access the operation op over the object o if and only if the following holds:

$$(op, o) \in \bigcup_{y \in eSR(s,t)} \{(op, o) \mid \exists (op, oc, exp) \in PD(y) : oc \in ObjCat(o) \wedge exp\}$$

7 Using the ERBAC Policy Language

Conditional event expressions (Definition 18) allow the enabling/disabling of roles in response to environmental conditions. In particular, by using $STECond$ conditions, it is possible to refer to space, time and/or particular events that might happen in the system. Such conditions can thus be used to limit the release of privileges to users only in the case in which such privileges are actually needed for executing a specific task under some time-space constraints. Furthermore, events make it possible to specify the behavior that the system should have under exceptional conditions.

7.1 Relaxing time constraints in response of events.

Our approach allows the formulation of access policies that react to the exceptional events that may occur in the environment. Specifically, each reactive policy should consist of at least two *apparently conflicting* directives. A first one that models normal system behaviour and a second one, that is used to describe specific anomalous events. The latter overrides the former.

For example, assume that in normal conditions an employee carries out her duties under some space-temporal conditions, e.g., a doctor in a hospital has access to the medical records of the patients in her department while she is in her office during the office hours. Such *normal* behaviour has been already described in Examples 1, 2 and 3. Notice that, whenever the user is not in her department or she is there outside office hours, every access to patient records are implicitly forbidden.

On the other hand, such type of specification may be inadequate to address critical "unconventional" conditions. As a simple example, consider the case in which a doctor needs to access the medical records of a person having a heart attack while she is off-duty. In this case the event "heart attack" (assuming the patient is wearing a monitoring device that is able to recognize such a medical condition) can be used to relax the access policy to the patient medical record.

Example 5 *Let $Doctor\langle Department\rangle$ be the template defined as in Example 1 that defines the privileges of doctors in a generic department.*

In this case $Doctor\langle Cardiology\rangle$ is the specific role instance that limits the privileges of a doctor to the rooms that belong to the department of Cardiology. If we assume that the system is able to recognize a patient in critical conditions within a given department by generating an event $CriticalPatient \in E$, then the policy for normal and critical situations is:

1. *$\langle$(WorkingHours, Cardiology, AnyEvent) enable $Doctor\langle Cardiology\rangle\rangle$*
2. *$\langle$(AnyTime, Cardiology, CriticalPatient) enable $Doctor\langle Cardiology\rangle\rangle$*

The above rules enable, under normal conditions, permissions to doctors assigned to the cardiology department only during their WorkingHours *and within the logical locations belonging to the department of cardiology. These limitations are overridden under specific medical conditions, so that a doctor can operate while she is off-duty.*[2] ■

In this way we obtain a transparent management of permissions that, according to the minimum privilege principle, grants privileges only when they are strictly needed.

7.2 Localized Events.

Also event localization supports the definition of special behavior in a flexible and compact way. Specifically, in some cases, it is desirable to modify the access policy only in contexts that are *close* to the event location. For example, in case of a medical emergency, it makes sense to modify the access policy of the doctors that are in the department in which the event occurred. At the same time, a critical event in a department should not influence the privileges of personnel in other departments.

Different notions of *closeness* can be considered; all of them rely upon the location in which the event occurred. Starting from this information, and depending on the information that are available on the structure of the surrounding space, finer access policies can be designed.

[2] We will show a more complex example where space conditions are relaxed, too.

Example 6 *Consider the role template defined in Example 2 spanning over a number of departments. Let CriticalPatient be a localized event. We might add the following conditional event expression to the REB.*

$\langle (AnyTime, Department, CriticalPatient),$
$\quad enable\ Doctor\langle EVisible(CriticalPatient) \cap LofT(Department)\rangle\rangle$

If $EVisible(CriticalPatient) \neq \{\bot\}$*, i.e,. if the event* CriticalPatient *is localized, it is possible to obtain the department in which the event occurred and enable the role of doctors in the specific department.* ■

Notice that, the "level of visibility", i.e., the set of logical locations in which the event is visible, heavily depends on the information that the environment can generate. In the simplest case, the level of visibility, might be statically defined at the time of event definition. For example, if a specific device is physically located in a specific department, the administrator can associate a static list of logical locations in which the events generated by such device should be visible. Starting from this simple example, more complex situations can be easily imagined, e.g., mobile devices, multiple devices in different logical locations, and so forth. Clearly, complex situations require smarter/more complex definitions of the $EVisible$ function.

7.3 Global Events.

A global event is, by definition, an event that is not localized. Such definition can be interpreted in two different ways. The first interpretation defines an event to be global if no information is known about the location in which it has been generated—which may happen, say, if the device that generated the event does not support localization. The second interpretation defines as global the events that affect or might affect the whole space. In this paper we consider the second notion of globality. Let us now consider the following simple example of global event.

Example 7 *Consider the case in which the a hospital has to face a Limited Access code due to some kind of disaster. In this case, the access to the hospital premises to guests is forbidden. We might define the following roles:*

$Guest\langle\rangle = role($ $\quad (enter, Entrance, true)$ $\quad (exit, Exit, true)$ $)$	$LimitedGuestAccess\langle\rangle = role($ $\quad (exit, Exit, true)$ $)$

In this case, the REB might contain the following:

1. $\langle (VisitHours, AnyPlace, AnyEvent), enable\ Guests\langle\rangle\rangle$
2. $\langle (AnyTime, AnyPlace, LimitedAccess), disable\ Guest\langle\rangle\rangle$
3. $\langle (AnyTime, AnyPlace, LimitedAccess), enable\ LimitedGuestAccess\langle\rangle\rangle$

The first conditional event expression defines the normal operational behaviour that allows guest to enter and exit the hospital during visiting hours. Whenever a LimitedAccess *event is generated, independently from the current time, the Guest role is disabled, that is no more guest can enter the hospital. At the same time, the LimitedGuestAccess is enabled, allowing guests that might be in the hospital, to leave it.* ■

We describe another example in a completely different context. The spatial constraints seem to be meaningful only in case we have to deal with "physical" objects. As we show in the next example, this is not really the case. Furthermore, we show that conditional event expressions can be used to limit the visibility of global events, by bounding the set of locations that can be actually reached.

Example 8 *Consider a web site that offers services to registered users only. The site policy allows read-write access to registered users during normal operations. On the other hand, it has to periodically execute some Account Check operations during which write-access should be forbidden. In order to reduce system unavailability, the administrator is able to execute such maintenance operation on a regional basis so as to exploit night hours and office closure.*

This policy can be modelled using the event "AccountCheck" as a global-general event used to restrict the access to the users by defining the following roles:

$FullMember\langle\rangle = role($ $(read - write, Resource, true)$ $)$	$LimitedMember\langle\rangle = role($ $(read, Resource, true)$ $)$

Now, if AccountCheck $\in E$ is a global event visible to all the users, it is possible to implement the site policy repeating the following rules for each region in the world:

- $\langle$*(NightTimeEurope, UserFromEurope, AccountCheck), disable FullMember*$\langle\rangle\rangle$
- $\langle$*(NightTimeEurope, UserFromEurope, AccountCheck), enable LimitedMember*$\langle\rangle\rangle$

■

In the above example, the REB restricts the set of users to which the global event applies, based on the features of the user, i.e, she requires access from a place where it is currently night time. A different type of restriction might apply to features that are related to the resource being accessed. For example, we might partition the set of storage devices for a given service in different logical locations and limit the effect of global events by using such locations.

7.4 Personalized vs General Events.

In the above examples we have implicitily assumed that events are *general*, i.e., in principle, they are visible to all users. The actual set of users whose policy might be modified in response of a given event generation is the one whose members meet the spatio-temporal conditions defined in the REB.

In our model, events may also be linked *a priori* to the set of users that may be affected by the event's instantiations, i.e., events may be *personalised*. For example, we might modify Example 8, by defining *a priori* the set of users affected by the event AccountCheck. In other words, if AccountCheck is personalized, any instantiation of such an event will be visible only to the users defined by $generatedFor(AccountCheck)$ and, thus, these users will be the only ones whose privileges will be restricted. As usual, depending on the information that the system can describe, the generatedFor function might be as easy as a static list of users or a complex function as in the following.

Example 9 *Consider the case in which the generatedFor function can describe the list of "users requesting the service from a region during night time". The REB of Example 8 can be rephrased as follows:*

- $\langle$*(AnyTime, AnyPlace, AccountCheck), disable FullMember*$\langle\rangle\rangle$

- $\langle$*(AnyTime, AnyPlace, AccountCheck), enable LimitedMember*$\langle\rangle\rangle$

■

In the Example 9, the list of users that meet the spatio-temporal conditions defined by the policy in Example 8 is encapsulated in the definition of the generatedFor function. In general, this is not the case. Indeed, the set of users defined by the generatedFor function might be independent from the one defined by a spatio-temporal condition in the REB. Notice that it is possible to combine the effect of the event personalization and the spatio-temporal conditions in the REB in order to dynamically re-define the set of affected users as the intersection of the sets of users meeting both conditions.

Example 10 *Let us assume that the event LimitedAccessHelp is a personalized event generated for all the users having specific qualifications, say doctors, paramedics etc. We might add to the REB the following rule:*

- $\langle(AnyTime, ER, LimitedAccessHelp), enable\ HelpGuest\langle\rangle\rangle$

In this case, the system should be able to generate two different events. The first one, LimitedAccess, described in Example 7, limits the set of users that can access the hospital. The second one, LimitedAccessHelp, should be generated in case the personnel realizes that they need "help" for managing the current situation. Such second event, would allow someone who is not employed by the hospital, but that has the proper qualifications, to temporarily gain the role of HelpGuest in order to fulfill the request. Such role, however, will be enabled only for the users that are either in or entering the ER department. ■

7.5 User-generated events.

Until now we have described ways in which system generated events have been used to guide the behavior of users whenever some context-dependent condition is raised by the environment. On the other hand, if the environment is able to identify/manage events generated by users, such capability can be useful to transparently manage all the situations that are direct consequence of the actions of a specific user/set of users.

Example 11 *As a concrete example, consider disk quota management. It might be useful to define a policy stating that write privileges are revoked from users that exceed their quota. This policy is formalized by the following template:*

$User\langle UserName\rangle = role($ $(rw, Directory, DirectoryName = UserName)$ $(rw, Files, true)$ $)$	$LimitedUser\langle UserName\rangle = role($ $(r, Directory, DirectoryName = UserName)$ $(r, Files, true)$ $)$

Let $QuotaExceeded \in E$ be the user generated event corresponding to quota exceeded[3]*. We can use the following rules.*

- $\langle(AnyTime, AnyPlace, QuotaExceeded),$
$disable\ User\langle generatedBy(QuotaExceeded)\rangle\rangle$

- $\langle(AnyTime, AnyPlace, QuotaExceeded),$
$enable\ LimitedUser\langle generatedBy(QuotaExceeded)\rangle\rangle$

[3]Technically, disk quota is an event generated by the system in response of the action of a user. However, for the sake of our presentation, this distiction is immaterial.

Let joe *be a user and let* $User\langle joe\rangle$ *and* $LimitedUser\langle joe\rangle$ *be the instances of the above templates that might be associated with such a user. The above rules encode the policy that forbid write access to joe whenever he exceeds his quota.* ■

We stress that, although the event is generated by a specific user, its effects might also influence the behaviour of other users. In the previous example the site policy might, in response to a QuotaExceeded event, forbid write access to every user on the specific device.

7.6 Unrelated conditions for role enabling/disabling

Clearly the conditions under which a role can be enabled or disabled may be completely unrelated. Such property is particularly useful whenever a process can be started under particular conditions but its execution does not require them to persist. Consider the following example. It is reasonable to assume that surgery can start only when all the specialized personnel is present. On the other hand, because of the criticality of such a process, it should not be possible to stop the process if, for some reason, some of the required conditions fails, (e.g., the principal surgeon faints). We can model such conditions by defining two events. The first one, SurgeryInProgress is activated by the system as an effect of the actual start of the surgery. Clearly such an event is activated only after all the required conditions are met, i.e., all the required personnel entered the specific operating room, the supervisor is identified, etc. Notice that, SurgeryInProgress is bound to the execution of a process while it is not bound to the presence of a person with a specific role (the surgeon). A second event *NoResponsible* is defined as the situation in which no person that is able to assume responsibility is present in the operating room, e.g., the case in which no doctor is present in the room. We can use such events as follows:

1. $\langle(AnyTime, OperatingRoom1, SurgeryInProgress)$,
   enable $Assistant\langle OperatingRoom1\rangle\rangle$
2. $\langle(AnyTime, OperatingRoom1, NoResponsible)$,
   disable $Assistant\langle OperatingRoom1\rangle\rangle$

Such a definition enables the role Assistant during the surgery and does not disable it in the unlucky case in which the principal surgeon faints, but other doctors are present in the room.

Notice that cascade role disabling could be alternatively implemented by means of the following rules.

1. enable $Surgeon\langle OperatingRoom1\rangle \rightarrow$
   enable $Assistant\langle OperatingRoom1\rangle$
2. disable $Surgeon\langle OperatingRoom1\rangle \rightarrow$
   disable $Assistant\langle OperatingRoom1\rangle$

However, with this formulation, the enabling/disabling of role Assistant depends only on the state transitions of role Surgeon and does not take the context into account. Such modeling would, thus, be too rigid and would not faithfully express the emergency handling policy.

7.7 Conditional event expression ordering

Priorities (and the corresponding ordering of conditional event expressions) support a *layered* and incremental formulation of policies, by means of general rules and suitable exceptions to those rules.

Example 12 *Event priority can be exploited to simplify and merge the access policies presented in Examples 7 and 10 as follows. Assume that the system is only able to generate a global/general* LimitedAccess *event. The REB can be written as follows:*

1. $\langle (VisitHours, AnyPlace, AnyEvent),$ *x: enable* $Guests\langle\rangle\rangle$
2. $\langle (AnyTime, AnyPlace, LimitedAccess),$ *x: disable* $Guest\langle\rangle\rangle$
3. $\langle (AnyTime, AnyPlace, LimitedAccess),$ *x: enable* $LimitedGuestAccess\langle\rangle\rangle$
4. $\langle (AnyTime, AnyPlace, LimitedAccess),$ *x+1: enable* $HelpGuest\langle\rangle\rangle$

Whenever a LimitedAccess event is generated, if a guest is not qualified to provide help, e.g., she did not provide any doctor *or* nurse *credential, then the Guest role is disabled and the LimitedGuestAccess is enabled. allowing her to leave the hospital. On the other hand, if a guest has provided some credential that qualify her as a member of HelpGuest, two enabling rules apply, namely rule* 3 *and* 4*. However, since the priority associated to rule* 4 *is higher than the one in rule* 3*, only the former will be applied, by allowing the user to gain the privileges of HelpGuest role.* ■

By using conditional event ordering, we can define *tight policies*. Consider, for example, the case of porters in hospitals. A porter has the role of moving patients and materials within an hospital. We can thus assume that, potentially, she has access to each area in the hospital. However, it is reasonable to limit the access of porters only to *specific areas* for the time needed to complete the transfer and only when a specific transfer has been requested.

Let us assume a spatial organization described by the following hierarchy over location types:

$$OpearatingRoom \preceq_{LT} Department \preceq_{LT} Pavilion \preceq_{LT} Hospital$$

For efficiency reasons, we might think of restricting the area of a specific porter to a specific pavilion by using a role template as follows:

$$\begin{array}{l} Porter\langle pavilion\rangle = role(\\ \quad \ldots \\ \quad (access, Department, DepartmentIn(pavilion), \\ \quad (access, OperatingRoom, RoomIn(DepartmentIn(pavilion)), \\ \quad \ldots \\ ) \end{array}$$

where $DepartmentIn$ (resp, $RoomIn$) is a predicate that evaluates whether or not a specific location belongs to a specific Department (resp., to the specific operating room) in the specified pavilion.

If we do assume the existence of an event $TransferRequired \in E$ with $Eloc(TransferRequired) \in PL$, we might define the following:

$$\begin{array}{l} 1.\langle (AnyTime, AnyPlace, AnyEvent), \text{disable } Porter\langle lloc \in LofT(Pavilion)\rangle\rangle \\ 2.\langle (AnyTime, AnyPlace, TransferRequired), \text{enable } Porter\langle \\ \quad lloc \in EVisible(TransferRequired) \mid pavilion \in LtoT(lloc)\rangle\rangle \end{array}$$

The effect of the first rule is to disable the instances of the template *Porter* bound to every pavilion in the hospital while the second one we enable the role instance related to the pavilion in which a transfer has been required.

Notice that, the two rules differ only in the event part and, thus, the second rule is more specific than the first one since $Eprios(AnyEvent) = Min_P$. We stress that the above policy does not work if no ordering is specified. Indeed, the above rules are conflicting and, according to the *denials take precedence* conflict resolution strategy, the Porter role would

never be enabled. We recall that the above discussion holds also if there exist multiple active events at the same time. In that case the system shall give precedence to the set of rules corresponding to the most critical event, i.e., the one with highest priority.

Another interesting case is the one related to the ordering of spatial conditions. In this case the system will tend to privilege rules that are referring to *more specific* spaces. In this way it is possible to specify policies for different logical layers corresponding to the different spatial organizations while being guaranteed, at the same time, that the system will always enforce the more specific one, i.e, the one that best fits the current user location.

An example of the above discussion can be done in the context of hospital-related policies, for the definition of the access policy to the operating room. Typically only a subset of the personnel working in a given Department has the qualifications to work in an operating room. For this reasons, the operating room represent an example of a (more specific) logical location that requires an access policy that is different from the more generic logical location to which it belongs to (e.g., Department, Pavilion, etc).

For the sake of simplicity, let us assume that the following roles are the ones that can work in the operating room: surgeon, anesthetist and operating room nurse. The first two roles are assigned to doctors with a given specialization while the third one corresponds to a specialization for the nursing staff. This means that in the operating room a doctor or a nurse should loose their "generic" privileges and, if their role is enabled, acquire the new and more specific one related to the work they are supposed to do. Now, assume that $Doctor\langle Department\rangle$ and $Nurse\langle Department\rangle$ are the role templates that define the permissions for doctors and nurses within a given department and $Surgeon\langle Department\rangle$, $OperartingRoomNurse\langle Department\rangle$ e $Anesthetist\langle Department\rangle$ are the roles associated with the permissions needed to work in the operating room. Let the above roles be defined as follows:

$$
\begin{aligned}
&Doctor\langle Department\rangle = role(pd_1(Department), \ldots, pd_l(Department))\\
&Nurse\langle Department\rangle = role(pn_1(Department), \ldots, pn_m(Department))\\
&\ldots\\
&Surgeon\langle Department\rangle = role(\\
&\quad ps_1(Department), \ldots, ps_n(Department), Doctor\langle Department\rangle\\
&\quad)\\
&Anesthetist\langle Department\rangle = role(\\
&\quad pa_1(Department), \ldots, ps_p(Department), Doctor\langle Department\rangle\\
&\quad)\\
&OperartingRoomNurse\langle Reparto\rangle = role(\\
&\quad po_1(Department), \ldots, po_q(Department), Nurse\langle Department\rangle\\
&\quad)
\end{aligned}
$$

If we assume the following definitions for the space and time representations:

- $Department, OperatingRoom \in LT$ such that $OperatingRoom \preceq_{LT} Department$
- $Surgery \in LL$ such that $Surgery \in LofT(Department)$
- $OperatingRoom1 \in LL$ such that $OperatingRoom1 \in LofT(OperatingRoom)$
- $WorkingHour$: temporal expression identifying the staff working hours.

Then privileges can be assigned with the following policy:

1.$\langle(WorkingHour, Surgery, AnyEvent)$, enable $Doctor\langle Surgery\rangle\rangle$
2.$\langle(WorkingHour, Surgery, AnyEvent)$, disable $Surgeon\langle Surgery\rangle\rangle$
3.$\langle(WorkingHour, Surgery, AnyEvent)$, disable $Anesthetist\langle Surgery\rangle\rangle$
4.$\langle(WorkingHour, Surgery, AnyEvent)$, enable $Nurse\langle Surgery\rangle\rangle$
5.$\langle(WorkingHour, Surgery, AnyEvent)$, disable $OperartingRoomNurse\langle Surgery\rangle\rangle$
...
6.$\langle(WorkingHour, OperatingRoom1, AnyEvent)$, disable $Doctor\langle Surgery\rangle\rangle$
7.$\langle(WorkingHour, OperatingRoom1, AnyEvent)$, enable $Surgeon\langle Surgery\rangle\rangle$
8.$\langle(WorkingHour, OperatingRoom1, AnyEvent)$, enable $Anesthetist\langle Surgery\rangle\rangle$
9.$\langle(WorkingHour, OperatingRoom1, AnyEvent)$, disable $Nurse\langle Surgery\rangle\rangle$
10.$\langle(WorkingHour, OperatingRoom1, AnyEvent)$, enable $OperartingRoomNurse\langle Surgery\rangle\rangle$

In the above definition, rules $1-3$ guarantee that every doctor within his own department has no more than all the necessary permissions needed to work in the department. The same policy is implemented for nursing staff by rules $4-5$.
Rules $6-10$ model the behavior of the system whenever the physical location of the user is within a more specific logical location, the OperatingRoom1. In this case, the system revokes the privileges that doctors and nurses had in the department (rules 6 and 9) and enables the more specific roles needed to work in the operating room (rules 7, 8 and 10).

The above strategy for policy definition has the following side effect. Although we adopt the "denials take precedence" conflict resolution policy, we can guarantee the correct enabling of roles because they are defined in "more specific" logical locations (OperatingRoom1) as opposed to the "more generic" Surgery. A second side effect is the disabling of roles for unqualified personnel entering the operating room. Indeed, if a nurse enters the operating room, her privileges are immediately revoked by rule 9. On the other hand, since she is not qualified to enable the role OpeartingRoomNurse, such a role will not be enabled with the effect that the unqualified personnel will be in the operating room without any privilege.

7.8 Proximity-based visibility

In the previous sections we have considered the case in which event visibility is only related to the space hierarchy. In other words, whenever an event is defined, the administrator also defines the set of locations in which such an event is visible. There are scenarios, however, in which some event should reach a subset of users that are somehow "close" to the location where the event has been generated. In this paper we will only present a case in which, informally, two users are close if they are in the same logical location. We stress, however, that other notions of "closeness" might be defined as well.

We present a concrete application scenario. We consider the case in which patients carry mobile monitoring devices that are able to (a) store the medical records of the patient and (b) generate alerts in case some medical emergencies, e.g., heart attack. Clearly the protection of the patient's medical record is a sensitive task that should be specified by a proper access policy. For example, we could assume that such information can be accessible only by the doctor who is responsible of the patient treatment and within some medical premises, e.g., doctor's office or a department within a hospital. However, it is reasonable to assume that every patient would be willing to disclose her medical records *to every doctor in case of real medical emergency*. In other words, in case the patient is having a heart attack, independently from the place the patient is, release her medical records to whoever qualifies as a doctor that is close enough to save her.

In order to let *whoever qualifies as a doctor* to access such information, we need to provide such users with a minimal set of privileges that apply outside the logical location in which they *usually* operate. In other words, a user that belongs to the medical staff in some

hospital will also be provided with a set of minimal privileges that can be used *outside* her home institution. This scenario assumes a federation of institutions that adopt a shared, global access policy such that a user identified as "surgeon" or "anesthetist" in the medical institution A is identified as $Doctor\langle\rangle$ in all the other institutions within the federation. Given such a role, the other roles can be defined as follows:

$$
\begin{array}{l}
Surgeon\langle Hospital, Department\rangle = role(\\
\quad role_1\langle Hospital, Department\rangle, \ldots, role_n\langle Hospital, Department\rangle, \\
\quad Doctor\langle\rangle, \\
\quad pp_1(Hospital, Department), \ldots, pp_m(Hospital, Department) \\
)
\end{array}
$$

$$
\begin{array}{l}
Anesthetist\langle Hospital, Department\rangle = role(\\
\quad role_1\langle Hospital, Department\rangle, \ldots, role_p\langle Hospital, Department\rangle, \\
\quad Doctor\langle\rangle, \\
\quad pp_1(Hospital, Department), \ldots, pp_q(Hospital, Department) \\
)
\end{array}
$$

Among the permissions associated with the role $Doctor\langle\rangle$ we should include one for accessing the medical records of a patient in case of medical emergency. Notice, however, that such a simple solution relaxes privacy constraints too much. Indeed, we need a way to specify that, in case of emergency, the patient's medical records should be released only to a doctor that is *close-by*. In other words, we need a way to *relax* the access policy while *binding* its applicability to some concept of physical closeness. In this way we do not release any sensitive information to personnel that cannot use it to help the patient while, at the same time, we guarantee the maximum possible disclosure of the patient records to all the users that can save her life.

Let us assume $MedicalDevices \in OC$ be the category of objects containing the patients monitoring devices. Furthermore, let $MedicalEmergency \in E$ be the localized event generated a medical emergency. Let $p, q \in PL$. We define the predicate $AreClose(p, q)$ to be true whenever p and q are close according to Definition 8. The template $Doctor\langle\rangle$ can be defined as follows:

$$
\begin{array}{l}
Doctor\langle ev\rangle = role(\\
\quad \ldots \\
\quad (read, MedicalDevice, \\
\qquad ev = \text{“}MedicalEmergency\text{”} \wedge \\
\qquad MedicalDevice.Owner = generatedBy(ev) \wedge \\
\qquad AreClose(Eloc(ec), CurrentPosition)\}) \\
\quad \ldots \\
)
\end{array}
$$

The conditional expression guarantees that the doctor has access to medical data only if the device has generated the event associated with a medical emergency and only for the user(s) associated to the event.

The policy that every institution should implement, must ensure the preservation of such a minimal role within the federation. We notice that, because of the ordering over rules in our model, the following set of rules yields the desired behavior of (a) assigning the proper role within the home institution and (b) assign the generic Doctor role outside.

$\langle(AnyTime, AnyPlace, MedicalEmergency), \text{enable } Doctor\langle MedicalEmergency\rangle\rangle$
...
$\langle(AnyTime, AnyPlace, AnyEvent), \text{disable } Anesthetist\langle Hospital, Department\rangle\rangle$
$\langle(AnyTime, AnyPlace, AnyEvent), \text{disable } Surgeon\langle Hospital, Department\rangle\rangle$
...
$\langle(AnyTime, Hospital, AnyEvent), \text{enable } Anesthetist\langle Hospital, Department\rangle\rangle$
$\langle(AnyTime, Hospital, AnyEvent), \text{enable } Surgeon\langle Hospital, Department\rangle\rangle$
...

Clearly the above idea can be applied in different application domains. For example, we might immediately grant access to sensitive information or communication means to off-duty policemen in case of terrorist attack. Similarly, in case of fire we might grant access to every user to maps augmented with the (sensitive) sensor information for guaranteeing secure escape paths.

7.9 ERBAC for usage control: Relationship with UCON

In [31] the authors introduces a meta-model that is able to describe different usage control models. Basic components in this meta-model are the following. The set of S of subjects consists of all the actors that can execute actions. Each action can be executed on one object belonging to the set O of objects. The set R of rights contains the privileges that allow a subject to execute an action on an object. The set A of authorization contains a number of predicates that are evaluated in order to decide whether or not a specific subject has the right to execute a given operation on a specific object. The set B of obligations consists of all predicates that verify the requirements a subject has to fulfill before or during an operation. Finally the set C of conditions is used to describe environmental or system-related factors. Both subjects and objects are characterized by attributes that can be used during the evaluation of predicates. Two basic ingredients in the UCON meta-model are *attribute modifiability* and *continuity of decision*. The former is used to describe the capability of an authorisation system to deal with the dynamicity of subjects' and/or objects' attributes. The latter describes the capability of a model to keep monitoring the 'granting conditions' also after the actual permission has been granted. Each user request is always evaluated *before* the required permission is granted. So a systems might have *pre-Authorization/oBligation/Condition*, denoted by *preA, preB* or *preC*, respectively. Systems that keep monitoring the authorisation process might have *ongoing-Authorization/oBligation/Condition*, denoted by *onA, onB* or *onC*, respectively. For example, continuous position monitoring in onA systems might be used to model mobility. The UCON meta-model is *transaction-based*. This means that each predicate is evaluated each time there is a user-request and the outcome of such predicates influences the authorisation process.

The ERBAC model presented in this paper is able to manage *role enabling/disabling transactions*. Since our model is an extension of the RBAC model, there is a one-to-one mapping between the sets S, R, O in the UCON and the corresponding sets in ERBAC. in both frameworks subjects and objects are associated to attributes. ERBAC does not support attribute modification. The set C of conditions correspond in ERBAC to the set of environmentally generated information, that is, STECond. ERBAC does not support obligations at all. Since in the ERBAC model role enabling (resp. disabling) always implies role activation (resp., deactivation), our model is implicitly designed to provide continuity of decision. It is possible to classify ERBAC as an instance of the $\text{UCON}_{\text{onA,onC}}$. ERBAC is an onA model since temporal and spatial conditions are continuously monitored in order to identify the "most appropriate" rule to be executed, c.f., Examples 4 and 5. Finally ERBAC can be classified as an onC model to the extent that "events" are used to trigger enabling/disabling of roles, as in Examples 7 and 12.

8 Prototype Implementation

We have implemented a prototype system that allows the specification and enforcement of ERBAC policies. It is available at [9]. Our prototype implementation is based on the XACML standard [25] for policy specification. We extended such a standard for allowing the specification of policies based on events. We have evaluated three different implementation of XACML, Sun's XACML [25], XACML Enterprise [1] and XACML Light [2]. Among the available implementations, we have extended the open source java SUN's implementation XACML [25] since it has a modular and easily extendible architecture. Furthermore, as noted in [43], Sun's implementation, has a good trade-off between time needed to load policies and time needed to compute access decisions.

8.1 The XACML Standard

OASIS Standard [25] introduces both a standard for policy specification and the description of an architecture that could be implemented to enforce such policies. Such an architecture consists of a number of elements, each devoted to a specific task. A central role is played by the Policy Decision Point (PDP) and the Policy Enforcement Point (PEP). The former receives access requests from the latter, decides whether or not the access should be granted and sends back the decision. The PEP is responsible for policy enforcement. It receives access requests from the users, forwards them to the PDP and implements its decisions. PDP requests and responses are specified using a canonical format, called the *XACML Context*. A specific component, the *Context Handler* is used to translate XACML contexts to/from the specific format supported by the PEP. Other elements in the architecture are the Policy Administration Point (PAP) that provides policies to the PDP, and the Policy Information Point (PIP) that is responsible for the evaluation of attributes, e.g., gathering state information or information from the environment.

Every access policy can be seen, informally, as a set of *Rules*. A *Rule* is the basic element in the language and it is defined by the triple (*Target, Effect, Condition*), where the Target identifies the circumstances to which the rule applies, the Effect specifies the outcome of the decision process and the Condition is a boolean expression over the set of attributes that can be used to further restrict the applicability of the rule. The *Target* consist of the 4-tuple (*Subjects, Resources, Actions, Environment*) that can be interpreted as the set of circumstances in which *Subjects required the execution of some Actions on the Resources in a given Environment*, where the latter element is represented by the set of attributes that are relevant for the decision process. The XACML defines four different values for *Effect*: Permit, Deny, NotApplicable, Indeterminate. The first two are used to permit or deny the required action. The effect of a rule is NotApplicable whenever the current request does not match the Target or the Condition. Finally the effect is Indeterminate whenever the rule evaluation requires the values of some attribute that is not currently available to the PDP.

XACML defines *Policies* as the set of rules that can be applied to a same Target, along with a specification of the algorithm for composing the effect of different rules when more than one is applicable (e.g., permit overrides deny). Furthermore, a policy might include a set of obligations, i.e., operations that should be enforced by the PEP in conjunction with the authorisation decision. A Policy set is, informally, a set of policies. In [24] the OASIS consortium provides a profile for specifying RBAC policies using XACML. Roles and permissions specifications are obtained by two distinct Policy Sets. The Permission Policy Set (or PPS) is used to define a set of permission. Recall that a permission can be seen as a pair (operation, object). A PPS contains Policies and Rules in which the Target defines the Resources (objects) and the Actions (operations), and possibly other conditions that are used to restrict access. Such policies might also include references to other PPS associated with junior roles (allowing the inheritance of permissions). A RPS is a policy set associating the holder of a given role to the RPS containing the associated permissions. The Target of an

RPS limits the set of subjects to which the role can be assigned. Finally a RPS can reference only a single PPS. A Role Assignment policy or policy set defines the association of users to roles. Notice that each PPS or RPS is a static object uniquely identified by a Uniform Resource Identifier (URI).

8.2 XACML Extension

The XACML standard [25] defines a number of basic data types along with a number of functions that can be used to process such data. Clearly, in order to allow the formulation of policies based on STEConditions we needed to extend XACML. Such an extension consists both of new data types and functions that are needed to process the new types.

A contextual condition is defined by means of a triple containing a spatial condition, a temporal condition and an event condition.
For spatial conditions we need to be able to represent physical locations, logical locations and location types. Logical locations and location types can be represented as strings. A more complex representation is required for physical locations. In this case we need a way of representing coordinates and, more generally, geometrical shapes in two- and three-dimensional spaces. For these reasons we decided to used GeoXACML [27], an extension of the XACML standard by the OpenGIS Consortium [29, 26, 28]. GeoXACML can describe arbitrary geometric shapes starting from the following base types [29]: Point, LineString, Polygon, MultiPoint (a set of points), MultiLine(a set of LineString) and MultiPolygon (a collection of Polygon). The XACML schema is extended to allow the possibility of describing such new type of geometries by means of an extension of XML, known as Geography Markup Language (GML) [26].
As for temporal conditions we needed to introduce a new data type in order to represent periodic expressions. The pair (I, P) is represented as follows: The interval I is defined as the pair of its endpoints. Furthermore, we have defined the basic calendars Hours, Days, Weeks, Months and Years used to specify periodic expressions. Finally event are represented by means of strings. For each data type we defined the function needed to operate on it.

The representation of role templates in XACML using RPS and PPS is not immediate. The basic reason is that a role template is essentially a parametric role and, thus, it is not sufficient to define a new static object to represent it. Furthermore, a role instance is defined by assigning specific values to each role parameter. for this reason it is, in principle, impossible to define all possible role instance *a priori*. A second problem arises from the fact that the variables that are used to define parametrized privileges must be a subset of the variables of the role template that contains it. For this reason we extended the XACML grammar in order to identify the variables in the role instance definition in order to use them in the instantiation process.

The following tables report the extension to XACML syntax adopted in our implementation. In Table 2 we report the syntax for the specification of role templates and periodic expressions. In Table 3 we report the functions we have implemented for managing the new conditions and data types.

Our prototype implementation extends SUN's XACML [42]. The main upgrades concern the PDP, i.e., the module that is responsible for policy decision. We have focussed our work on the issues related to context-dependent decisions, leaving out context-indepedent matters, e.g., triggers.

Space representation has been implemented by using the JTS [40] library. The implemented prototype supports the point and polygon types, that are necessary to define the physical location of an object and logical locations. Location types are represented by means

Table 2: XACML Data Type Extension

What	Syntax	Parameters

Role Templates	`<Attribute>`
	`  <AttributeValue DataType="urn:my:dataType:role">`
	`  <RoleName>string</RoleName>`
	`  <RoleParams [number=int]>`
	`  [<Param Name=String DataType=type>value</Param>]`
	`  </RoleParams>`
	`  </AttributeValue>`
	`</Attribute>`
		number: number of variables in the template. **value**: omitted in case of templates.

Periodic Expressions	`<AttibuteValue DataType="urn:my:periodic-expression"`
	`  [TimeZone=String]>`
	`  [<Interval [DateFormat=String]>`
	`  <Start></Start>`
	`  <End></End>`
	`  </Interval>]`
	`  [<PeriodicExp>`
	`  <CalendarStart Type=String>s1-e1,s2-e2,...`
	`  </CalendarStart>`
	`  [<CalendarStart Type=String>...</CalendarStart>`
	`  ...]`
	`  <CalendarLength Type=String>int</CalendarLength>`
	`  </PeriodicExp>]`
	`</AttributeValue>`
		si-ei: Index interval **Calendar Type:** in {Years, Months, Weeks, Days, Hours}

of directed graph in which an edge between types A and B indicates that B is more specific than A. A *Location Handler* has been implemented in order to support the PDP in all decisions related to spatial conditions.

In order to make event management more flexible, we defined an interface that every event must implement. In other words, each event must be associated by the administrator to a name and a priority and, optionally, to the user who generated it, the user to whom the event is addressed to and the location in which the event occurred. Given such information the *Event Handler* will be able to support the PDP in the decision process.

8.3 Complexity considerations

Since our prototype is based on Sun's XACML implementation, all the considerations reported in [43] still hold. In particular, running time is proportional to the size of the implemented policies measured as the total number of PolicySets and rules. What we are most interested in evaluating is the impact on running time of the new elements introduced in this paper. In particular, whenever more than one rule applies to the received access request, the PDP should compute the *most specific* context and apply the corresponding rule. in order to find the most specific rule the PDP needs to sort them according to (a) the event expression priority, (b) the event priority, and (c) spatial specificity, in this order. The first two are immediate, the third test requires a DFS over the graph representing location types. The probability that an access request involves a large number of rules based on the spatial condition is low. Indeed, the Target in the XACML rule is used to filter the rules in the policy. It is reasonable to assume that, for well designed policies, the average number of rules matching a given access request is low.

Another operation that could heavily affect the running time is the need to retrieve all

Table 3: XACML Function Extension

URI	Parameters	Returned Value
urn:my:function:spatial:point-within-location-logical	LogicalL: stringAttribute Point: GeometryPointAttribute	$Point \in LogicalL$
urn:my:function:spatial:point-within-location-type	LType: stringAttribute Point: GeometryPointAttribute	$Point \in LofT(LType)$
urn:my:function:role:is-instance	Template: RoleAttribute Instance: RoleAttribute	Instance is a proper instance of Template
urn:my:function:role:same-instance	Instance1: RoleAttribute Instance2: RoleAttribute	Both parameters are instances of the same template.
urn:my:function:time:inside urn:my:function:time:not-inside	Date: DateTimeAttribute Pexp: PeriodicExpressionAttribute	$Date \in Sol(Pexp)$ $Date \notin Sol(Pexp)$
urn:my:function:event:visible-at	EvName: StringAttribute Point: GeometryAttribute UserName:StringAttribute	EvName is visible by UserName in Point
urn:my:function:event:visible-in	EvName: StringAttribute LogicalLoc: StringAttribute UserName:StringAttribute	EvName is visible by UserName in LogicalLoc
urn:my:function:event:generated-by-and-visible-at	GenUser: StringAttribute EvName: StringAttribute Point: GeometryAttribute UserName:StringAttribute	EvName has been generated by GenUser and is visible by UserName in Point
my:rule-combining-algorithm:deny-overrides		Rule combining algorithm evaluating the MostSpecific rule.

logical locations that a given physical location belongs to (e.g., the user location). Notice that this operation is required in each STE-cond expression evaluation; it clearly depends on the number of logical locations and, thus, on the complexity of the spatial hierarchy the system has to manage. Furthermore, the higher the complexity of the polygon (i.e., the shape) that describes a logical location, the higher the actual time needed to check containment. A naive way of performing such an operation is checking whether a given point belongs to each logical location in the hierarchy. Such a strategy takes linear time in the (total) number of logical locations. In order to reduce the computational burden for this operation, we implemented the following strategy. Each logical location is identified by a set of polygons. For each logical location we consider the smallest rectangle (aligned to two given axis) that includes all its polygons and we identify the two intervals on the x and y axis corresponding to the rectangle sides. We construct two interval trees, an X-tree and a Y-tree, each including the coordinates of all the identified intervals on the corresponding axis. Whenever we need to recover the logical locations including a given physical location (x, y), we extract from the X-tree all rectangles having a side intersecting x and from the Y-tree all rectangles having a side intersecting y. Next, we compute the intersection of the two list of rectangles obtained so far, since every logical location containing (x, y) must belong to both list. We finally check all the polygons contained in the identified rectangles. In this way, by using the proper data structures [12], if the number of intervals containing x in the X-tree and y in the Y-tree are both $O(k)$, our strategy takes $O(k + \log n)$ time instead of $O(n)$. Clearly, we need $O(n \log n)$ time to build the trees. However this operation occurs only at start-up.

8.4 Performance Evaluation

In real-life scenarios the system's response time clearly depends on a number of factors ranging from the access policy size to the efficiency of the devices that feed data to the PDP. In this paper we evaluate the overhead caused by policy rule evaluation alone (contingent aspects such as device response time and transmission overhead will not be considered).

A viable, efficient implementation of periodic rule triggering has already been discussed in the context of TRBAC [8]. Since the set of all temporal expressions in a REB has a periodic behavior, the approach adopted in [8] consists in materializing, at REB load time, an agenda that decribes rule activation times for a single period (in terms of offsets from the period's beginning). The agenda is used to program a cron daemon (at the beginning of each period) that triggers role enabling/disabling rules.

The same idea applies to our framework and suggest a simplification of context expressions that can be regarded as a pre-compilation of temporal conditions. Each context expression $\langle TE_i, S, AnyEvent\rangle$ can be transformed into $\langle AnyTime, S, E(TE_i)\rangle$, where $E(TE_i)$ is the event generated by the cron daemon (using the agenda) when TE_i is satisfied by the current time.

Note that in our reference examples most context expressions are shaped like either $\langle TE_i, S, AnyEvent\rangle$ or $\langle AnyTime, S, E\rangle$, that is, either the temporal expression or the event expression is trivial[4], So the above pre-compilation reduces most context expressions to the $\langle AnyTime, R, E\rangle$ format.

For this reason, our experiments deal with context expressions of this kind and focus on the overhead caused by the space expression S, that is, by event propagation across physical and logical locations, and by event overriding based on location specificity.

As stated before, our prototype at start-up loads and parses the files describing the access policy and the spatial hierarchy, building up their internal representation that will be kept in memory for evaluating access requests. This phase may be relatively expensive, but it takes place only once and thus we have not considered it in our scalability analysis.

[4]The unique exception where both the temporal expression and the event expression are nontrivial can be found in Example 8. Interestingly, the same policy is reformulated without time constraints, cf. Example 9.

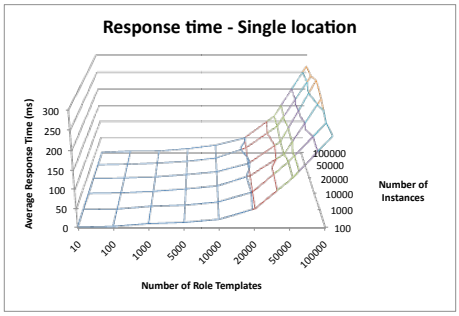

Figure 1: Evaluation time as function of number of policies and instances.

The following experiments have been carried out on a Intel Core i7 processor at 1.7 GHz, equipped with 8 Gb of RAM and 512 Gb SSD, running Mac OSX v. 10.9.1.

The first experiment we run is used to establish a baseline against which we measure the impact of the ERBAC model over the existing implementation. We have considered policies defined over a single logical location, with no active events and we have measured the average evaluation time, that is, we have created 100 access requests and computed the average amount of time needed by the systems to answer each request. The number x of role templates used in our experiments belongs to the set $\{10, 100, 1000, 5000, 10000, 20000, 50000, 100000\}$. Since templates do not define an actual policy until they are instantiated, we run several experiments by instantiating, for each value of x in the above set, a number y of role instances in the following set: $\{100, 1000, 10000, 20000, 50000, 100000\}$. Role templates are instantiated in a round-robin fashion. This means that, if $x > y$, then at least one instance is instantiated for each of the first y role templates. If $x < y$, instead, then each role template is instantiated, roughly, x/y times. Once the instances have been generated, we generated 100 access requests by randomly selecting the role instances to use as target users.

The results of these experiments, reported in Figure 1, show that the average response time depends only on the number or templates in the policy since access requests are *indexed* by the role instance of the user requesting access. We notice that the average evaluation time is below $50ms$ when the number of tole templates is below 20.000 and, in all cases, it is below $300ms$.

Next we run several experiments with different numbers of role templates and role instances. All the experiments show similar behaviour in terms of performance. Here we report the results only for the experiments with number of templates equal to 1.000 and number of role instances equal to 10.000.

Then we evaluated the impact of the spatial hierarchy over system performance. We have first considered a *linear* hierarchy consisting of two disjoint sets of square-shaped logical locations $S_N = \{loc_0, \ldots, loc_{k-1}\}$ and $S_I = \{loc_k, \ldots, loc_{n-1}\}$. Each set S_X consists of locations of size $\{1, 2, \ldots, |S_X|\}$ where location of size s contains all locations of size s', with $1 \leq s' < s$, and it is included in all locations of size s'', with $s < s'' \leq |S_X|$. We will call *intersecting locations* the ones in S_I intersecting the unit-square in the first quadrant, and *non-intersecting locations* the ones in the set S_N. Each logical location loc_i is associated to a different location type $type_i$. Type $type_i$ is more specific than type $type_j$ if and only if the loc_i is contained in loc_j. Figure 2 shows a graphical representation of the linear hierarchy. We observe that the graph of location types consists of two linear components, a first one consisting of the path $(type_0, type_1, \ldots, type_{k-1})$ and the second one consisting of the path $(type_k, type_{k+1}, \ldots, type_n)$. Our experiments pick a resource contained in loc_k, i.e., the unit-square location in the first quadrant, and a number of access requests generated by randomly chosen users, (at least) half of which do have the rights to access to the resource. Such spatial hierarchy has been chosen because, as stated in Section 8.3, the evaluation of the most specific context requires (a) the computation of the smallest location in S_I containing

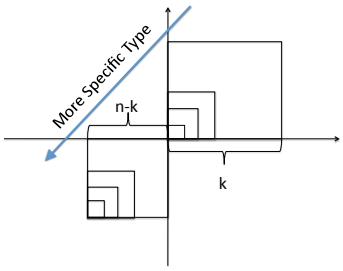

Figure 2: Linear hierarchy with n locations and $n-k$ intersecting locations.

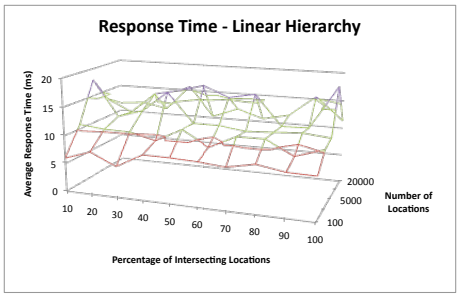

Figure 3: Response Time for Linear hierarchy without events.

the user position and (b) the execution of a DFS over the graph of location types. We use the underlying library interfaces to compute (a) and we have implemented a DFS algorithm for (b). The spatial hierarchy defined above has the property that, for a given k, it maximizes the number of locations in S_I.

The results of the experiments, reported in Figure 3, show that, for the linear hierarchy, the average evaluation time is always below $20ms$, independently from the percentage of the intersecting locations. As expected, response time slightly increases as the number of locations increases. Fluctuations are mainly due the the reasonable hypothesis that access requests are generated by random users at random places.

We have then considered the impact of events on the average evaluation time. In these experiments we have associated to each role template a different event that was used in the only activation rule associated to the template. We have then populated the environment with a number of *active events*. Clearly, whenever the number of active events is greater than the number of templates, such extra events influence the system only in terms of performance while they cannot influence the access control behaviour. Our experimental evaluation shows that also in this case the percentage of intersecting locations does not influence the running time of the system. For this reason we report in Figure 4 the average evaluation time only for a linear hierarchy with 50% of intersecting locations. As already anticipated in Section 8.3, the impact of event management in access control decision making is negligible.

Subsequently, a different scenario has been evaluated, in which the space hierarchy is shaped like a quad-tree. More precisely, the hierarchy is defined recursively as follows. Each location at layer i, for $i = 0, \ldots, \ell - 1$, is a square of size $2^{\ell-i-1}$. Each such location contains 4 square-shape equally sized locations at layer $i+1$. Each logical location loc is associated to a different location type t_{loc}. Each location type of layer $i+1$ is more specific that the location type of the logical location of layer i that contains it. Figure 5 reports the structure of the quad-tree space hierarchy. Given such a definition, the number of logical locations in a structure with ℓ layers, from 0 to $\ell-1$ can be approximated using the formula $4^{\ell}/3$. Figure 6 reports the average response time obtained by our experiments. Also in this case, the average response time is acceptable as in all cases is below $80ms$. Figure 7 reports the comparison

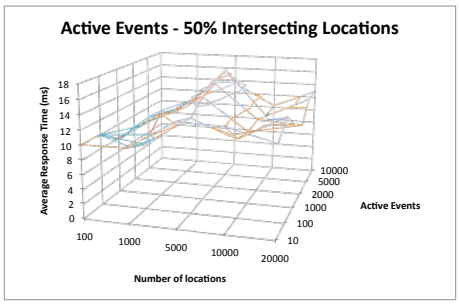

Figure 4: Response Time for Linear hierarchy with events.

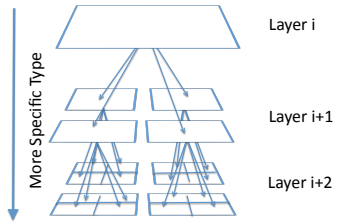

Figure 5: Quadtree hierarchy: recursive definition.

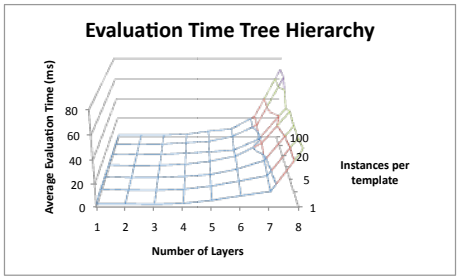

Figure 6: Response time for quadtree hierarchy.

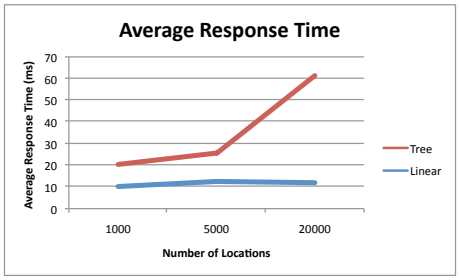

Figure 7: Comparison between linear and tree spatial hierarchy.

between the response time for linear and tree-shaped hierarchies with a comparable number of locations and types. From this picture it is clear that, whenever the graph of location types become more complex, the response time increases considerably.

9 Conclusions and future work

We introduced a reactive policy model, ERBAC, where policy rules are triggered by context-dependent conditions involving time, space, and application-dependent events, possibly generated by devices in a ubiquitous computing application scenario.

The policy model is equipped with formal syntax and semantics. We have shown its expressiveness through a variety of examples. It turned out that ERBAC can naturally handle emergency-handling policies, by adapting and possibly relaxing access control contraints to face abnormal situations that require ad hoc decisions. Such capability is supported by a flexible overriding mechanism, based on explicit priorities as well as (implicit) specificity-based rule ordering.

The policy model has been implemented by extending XACML's syntax and by constructing a suitable policy decision point (PDP). Experimental evaluation shows that the overhead caused by context-based, policy-rule evaluation is moderate and compatible with many real-world applications, even if our stress tests show that response time may increase non-linearly as the space hierarchy becomes more complicated.

We conclude that the framework is ready for experimental deployment in real ubiquitous computing applications. Additional experiments should be targeted at potential hardware-related limitations, such as the overhead caused by limited device computing power and network/wireless transmission delay. In this context, device and transmission reliability may cause further issues; ERBAC's ability of handling exceptional situations may prove to be suitable for mitigating this category of problems.

References

[1] XACML enterprise. http://code.google.com/p/enterprise-java-xacml/.

[2] XACML light. http://sourceforge.net/projects/xacmllight/.

[3] R. Abdunabi, I. Ray, and R. France. Specification and analysis of access control policies for mobile applications. In *Proceedings of the 18th ACM symposium on Access control models and technologies*, SACMAT '13, pages 173–184, New York, NY, USA, 2013. ACM.

[4] S. Aich, S. Mondal, S. Sural, and A. Majumdar. Role based access control with spatiotemporal context for mobile applications. In *Transactions on Computational Science IV*, volume 5430 of *LNCS*, pages 177–199. Springer, 2009.

Table 4: Glossary

Function Name/Symbol	**Meaning**	**Defined in**
$\subset_{LL}$	Partial order over logical locations	Section 4.1.2
$\preceq_{LT}$	Partial order over location types	Definition 6
$<$	Partial order defined over conditional event expressions	Definition 23
Caused()	Subset of the conditional event expressions in the REB that are used for access control decisions at time t, including triggers and run-time requests.	Definition 26
$CEXP$	Conditional event expression in the form $\langle cond, ev\rangle$ with $cond \in STECond$ and $ev \in SEXP \cup PEXP$	Definition 18
$conf(x)$	Denotes the SEXP conflicting with x	Definition 17
$ECond$	Event Condition	Definition 12
$Eloc : E \to PL \cup \{\perp\}$	Returns the physical location in which an event has been generated	Definition 9
$Eprios : E \to Prios$	Returns the priority of an event	Definition 9
$EV(t)$	Set of prioritized event expressions received at time t.	Definition 25
$EVisible : E \to LL \cup \{\perp\}$	Returns the set of logical locations in which an event is visible	Definition 9
$generatedBy : E \to U \cup \{\perp\}$	Maps an event to the user who generated it	Definition 9
$generatedFor : E \to 2^U \cup \{\perp\}$	Maps an event to the set of users to whom it is addressed	Definition 9
LL	Set of Logical Locations	Section 4.1.2
$LofT : LT \to 2^{LL} \cup \{\perp\}$	Maps locations type to logical locations	Definition 5
LT	Set of Location Types	Section 4.1.2
$LtoT : LL \to 2^{LT}$	Maps logical locations to location types	Definition 5
$MostSpecific(S)$	The subset of event expressions in S that are minimal w.r.t. the $<$ partial order	Definition 24.
$NonBlocked(S)$	The subset of mutually compatible event expressions in S	Definition 21.
$PEXP$	Prioritized Event Expression: "p: enable/disable r [for u]"	Definition 16
PL	Set of Physical Locations	Section 4.1.2
$PtoL : PL \to 2^{LL} \cup \{\perp\}$	Maps physical locations to logical ones	Definition 5
$Q(t)$	Subset of the conditional event expressions in the REB whose conditions are met by the system state at time t.	Definition 26
$SCond$	Spatial Condition	Definition 11
$SEXP$	Simple Event Expression: "enable/disable r [for u]"	Definition 16
$Sol(I, P)$	Returns the (explicit) set of time intervals identified by the temporal exp. (I, P)	Definition 3
$ST(t)$	Set of roles that are globally enabled at time t	Definition 25
$TCond$	Temporal Condition	Definition 10
$time()$	current time	Definition 3
$UserPos : U \to PL$	Returns the physical locations of a device	Definition 5

[5] S. Aich, S. Sural, and A. Majumdar. STARBAC: Spatiotemporal role based access control. In *Proceedings of the 2007 OTM confederated international conference: CoopIS, DOA, ODBASE, GADA, and IS-Volume Part II*, pages 1567–1582. Springer-Verlag, 2007.

[6] J. Bacon, K. Moody, and W. Yao. A model of OASIS role-based access control and its support for active security. *ACM Trans. Inf. Syst. Secur.*, 5(4):492–540, 2002.

[7] E. Bertino, C. Bettini, E. Ferrari, and P. Samarati. An access control model supporting periodicity constraints and temporal reasoning. *ACM Trans. Database Syst.*, 23(3):231–285, 1998.

[8] E. Bertino, P. A. Bonatti, and E. Ferrari. TRBAC: A temporal role-based access control model. *ACM Trans. Inf. Syst. Secur.*, 4:191–233, August 2001.

[9] P. Bonatti, C. Galdi, and D. Torres. ERBAC prototype implementation. http://wpage.unina.it/clemente.galdi/ERBAC.

[10] P. Bonatti, C. Galdi, and D. Torres. Erbac: event-driven RBAC. In *Proceedings of the 18th ACM symposium on Access control models and technologies*, SACMAT '13, pages 125–136, New York, NY, USA, 2013. ACM.

[11] L. Chen and J. Crampton. On spatio-temporal constraints and inheritance in role-based access control. In *Proceedings of the 2008 ACM symposium on Information, computer and communications security*, ASIACCS '08, pages 205–216, New York, NY, USA, 2008. ACM.

[12] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms (3. ed.)*. MIT Press, 2009.

[13] M. J. Covington, P. Fogla, Z. Zhan, and M. Ahamad. A context-aware security architecture for emerging applications. In *Proceedings of the 18th Annual Computer Security Applications Conference*, ACSAC '02, pages 249–, Washington, DC, USA, 2002. IEEE Computer Society.

[14] M. J. Covington, W. Long, S. Srinivasan, A. K. Dev, M. Ahamad, and G. D. Abowd. Securing context-aware applications using environment roles. In *Proceedings of the sixth ACM symposium on Access control models and technologies*, SACMAT '01, pages 10–20. ACM, 2001.

[15] M. L. Damiani, E. Bertino, B. Catania, and P. Perlasca. GEO-RBAC: A spatially aware RBAC. *ACM Trans. Inf. Syst. Secur.*, 10(1), 2007.

[16] D. F. Ferraiolo, R. S. Sandhu, S. I. Gavrila, D. R. Kuhn, and R. Chandramouli. Proposed NIST standard for role-based access control. *ACM Trans. Inf. Syst. Secur.*, 4(3):224–274, 2001.

[17] C. K. Georgiadis, I. Mavridis, G. Pangalos, and R. K. Thomas. Flexible team-based access control using contexts. In *SACMAT*, pages 21–27, 2001.

[18] L. Giuri and P. Iglio. Role templates for content-based access control. In *Second ACM Workshop on Role-Based Access Control*, pages 153–159, 1997.

[19] J. Joshi, E. Bertino, U. Latif, and A. Ghafoor. A generalized temporal role-based access control model. *IEEE Trans. Knowl. Data Eng.*, 17(1):4–23, 2005.

[20] D. Kulkarni and A. Tripathi. Context-aware role-based access control in pervasive computing systems. In *13th ACM Symposium on Access Control Models and Technologies (SACMAT 2008)*, pages 113–122, 2008.

[21] D. Kulkarni and A. R. Tripathi. A framework for programming robust context-aware applications. *IEEE Trans. Software Eng.*, 36(2):184–197, 2010.

[22] O. G. Morchon and K. Wehrle. Efficient and context-aware access control for pervasive medical sensor networks. In *PerCom Workshops*, pages 322–327. IEEE, 2010.

[23] O. G. Morchon and K. Wehrle. Modular context-aware access control for medical sensor networks. In B. Carminati and J. Joshi, editors, *SACMAT 2010, 15th ACM Symposium on Access Control Models and Technologies, Pittsburgh, Pennsylvania, USA, June 9-11, 2010, Proceedings*, pages 129–138, 2010.

[24] OASIS Consortium. Core and hierarchical role based access control (rbac) profile of xacml v2.0. http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-rbac-profile1-spec-os.pdf.

[25] OASIS Consortium. extensible access control markup language (XACML), v. 2.0.

[26] OpenGIS Consortium. Geography Markup Language (GML) simple features profile. http://www.opengeospatial.org/standards/gml.

[27] OpenGIS Consortium. Geospatial eXtensible Access Control Markup Language (GeoXACML) v 1.0. http://www.opengeospatial.org/standards/geoxacml.

[28] OpenGIS Consortium. GeoXACML Implementation Specification - Extension B (GML3) Encoding. http://www.opengeospatial.org/standards/gml.

[29] OpenGIS Consortium. Implementation Standard for Geographic information - Simple feature access - Part 1: Common architecture. http://www.opengeospatial.org/standards/sfa.

[30] S. Osborne, editor. *Fifth ACM Workshop on Role-Based Access Control*, Berlin, 2000. ACM.

[31] J. Park and R. S. Sandhu. The UCON_{ABC} usage control model. *ACM Trans. Inf. Syst. Secur.*, 7(1):128–174, 2004.

[32] I. Ray, M. Kumar, and L. Yu. Lrbac: A location-aware role-based access control model. In A. Bagchi and V. Atluri, editors, *ICISS*, volume 4332 of *Lecture Notes in Computer Science*, pages 147–161. Springer, 2006.

[33] I. Ray and M. Toahchoodee. A spatio-temporal role-based access control model. In S. Barker and G.-J. Ahn, editors, *Data and Applications Security XXI*, volume 4602 of *Lecture Notes in Computer Science*, pages 211–226. Springer Berlin / Heidelberg, 2007.

[34] I. Ray and M. Toahchoodee. A spatio-temporal access control model supporting delegation for pervasive computing applications. *Trust, Privacy and Security in Digital Business*, pages 48–58, 2008.

[35] G. Sampemane, P. Naldurg, and R. H. Campbell. Access control for active spaces. In *Proceedings of the 18th Annual Computer Security Applications Conference*, ACSAC '02, pages 343–, Washington, DC, USA, 2002. IEEE Computer Society.

[36] R. S. Sandhu. Role hierarchies and constraints for lattice-based access controls. In E. Bertino, H. Kurth, G. Martella, and E. Montolivo, editors, *ESORICS*, volume 1146 of *Lecture Notes in Computer Science*, pages 65–79. Springer, 1996.

[37] R. S. Sandhu, editor. *Second ACM Workshop on Role-Based Access Control*, Fairfax, VA., 1997. ACM.

[38] R. S. Sandhu, editor. *Third ACM Workshop on Role-Based Access Control*, Fairfax, VA., 1998. ACM.

[39] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, 1996.

[40] V. Solutions. JTS topology suite. http://www.vividsolutions.com/jts/jtshome.htm.

[41] M. Strembeck and G. Neumann. An integrated approach to engineer and enforce context constraints in rbac environments. *ACM Trans. Inf. Syst. Secur.*, 7(3):392–427, Aug. 2004.

[42] Sun Microsystems. Sun's xacml implementation. http://sunxacml.sourceforge.net.

[43] F. Turkmen and B. Crispo. Performance evaluation of XACML PDP implementations. In *Proceedings of the 2008 ACM workshop on Secure web services*, SWS '08, pages 37–44, New York, NY, USA, 2008. ACM.