Journal of Computer Security 0 (2015) 1–0
IOS Press

Event-Driven RBAC

Piero Bonatti [a], Clemente Galdi [a,*] and Davide Torres [b,**]
[a] *Dipartimento di Ingegneria Elettrica e Tecnologie dell'Informazione, Università di Napoli "Federico II", Via Claudio, 80125, Napoli (Italy)*
{pieroandrea.bonatti, clemente.galdi}@unina.it
[b] *Publiservizi s.r.l.*
E-mail: torres.davide@gmail.com

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—and its implementation. While almost all the individual features of ERBAC occur separately in some previous model, the detailed design of the policy language, its implementation in XACML, and its testing contribute to the development of expressive, event-driven policy frameworks by demonstrating that this rich model can be satisfactorily implemented, and that its expressivity and performance are compatible with a variety of realistic application scenarios. In particular, a number of examples illustrate ERBAC's expressive power, and its ability of handling exceptional situations in a flexible way, while keeping policies compact and manageable. The prototype extends XACML's language and the implementation of the PDP to support the new model. Systematic scalability experiments show that the computational cost of policy rule evaluation in ERBAC is compatible with real-world applications.

Keywords: Event-driven access control, Role Based Access Control

1. Introduction

The advent of pervasive/ubiquitous/mobile systems has increased the expressiveness requirements on policy models and languages, as well as the complexity of access control decisions. 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. The formulation of the current user position depends on the application scenario; space may be divided into several logical areas. On the one hand, the access control system might be used to control user access 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.*

* Corresponding author. E-mail: clemente.galdi@unina.it
** Work done while with the Università di Napoli "Federico II".

The ability of interacting with the environment is a characterizing feature of pervasive and ubiquitous systems. Such systems are able to obtain information from the environment and possibly use that information to influence the environment itself. The events detected by monitoring the environment 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 spatiotemporal 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*).

Several policy models extending standard models with spatiotemporal conditions have been studied, and some prototypes where policies are driven by sensor inputs have been experimented with. Still, some work remains to be done before a general purpose, spatiotemporal, and event-driven policy framework can be built. One reason is that the works in the literature focus on specific aspects (e.g., policy model only, implementation only, spatial or temporal conditions only, etc.). A fully detailed, general purpose, and event-driven policy language assessed with an extensive systematic validation is still missing. More precisely, a framework with *all* of the following features is still missing.

1. The policy framework should be essentially *policy neutral*, in order to be general purpose, that is, it should be expressive enough support a variety of policies based on dynamic environment conditions. The language should also be structured so as to facilitate the formulation, validation, and maintenance of complex policies, e.g. by incorporating models such as RBAC (that facilitates permission and user handling)[1] and XACML's well-structured policy *composition* features.
2. *Policies and mechanisms should be clearly separated*, according to a widely advocated practice. In some systems, policies are merged with the application code, and coded with the same language; this lack of structure makes it more difficult to understand, validate, and maintain policies, and makes them vulnerable to a wide range of programming errors. Such programming-framework approach is also in conflict with the following requirement.
3. A *complete, formal definition of syntax and semantics* of the policy language should be specified. Without such definition, the expressiveness boundaries of the policy language are not clearly specified, which prevents systematic testing of the language's performance as policies become more complex. Some papers present the language by way of examples; there is no closed list of operators. On the one hand, when some example cannot be captured right away, one may postulate the existence of more operators; in that case, however, the reported experiments (if any) don't reflect the impact of the new operators on language performance. A similar difficulty is encountered with some fully abstract policy models that represent time and space as generic sets of points, without providing any concrete syntax for spatiotemporal operators; experiments need concrete syntax and

[1] The RBAC paradigm [41,39,40,32] solves this issue by interposing roles between users and permissions, that is, 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. Another appealing feature is that the assignment of permissions to roles, which is perhaps the most delicate aspect in policy authoring, is relatively persistent. Moreover, when this mapping needs to be changed, it does not need to be reflected manually on a multitude of individual users. The success of RBAC models is also due to the fact that roles are a natural way of representing job functions within an organization [41,17].

operators to be carried out. Some implemented systems lack a formal semantics, which has serious implications in terms of potential ambiguities and security flaws, and it prevents any sound validation of security monitors.

4. The *expressiveness* of the policy framework should be validated by means of articulated examples. Emergency handling is of particular importance in pervasive computing applications; nonetheless only a few papers deal with this issue.
5. The practical applicability of the policy language should be further assessed by means of *systematic scalability tests*, so as to understand whether the nice dynamic constructs supported by a rich policy language can be actually used in practice.

Each work in the literature fails to address one or more of the above points. For instance, some policy models and implemented systems do not provide a detailed syntax for spatiotemporal and event conditions, which prevents systematic performance analysis. In some cases, behavior is not event-driven (i.e. conditions are checked only when an access request is received). Other papers include no formal semantics. Others do not support roles or policy composition. Last but not least, many works provide no performance evaluation. (A detailed analysis of related works is provided in the following sections.)

In this paper we contribute to the design of a general purpose, spatiotemporal and event-driven policy framework by tackling the aforementioned gap, that is, by: (i) specifying a fully detailed, expressive policy language, (ii) assessing its expressiveness by modelling articulated scenarios, including emergency handling situations, (iii) implementing the framework using as much as possible standard components, languages, and protocols, and (iv) carrying out systematic scalability tests. Here are more details:

1. A complete definition of syntax, including spatiotemporal operators, is provided, together with its formal semantics. Concerning expressiveness, the language supports: (i) roles and role templates, by adopting a rich, event-driven extension of the RBAC model, that we call ERBAC; and (ii) policy composition, by merging the dynamic features of ERBAC into XACML. The structure of XACML (in particular, its TARGET element) is very effective in selecting the relevant portion of a large policy for each access control decision, which may significantly improve the performance of the policy decision point. Detailed and complete syntax enables both formal semantics definition and sound expressiveness and scalability analysis, as explained in the previous paragraphs.
2. The policy model ERBAC comprises a rich set of features that occurred separately in previous models. Of course, such features include spatiotemporal conditions and event-based conditions. The representation of time is inherited from the TRBAC model, and the representation of space supports logical and physical locations, as well as location types (which is less common in the literature, but very useful in formulating policies). Behaviour is properly event-driven (in some other models the access control mechanism reacts only to standard (*subject,object,operation*) access requests). We introduce the notions of *personalized* and *localized* events in order to control the extension of event effects. In response to spatiotemporal and other dynamic conditions, ERBAC policies can enable and disable roles. Role templates are important, too, as they play a fundamental role in e-health applications (e.g. permitting access to the medical record of a given patient only to her doctor, as opposed to all doctors), that constitute a recurring reference scenario for dynamic policy models. Table 1 summarizes the differences between ERBAC and the other models with similar goals.
3. The expressiveness of the language's constructs is assessed through a number of articulated examples in Sec. 7. Our main reference scenario is focussed on the management of hospital resources, but we provide also some other examples to illustrate the general applicability of the framework. We explicitly address emergency handling policies—an important issue in medical and domotic ap-

plications, which is not widely addressed in the literature. The key language feature for emergency handling is policy *overriding*. In particular, by means of priority-based overriding and localized events, we show how to relax location-based policies (similar effects can be obtained for time-based policies, as well).

4. We introduce and describe a prototype implementation of ERBAC that extends Sun Microsystems' implementation of XACML [44]. The extension of XACML with dynamic conditions and spatiotemporal operators answers the following practical question: do the extension points of the XACML standard suffice to incorporate the dynamic feature of ERBAC? We answer positively this question by providing the details of an extension of XACML that fulfills this requirement.
5. The performance of part of the language features (such as policy composition) have been already assessed in previous work. Here we focus on the scalability of spatiotemporal conditions, by means of an original suite of systematic scalability tests. Currently, the language does not support event composition operators (such as sequence, co-occurrence, and so on); accordingly, scalability along that complexity dimension lies beyond the scope of this paper. In our reference examples, similar functionalities can be simulated using event attributes and the other constructs of the language.

Our analysis shows that the ERBAC-based policy framework

- is expressive enough to address an interesting range of application scenarios, including some non-trivial emergency handling examples;
- it can be effectively embedded in industrial-strength standards such as XACML, GeoXACML, and GML;
- the overhead introduced by composition and spatiotemporal constructs is moderate and compatible with a variety of applications; performance scales well as policies grow larger and more complex.

So, in summary, the ERBAC-based framework assesses the feasibility of a rich, dynamic, spatiotemporal RBAC model.

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 [33]. 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. In order to help the reader, we collected the glossary of main terms and concepts in Table 8.

2. Previous Works

In role based access control models [38], 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 [19] 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 [16,34] and temporal conditions [7,8,21]. Further context-aware RBAC models can be found in [5,35,36,6,15,12,18,21,43,37,22,23,11]. A comparison of the involved policy models is summarized in Table 1. A wider comparison includes works that do not feature an abstract policy model (nor any formal semantics, in general), and are system-oriented, or programming-framework oriented; it is summarized in Table 2. Notice that (i) we could not simply adopt any existing framework out of the box, since each of them is missing some important feature; (ii) to the best of our knowledge, no systematic scalability tests has been carried out for increasing policy complexity. The details are discussed in the following.

In [15], 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 [14] the authors provide an implementation of GRBAC where access control is managed by means of environmental roles.

In [22,23] 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 [24,25] 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

Table 1
A comparison of policy models with some form of dynamic conditions

Features:	Event driven	Time	Space physical	Space logical	Space types	Role templates	Conflict resolution
[4]		✓	✓	✓			✓
[5]		✓	✓	✓	✓		
[6]				(1)			
[7]		✓					
[11]		✓	✓	✓	✓		
[12]		✓		(1)			
[14]	✓	✓	✓	✓		(2)	
[15]	✓	✓	✓	✓		(2)	
[16]			✓		✓		
[18]		✓		✓			
[19]						✓	
[21]		✓					✓
[22]	✓	✓	✓	✓		✓	
[23]	✓	✓	✓	✓		✓	
[34]		✓	✓	✓			
[36]		✓	✓	✓			
ERBAC	✓	✓	✓	✓	✓	✓	✓

(1) No syntax / structure is specified for locations

(2) Simulated with environmental roles

Note: "Event driven" here means "supporting arbitrary, asynchronous, environment-generated events". Accordingly, the frameworks that support only temporal events – that can be predicted and scheduled in advance, as in [8] – are not classified as "event driven" in this table.

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 [5,4,35,36,12,11] 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.

While the traditional RBAC model requires users to explicitly activate the roles they need (in order to encourage the application of the least privilege principle), several of the frameworks designed for pervasive environments, including [24,25,4,16], support the automated activation and deactivation of roles based on spatio-temporal and event conditions. In some cases, e.g. [37,14], there is no distinction between role enabling and activation. ERBAC follows this approach, nonetheless separate enabling and activation can be easily simulated as shown in Sec. 5.2. One of the advantages of the automated activation approach is that carefully designed (de)activation conditions relieve the user from the responsibility of enforcing the minimal privilege principle and may provide better guarantees. Another advantage is improved user experience, as activation is determined by the user's natural interaction with the environment rather than by additional explicit actions that may look redundant to the user. It is our opinion that the attempt to use a resource should also work as a role activation request; in this way, enabling corresponds to the

Table 2
A comparison of relevant policy frameworks

	Event driven	Time	Space	Location types	Roles	Role templates	Formal semantics	Implem.	Scalability tests
[4]		✓	✓		✓		✓	✓	
[5]		✓	✓	✓	✓		✓		
[6]	✓	✓	(1)		✓		✓	✓	
[7]		✓					✓	(2)	
[8]		✓					✓	✓	
[11]		✓	✓	✓	✓		✓		
[12]		✓	(1)		✓		✓		
[14]	✓	✓	✓		✓			✓	
[15]	✓	✓	✓		✓		(2)	✓	
[16]			✓	✓	✓		✓		
[18]		✓	✓		✓			✓	
[19]					✓	✓	✓		
[21]		✓			✓		✓		
[22]	✓	✓	✓		✓	✓		✓	
[23]	✓	✓	✓		✓	✓		✓	
[24]	✓				✓		(3)		
[25]	✓				✓		(3)	✓	
[34]		(1)	✓		✓		✓		
[36]		✓	✓		✓		✓		
[37]			✓		✓		(3)	✓	
[43]		✓	✓		✓	✓		✓	
ERBAC	✓	✓	✓	✓	✓	✓	✓	✓	✓

(1) No syntax / structure is provided

(2) Partial

(3) Syntax is not fully specified

Note: "Event driven" here means "supporting arbitrary, asynchronous, environment-generated events". Accordingly, the frameworks that support only temporal events – that can be predicted and scheduled in advance, as in [8] – are not classified as "event driven" in this table.

spatio-temporal part of the condition associated to a role's activation rule, while the activation request corresponds to the event part of the same condition (where the event is generated by attempted resource usage). In such application contexts, we believe that separate enabling and activation constructs are less preferrable than a simpler syntax where the distinction between enabling and activation disappears. We expect only a small number of cases where the distinction is helpful, and in those cases the encoding illustrated in Sec. 5.2 can be used.

ERBAC is not the main contribution of our paper. Its features have been studied separately elsewhere; unfortunately, none of the above framework exhibits the full set of properties we need (cf. Tables 1 and 2), for example: We cleanly separate policies from the application logic, while [18,37,22,23] don't. In our reactive framework, the environment is monitored continuously (through an event-based mechanism) while [5,35,36,18,43,37] check policy conditions only once before granting access. Joshi [21] deals only with time and [37] only with space; [11] does not support general events. Neumann [43] does not provide

a formal model. Finally, we mention that to the best of our knowledge ERBAC's notions of personalized and localized events are novel. These considerations led us to introduce a policy model tailored to our goals, even if its novelty is moderate.

Our main contribution is the definition, implementation (including the extension of XACML with reactive policies), and evaluation—in terms of expressiveness as well as performance and scalability—of a rich reactive policy framework. In this respect, note that [5,35,36,6,15,18,21,11] do not illustrate any implementation, and none of the papers mentioned so far contains any experimental scalability testing. Similarly, no work has shown how to embed spatiotemporal conditions and reactive policies in XACML. A preliminary version of this paper appears in [10].

3. Preliminaries

The RBAC model we are going to extend in this paper has been introduced in [38] 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 organization. A role can be seen as a set of permissions that are needed to execute a given job within an organization. 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;*
- *$user : S \to U$, a function associating each session to the user who started it;*
- *$roles : S \to 2^R$, 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 ([35,36,5]). 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[2] 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}).

[2] 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:*

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

Table 3 contains the taxonomy of events as described above.

Table 3
Event Taxonomy

Property	**Name**	**Condition**
Visibility	Global	$Eloc = \bot$
	Localized	$Eloc \neq \bot$
Generator	System Generated	$GeneratedBy = \bot$
	User Generated	$GeneratedBy = u \in U$
Target Users	General	$generatedFor = \bot$
	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 contains.*
- $ObjCat : OBJ \rightarrow 2^{OC}$. *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 parametrized privilege as follows:

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

Similarly, we will use role templates as a means for defining generic, parametrized 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.

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

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* $REXP$*.*

As it has been done in TRBAC and in its derived models, we use the term *event expression* to refer to enabling/disabling requests. However, in the previous models an *event* was only used to identify policy and/or user requests, i.e., enable/disable role, activate/deactivate role, modify role permissions, modify user-role assignments. In ERBAC the concept of *event* is much wider and includes, as stated in Section 4.1.3, *all the aspects of the context that are different from time and space and that are monitored and/or measured*.

When conflicting events arise—that is, if one expression requires the enabling of a given role r and, at the same time, another one requires to disable the same role r—we need a way to decide which one should prevail. We observe that in this decision process, an expression that is bound to a specific user u, e.g, enable r for u, should not influence decisions regarding other users. On the other hand, a general expression, that is not restricted to any specific user, should, in principle, be able to prevail on every conflicting expression that is bound to a single user. Given the above discussion we can define the conflicts between expressions.

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

- $conf$(enable *r*) $=$ {disable *r*}
- $conf$(disable *r*) $=$ {enable *r*}
- $conf$(enable *r* for *u*) $=$ {disable *r*, disable *r* for *u*}
- $conf$(disable *r* for *u*) $=$ {enable *r*, 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:*

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

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:*

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

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.*

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

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:*

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

■

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*.*

5.1. Constraints

The ERBAC model does not include constraints. However, such a feature can be easily simulated using the ERBAC interaction with the environment and/or the use of triggers.

Separation of Duties. Separation of Duty is a principle that is used to guarantee that no single person is responsible for the completion of a whole task. Such type of constraint can be easily simulated using triggers as follows. Suppose r_1 and r_2 are two roles that should not be enabled at the same time by the same user. We can implement the SOD contain by using the following role triggers:

– $(p$*:enable* r_1 *for* $u), (enabled\ r_2\ for\ u) \rightarrow (p$*+1:disable* r_1 *for* $u)$
– $(p$*:enable* r_2 *for* $u), (enabled\ r_1\ for\ u) \rightarrow (p$*+1:disable* r_2 *for* $u)$

For example, whenever a user tries tries to enable role r_1, e.g., by entering a new location, the above trigger prevents such operation by generating an exception, with higher priority, that conflicts with the enabling request.

Threshold-based constraints. A second type of constraint allows the expression of conditions that depends on the cardinality of sets, e.g, of users. One example could be a constraint to describe the following access policy: *No more than three guests can use the control panel in the secure room at the same time*. In ERBAC we can use events for providing such type of constraints. We can assume that, whenever the maximum threshold is reached, the environment generates an event that is used to trigger the proper disabling rule in the REB. We can use the *generateBy* function in order to identify the last user who enabled the role and, thus, generated the event. In the above example, we can assume that there exists a role *Use_Panel* that allows guests to use the control panel. Furthermore, whenever *"the number of guests using the control panel in the secure room is four"*, the environment generates an event 4th_Guest. We can safely assume that the priority of event 4th_Guest is higher that the priority of AnyEvent. We can implement the above constraints by adding the following to the REB.

– $\langle(AnyTime, Secure\_Room, AnyEvent)$, enable $Use\_Panel\rangle\rangle$
– $\langle(AnyTime, Secure\_Room, 4th\_Guest)$,
disable Use_Panel for $generatedBy(4th\_Guest)\rangle\rangle$

As it will be more clear in next section, whenever the event 4th_Guest is active, the second conditional event expression will be "more specific" than the former one and, thus, will prevail.

5.2. Comparison with GTRBAC

The GTRBAC model introduces a number of different constraints both on role enabling/assignment and activation. We will review the differences between GTRBAC and ERBAC and show how it is possible to simulate features of the former model in the latter.

Simulating role activation/deactivation. The GTRBAC model distinguishes role enabling and role activation. In particular in GTRBAC, and in its derived models, it is assumed that while role enabling is *policy driven*, role activation is executed by the user at her own discretion and it is, this, *user driven*. Such distinction does not exists in ERBAC because our goal is to completely automate the access policy process by eliminating explicit user requests or delegating to the environment the sensing of such requests. However, role enabling and activation can be simulated in ERBAC by using triggers and events. We stress that, given the simulation below, if required it is possible to extend the ERBAC syntax in order to provide such a distinction. In the spirit of ERBAC, we will assume that the *user requests* for activation and deactivation of a given role r can be intercepted by the events *ActivateRole_r* and *DeactivateRole_r*, respectively, both having the same priority of *AnyEvent*[3]. In Table 4 we show how to simulate the separation between role enabling/disabling and activation/deactivation by using the events and role triggers. Let (I, P) be any temporal expression, let r be a role with associated permissions p. The simulating environment will include the roles try_r, r_e and r_a. The roles try_r and r_e have no associated permissions while r_a has associated permissions p. The idea for the simulation is the following. The enabling of role r corresponds to the enabling of role r_e (Rule 1) while the activation of role r corresponds to the enabling of role r_a. Whenever the user tries to activate the role r, the event *ActivateRole_r* is generated and the guarding role try_r is enabled (Rule 2). The role r is actually activated (i.e., the role r_a is enabled) only as a consequence of the trigger that fires if (a) the role r_e was already enabled and the enabling of role try_r occurs (Rule 3) or (b) the enabling requests for roles r_e and r_a occur simultaneously (Rule 4). The enabling of role try_r lasts one time unit and it is immediately disabled by a trigger (Rule 5). In this way, such a role can be used to register the occurring of the activation request only in the moment in which such a request is generated. The deactivation of role r corresponds to the deactivation of role r_a (Rule 6). The disabling of role r corresponds to the disabling of role r_e (Rule 7). Finally, when disabling the role r_e, we need to force the disabling of role r_a, too (Rule 8).

The GTRBAC model introduces a number of different enabling and activation constraints. We will discuss each of them separately.

Periodicity constraints on role enabling. Periodicity constraint in GTRBAC "specify the exact intervals during which a role can be enabled or disabled" and, thus, correspond exactly to Temporal Conditions in ERBAC.

[3] As we will see in next section, if the events e_1 and e_2 have the same priority, none of the expressions $\langle(s, t, e_1), p : ev\rangle$ and $\langle(s, t, e_2), p : ev\rangle$ is more specific than the other.

Table 4
Simulation of Role Enabling and Activation in ERBAC

GTRBAC	Rule no	ERBAC
$((I,P)$, enable $r)$	1	$\langle(AnyPlace,(I,P),AnyEvent)$, enable $r_e\rangle$
$((I,P)$, activate $r)$	2	$\langle(AnyPlace,(I,P),ActivateRole\_r)$, enable $try\_r\rangle$
	3	enabled r_e, enable $try\_r \rightarrow$ enable r_a
	4	enable r_e, enable $try\_r \rightarrow$ enable r_a
	5	enabled $try\_r \rightarrow$ disable try_r
$((I,P)$, deactivate $r)$	6	$\langle(AnyPlace,(I,P),DeactivateRole\_r)$, disable $r_a\rangle$
$((I,P)$, disable $r)$	7	$\langle(AnyPlace,(I,P),AnyEvent)$, disable $r_e\rangle$
	8	disable $r_e \rightarrow$ disable r_a

Duration constraints on role enabling. GTRBAC defines the *role enabling events* "enable/disable r" and the *assignment events* in the form $assign_p/deassign_p$ p to r (that assign/deassign permission p to/from tole u) or $assign_U/deassign_U$ u to r (that allow/disallow user u to enable and activate role r). The assignment events can be statically simulated by introducing new roles corresponding to the possible role-permissions and user-roles assignments. The enabling of the new roles can be triggered by specific events; see also the analogous simulation approach in [20, Sec. 3.1]. For this reason we will only concentrate the simulation on role enabling events.

Duration constraints in GTRBAC are used to modify the amount of time in which a role can be enabled. Notice that the policy can specify a periodic expression in which the role *might be enabled*. The duration constraint can reduce the amount of time in which a role is *actually enabled*. We thus need to distinguish the policy driven role enabling from the *actual role enabling request*. In ERBAC we model the latter by means of an $Enable_r$ event.

Duration constraints can be defined both *per-role*, i.e., the constraint holds for a specific role, and *per-user-role*, i.e., the constraint holds for a specific user enabling/activating a specific role. In the following we show how to simulate per-role duration constraints. The same technique can be used to simulate per-user-role constraints by simply substituting "*enable/disable* r" with "*enable/disable* r for u". GTRBAC specifies three possible syntax for duration constraints. In the following E denotes the simple event *enable* r. The same simulation idea, with the obvious modifications, can be applied in case E denotes the event *disable* r.

T1: $((I,P),D_R,p:E)$. The event E is valid for the duration D_R within each valid periodic interval specified by (I,P). This means that the role enabling request will be ignored if it occurs outside $Sol(I,P)$. On the other hand it will lead to the role enabling if it occurs within the time periods defined by the periodic expression (I,P). Furthermore, in the latter case, the validity of the enabling request will last for at most D_R time units. After such amount of time the role will be automatically disabled.

T2: $(D_R,p:E)$. The event E is valid 'once', for the maximum duration D_R that means that role enabling can occur at any time. Once the enabling request is received, it is valid for D_R time units. After such amount of time the role is disabled.

T3: $(D,D_R,p:E)$. The constraint $c=(D,D_R,p:E)$, with $D_R \leq D$, can be seen as a constraint $(D_R,pr:E)$ with a limited validity D. That is, the constraint c can be enabled at time t, e.g., by

a run-time request, and it affects access control decision until time $t + D$. During c's validity, the duration restriction D_R applies to event E.

We first show how to simulate periodic duration constraints $((I, P), D_R, p : E)$ of type $T1$ above and we then extend the solution to the other types of constraints. Let us assume that $I = [b, e]$ and let $\hat{I} = [e, e]$. The simulation of duration constraints for role enabling can be obtained using the following events and triggers.

1. $\langle(AnyPlace, (I, P), Enable_r), p$*:enable* $r\rangle$.
2. *enable* $r \rightarrow$ *disable* r *after* D_R.
3. $\langle(AnyPlace, (\hat{I}, P), AnyEvent), Max_P$*:disable* $r\rangle$

Rule 1 binds the *actual* enabling of role r to the role enabling request. Rule 2 triggers the disabling of role r after the maximum allowed duration D_R. Rule 3, if required, periodically disables the role r whenever the policy requires to.

The duration constraints of type $T2$ are simpler than periodic ones since they are 'one-shot' constraints. Thus in order to obtain a simulation for this type of constraints it is sufficient to remove periodicity by (a) substituting in Rule 1 (I, P) with $AnyTime$ and (b) removing Rule 3. The former substitution makes the enabling request always valid while the latter removal prevents to renew the validity of the enabling for more than once.

Finally, in order to simulate $T3$ constraints, it is sufficient to consider the constraint c itself as a role and manage its duration D by adding a guarding role c_g.

Temporal Constraints on Role Activations. GTRBAC defines two types of temporal constraints on role activations. The first one, called the *Total active duration constraint*, restricts the *sum* of all activation durations to be upper bounded by some value. The second one, called the *The maximum duration constraint*, limits the maximum allowable duration of *each* activation of a role. The total active duration constraint can be simulated by associating to each role r an event *TimeExceeded_r* whose generation is used to trigger the disabling of role r. The maximum duration constraint can be simulated by combining the simulation of role activation and temporal constraint on role enabling.

Cardinality contraints on role activations. GTRBAC defines two types of cardinality constraints. The total number of activations and the maximum number of concurrent activations.

The total number of activation constraint forces a role r to be enabled (activated in the GTRBAC parlance) at most n times during a given period of time. It is possible to simulate such a constraint by using n guard roles $r_{g_1}, \ldots, r_{g_n}$. Intuitively, if r_{g_i} is enabled but $r_{g_{i+1}}$ is not, then role r has been enabled i times. The simulation of such a constraint can be done as follows:

1. $\neg enabled(r_{g_1})$, enable $r \rightarrow$ enable r_{g_1}
2. $enabled(r_{g_i}), \neg enabled(r_{g_{i+1}})$, enable $r \rightarrow$ enable $r_{g_{i+1}}$ (For $i = 1, \ldots n-1,$)
3. $enabled(r_n)$, enable $r \rightarrow Max_P$:disable r

The *maximum number of concurrent activation constraint.* limits to n the number activation for a given role r that can be executed concurrently. Clearly, the difference between these types of constraint is that in the latter case every role disabling/deactivation should allow the possibility of enabling another instance of the same role. In order to simulate such a constraint in ERBAC, it is sufficient to add the following triggers to the ones shown above:

1. $enabled(r_{g_n})$, disable $r \rightarrow$ disable r_{g_n}
2. $enabled(r_{g_i}), \neg enabled(r_{g_{i+1}})$, disable $r \rightarrow$ disable r_{g_i} (For $i = 1, \ldots n-1,$)

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 $q \in Prios$ and $cev \in conf(ev)$ such that $q : cev \in S$ 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* $sc_1, sc_2 \in LT$
- $\forall tl_1 \in LtoT(sc_1) : sc_2 \not\preceq_{LT} tl_1$ $\wedge$ $\exists tl_1 \in LtoT(sc_1) : tl_1 \preceq_{LT} sc_2$, *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 either of type $OperatingRooms$ or $Department$, 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).
Consider the case in which 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

$$
\begin{array}{l}
MostSpecific(S) = \{ \\
\quad \langle(WorkingHours, OperatingRoom1, AnyEvent)\rangle, \textit{enable } Surgeon\langle OperatingRoom1\rangle\rangle \\
\}
\end{array}
$$

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:*

$$
\begin{array}{l}
MostSpecific(S) = \{ \\
\quad \langle(AnyTime, OperatingRoom1, SurgeryInProgress)\rangle, \textit{enable } Surgeon\langle OperatingRoom1\rangle \\
\}
\end{array}
$$

■

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 $\Delta t) \in RQ(t - \Delta t)$ then $p : ev \in Caused(t, EV, ST, \mathcal{R}, RQ)$*
2. *If $[E_1, \ldots, E_n, C_1, \ldots, C_k \rightarrow p : ev$ after $\Delta t] \in c$ and the following hold:*
   - *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 existsï£¡ $p : E_i \in Caused(t - \Delta t, EV, ST, \mathcal{R}, RQ)$ 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:* $(Eloc(ec) = \bot \wedge generatedFor(ec) = \bot)$
   $\forall u \in \{v \in U \mid UserPos(v) \models sc\}$
   (b) *Global, Personalized Events:* $(Eloc(ec) = \bot \wedge generatedFor(ec) \neq \bot)$
   $\forall u \in \{v \in generatedFor(ec) \mid UserPos(v) \models sc\}$.
   (c) *Localized, General Events:* $(Eloc(ec) \neq \bot \wedge generatedFor(ec) = \bot)$
   $\forall u \in \{v \in U \mid UserPos(v) \models sc \wedge EVisible(ec) \cap PtoL(UserPos(v)) \cap \emptyset\}$.
   (d) *Localized, Personalized Events:* $(Eloc(ec) \neq \bot \wedge generatedFor(ec) \neq \bot)$
   $\forall u \in \{v \in generatedFor(ec) \mid UserPos(v) \models sc \wedge EVisible(ec) \cap PtoL(UserPos(v)) \neq \emptyset\}$.

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 behaviour 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;*
- *$user : S \to U$. 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.

- *$roles : S \to 2^R$. 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 behavior 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* behavior 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. *⟨(WorkingHours, Cardiology, AnyEvent) enable Doctor⟨$Cardiology$⟩⟩*
2. *⟨(AnyTime, Cardiology, CriticalPatient) enable Doctor⟨$Cardiology$⟩⟩*

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.*[4] ■

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.

[4] 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.*

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

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:*

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

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

In this case, the REB might contain the following:

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

The first conditional event expression defines the normal operational behavior 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 implicitly 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 *personalized*. 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:*

$$
\begin{array}{l}
User\langle UserName\rangle = role(\\
\quad (x, Directory, DirectoryName = UserName) \\
\quad (rw, Files, true) \\
) \\
\\
LimitedUser\langle UserName\rangle = role(\\
\quad (x, Directory, DirectoryName = UserName) \\
\quad (r, Files, true) \\
)
\end{array}
$$

Let $QuotaExceeded \in E$ *be the user generated event corresponding to quota exceeded*[5]*. We can use the following rules.*

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

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 behavior 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 $Assistant\langle OperatingRoom1\rangle$

[5]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 distinction is immaterial.

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 modelling 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:

1. $\langle (AnyTime, AnyPlace, AnyEvent)$, disable $Porter\langle lloc \in LofT(Pavilion)\rangle\rangle$
2. $\langle (AnyTime, AnyPlace, TransferRequired)$, enable $Porter\langle$
   $lloc \in EVisible(TransferRequired) \mid pavilion \in LtoT(lloc)\rangle\rangle$

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:

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

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 $OperartingRoomNurse\langle Surgery\rangle\rangle$

$\ldots$

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:

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

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

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.

$$
\begin{array}{l}
\langle (AnyTime, AnyPlace, MedicalEmergency), \text{enable } Doctor\langle MedicalEmergency\rangle\rangle \\
\ldots \\
\langle (AnyTime, AnyPlace, AnyEvent), \text{disable } Anesthetist\langle Hospital, Department\rangle\rangle \\
\langle (AnyTime, AnyPlace, AnyEvent), \text{disable } Surgeon\langle Hospital, Department\rangle\rangle \\
\ldots \\
\langle (AnyTime, Hospital, AnyEvent), \text{enable } Anesthetist\langle Hospital, Department\rangle\rangle \\
\langle (AnyTime, Hospital, AnyEvent), \text{enable } Surgeon\langle Hospital, Department\rangle\rangle \\
\ldots
\end{array}
$$

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 [33] 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 authorization 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 authorization 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 authorization 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 [27] 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 [27], XACML Enterprise [1] and XACML Light [2]. Among the available implementations, we have extended the open source java SUN's implementation XACML [27] since it has a modular and easily extendible architecture. Furthermore, as noted in [45], 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 [27] 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 authorization decision. A Policy set is, informally, a set of policies. In [26] 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 [27] 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 representa-

tion 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 [29], an extension of the XACML standard by the OpenGIS Consortium [31,28,30]. GeoXACML can describe arbitrary geometric shapes starting from the following base types [31]: 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) [28].
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 5 we report the syntax for the specification of role templates and periodic expressions. In Table 6 we report the functions we have implemented for managing the new conditions and data types.

Our prototype implementation extends SUN's XACML [44]. 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 [42] 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 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 [45] 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

Table 5
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}

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 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

Table 6
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	
$Date$: DateTimeAttribute $Pexp$:PeriodicExpressionAttribute	$Date \in Sol(Pexp)$
urn:my:function:time:not-inside	
$Date$: DateTimeAttribute $Pexp$:PeriodicExpressionAttribute	$Date \notin Sol(Pexp)$
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 it is visible by UserName in Point
my:rule-combining-algorithm:deny-overrides	
	Rule combining algorithm evaluating the MostSpecific rule.

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 [13], 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 describes 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. 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. 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.

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 have run is used to establish a baseline against which we measure the impact of the ERBAC model over the existing implementation. In order to do so, we need to design an experiment

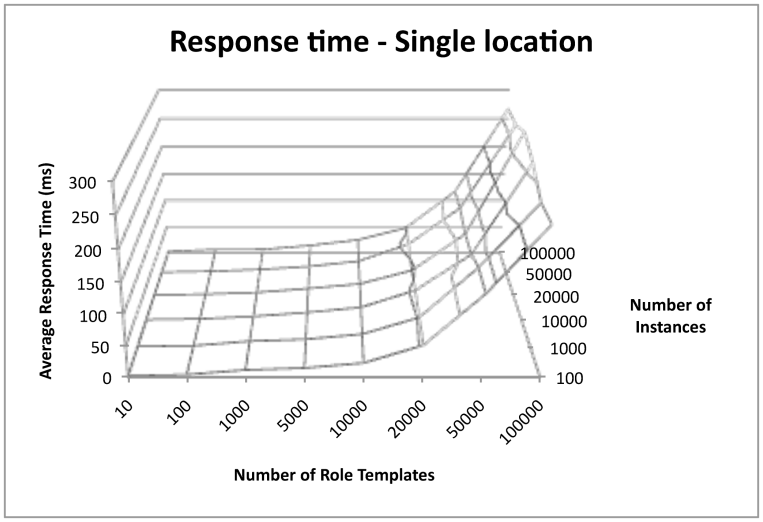


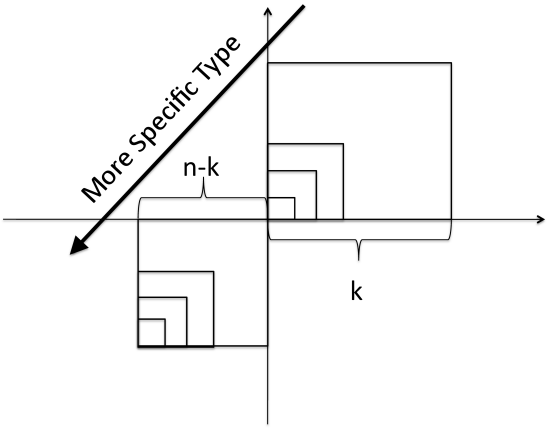
Fig. 1. Evaluation time as function of number of policies and instances.

in which the decision process depends neither on the spatial hierarchy nor on the set of active events. For this reason we have considered policies defined over a single logical location, the *whole space*, with *no active events*. Given this naïve scenario, 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.

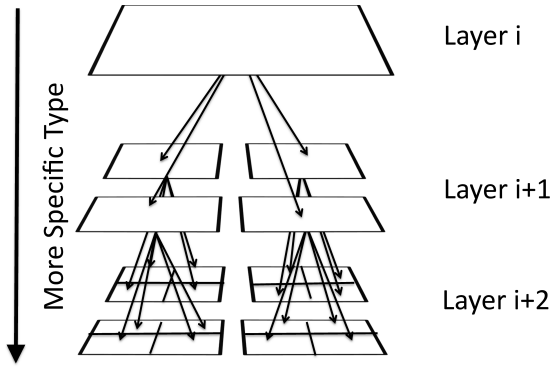
The results of these experiments, reported in Figure 1, show that the average response time depends only on the number or templates in the policy since access requests are *indexed* by the role instance of the user requesting access. We notice that the average evaluation time is below $50ms$ when the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

We have then evaluated the impact of space hierarchy and event management on the response time. For each experiment type, we have run different experiment sets with variable number of role templates and instances. For each set the behaviour of the response time is similar to the one shown in Figure 1. For this reason, in the following graphs and tables, we only report the results obtained with 1000 templates and 10000 role instances.

We started by evaluating the impact of the spatial hierarchy over system performance. We preliminarily recall that the access control decision for a given request depends on the most specific rule among *all* the rules that are applicable for the specific request. In order to do so, whenever a request is made, the system needs to recover *all* the logical locations that include the location P in which the request has been generated. As stated in Section 8.3, if n is the total number of locations and k is the number of locations containing P, the time needed to recover all the k locations of interest is $O(k + \log n)$.



(a) Linear hierarchy with n locations and $n-k$ intersecting locations.



(b) Quadtree hierarchy: recursive definition.

Fig. 2. Spatial hierarchies.

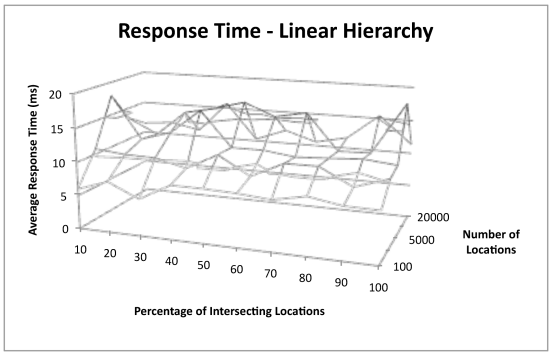
In order to evaluate the performance of the selection algorithm we have considered two different topologies for space hierarchy. The first one allows to have a number k of locations of interest that can range from 1 up to n. The second one is a quad-tree topology in which the number n of locations is exponential in the number k of locations of interest. The former topology stresses the algorithm as the number of locations intersecting the user location increases (this value coincides with k). The latter tests the scalability of the algorithm as the total number of locations increases while keeping the number of intersecting locations fixed.

Accordingly, we have first considered a *linear* hierarchy consisting of two disjoint sets of square-shaped logical locations $S_I = \{loc_0, \ldots, loc_{k-1}\}$ and $S_N = \{loc_k, \ldots, loc_{n-1}\}$. 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 2a 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 $(type_k, type_{k+1}, \ldots, type_n)$.

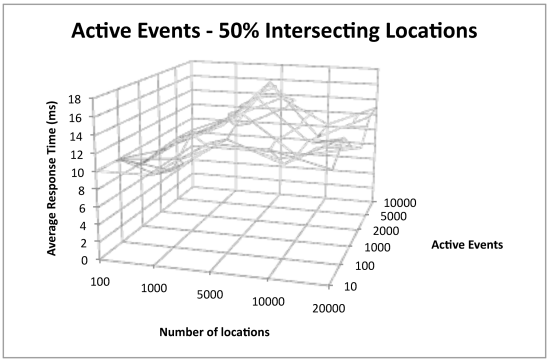
The access requests are always generated by picking a point in loc_k, i.e., a point intersecting all the locations in S_I. Furthermore, the set of requests is randomly generated in such a way at least half of the request will be successful in gaining the access to the resource.

Such spatial hierarchy has been chosen because, as stated above, the evaluation of the most specific context requires (a) the computation of the smallest location in S_I containing 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.

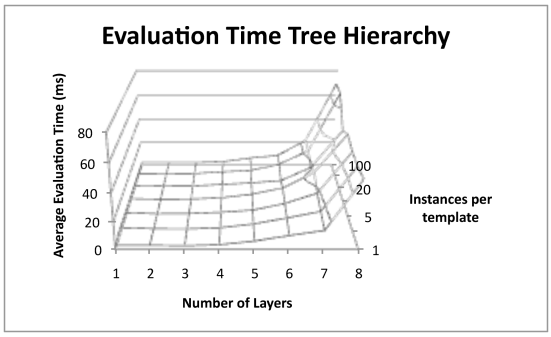
The results of the experiments, reported in Figure 3a, show that, for the linear hierarchy, the average evaluation time is always below $20ms$, independently from the percentage of the intersecting locations. As expected, response time slightly increases as the number of locations increases. Fluctuations are



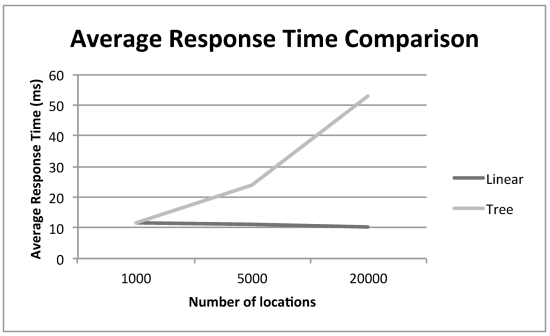
(a) Response Time for Linear hierarchy without events.



(b) Response Time for Linear hierarchy with events.



(c) Response time for quadtree hierarchy.



(d) Comparison between linear and tree hierarchies.

Fig. 3. Performance evaluation.

mainly due to the reasonable hypothesis that access requests are generated by random users at random places.

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-shaped 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 2b 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 3c 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 3d reports the comparison 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.

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. Furthermore, to each role we have associated a single activation rule. This last choice was made in order to measure only the impact of event management by avoiding any possible complication related to rule overriding. We have run several experiment

sets by considering each time a different number of logical locations containing the request location. 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. We report in Figure 3b the average evaluation time only for a linear hierarchy with 50% of intersecting locations. Results for different percentages of intersecting locations are similar and thus omitted. As already anticipated in Section 8.3, the impact of event management in access control decision making is negligible. The results of the experiments are summarized in Table 7.

Table 7
Summary of Experiments

Experiment Name			
Fixed Params	Variables	Av. Eval. Time	Fig. #
Baseline			
#Locations=1	#Role templates (10-100K)	$< 50ms$ (20K templ.)	Fig 1
#Events=0	#Role instances (100-100K)	$< 300ms$ (100K templ)	
Linear Hierachy - No Events			
#Role templates=10K	#Locations	$<$ 20 ms	Fig 3a
#Events=0	#Insersecting locations		
Linear Hierachy - With Active Events			
#Role temp.=10K	#Locations	$<$ 20 ms	Fig 3b
#Intersect. loc. 50%	#Active events		
Quad-tree Hierachy - No Events			
#Events=0	#Layers	$<$ 80 ms	Fig 3c

9. Conclusions and future work

Summarizing, we designed and implemented an expressive, reactive policy framework. The underlying policy model, ERBAC, collects a number of features separately supported by previous models: It is event-driven, and supports spatiotemporal conditions and role templates (the latter are particularly important in e-health and hospital scenarios). Moreover, ERBAC introduces the notions of personalized and localized events that are useful in defining emergency handling policies. The policy model is equipped with formal syntax and semantics. Periodic time conditions are encoded as in TRBAC, while locations are classified along three dimensions: physical, logical, and the less common location types, that make it possible to associate a policy to all the locations with the same functionality (e.g. surgery room, or meeting room).

The expressiveness of the policy language has been tested through a variety of examples, to show its potential both in a hospital domain and, more generally, as a general purpose reactive policy framework. It turned out that ERBAC can naturally handle emergency-handling policies, by adapting and possibly

relaxing access control constraints 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. In this respect, our framework generalizes previous approaches, where emergency handling has been almost exclusively limited to handling application-generated exceptions, and hardware and communication failure [23] (that can be handled like other events in our model). Medical emergencies are handled in [24] through a fixed hierarchic procedures organized in three emergency levels; our policy language makes it possible to define more flexible emergency handling procedures.

ERBAC has been merged into XACML for two purposes: (i) supporting complex policy structuring and modularization through the policy composition constructs of XACML's language; we stress that almost all the theoretical frameworks are "flat", i.e. they do not support any policy structuring/composition constructs; (ii) testing the policy model in a standard architecture of industrial strength. These goals required extending XACML's syntax and constructing a suitably enriched policy decision point (PDP), as explained in Section 8. As a result, we showed that a scalable working system can be obtained through a suitable integration of standard components and focussed specialized components, in a standard-preserving way.

The implementation made it possible to experimentally evaluate the performance of the policy language, and in particular the cost of evaluating complex conditions. We relied upon previous work on XACML to assess the feasibility of composition constructs (execution time exhibits a linear behavior in the worst case; careful policy structuring, through the TARGET tags, may further reduce the overhead). Then we focussed on spatiotemporal conditions. Temporal conditions can be reduced to atomic event handling. All the complexity is absorbed in the (static) pre-compilation of a cron agenda, and has virtually no impact at run time. So we focussed our scalability tests on spatial conditions.

Our 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.

Currently, ERBAC does not support event composition operators, such as sequence and co-occurrence. Such operators in principle make sense in pervasive computing environments and are an interesting subject for future extensions, although in our examples we felt no need for them, perhaps because event conditions are hosted by a prioritized policy rule language rich enough to simulate some event correlation conditions.

We conclude that the framework is a good starting point for designing a general purpose, reactive policy framework, because of its expressiveness and performance. We stress that this evidence could not have been gathered without carrying out the full design process, comprising the rich policy model selection, its assessment, implementation, and testing. The current system is ready for experimental deployment in realistic ubiquitous computing environments. 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.
[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] S. Chandran and J. Joshi. LoT-RBAC: A location and time-based RBAC model. In A. Ngu, M. Kitsuregawa, E. Neuhold, J.-Y. Chung, and Q. Sheng, editors, *Web Information Systems Engineeringâ WISE 2005*, volume 3806 of *Lecture Notes in Computer Science*, pages 361–375. Springer Berlin Heidelberg, 2005.
[12] L. Chen and J. Crampton. On spatio-temporal constraints and inheritance in role-based access control. In *Proceedings of the 2008 ACM symposium on Information, computer and communications security*, ASIACCS '08, pages 205–216, New York, NY, USA, 2008. ACM.
[13] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms (3. ed.)*. MIT Press, 2009.
[14] 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.
[15] 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.
[16] M. L. Damiani, E. Bertino, B. Catania, and P. Perlasca. GEO-RBAC: A spatially aware RBAC. *ACM Trans. Inf. Syst. Secur.*, 10(1), 2007.
[17] 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.
[18] C. K. Georgiadis, I. Mavridis, G. Pangalos, and R. K. Thomas. Flexible team-based access control using contexts. In *SACMAT*, pages 21–27, 2001.
[19] 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.
[20] J. Joshi, E. Bertino, and A. Ghafoor. An analysis of expressiveness and design issues for the generalized temporal role-based access control model. *IEEE Trans. Dependable Sec. Comput.*, 2(2):157–175, 2005.
[21] 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.
[22] 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.
[23] D. Kulkarni and A. R. Tripathi. A framework for programming robust context-aware applications. *IEEE Trans. Software Eng.*, 36(2):184–197, 2010.
[24] 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.
[25] 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.
[26] 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.
[27] OASIS Consortium. extensible access control markup language (XACML), v. 2.0.
[28] OpenGIS Consortium. Geography Markup Language (GML) simple features profile. http://www.opengeospatial.org/standards/gml.
[29] OpenGIS Consortium. Geospatial eXtensible Access Control Markup Language (GeoXACML) v 1.0. http://www.opengeospatial.org/standards/geoxacml.

[30] OpenGIS Consortium. GeoXACML Implementation Specification - Extension B (GML3) Encoding. http://www.opengeospatial.org/standards/gml.
[31] OpenGIS Consortium. Implementation Standard for Geographic information - Simple feature access - Part 1: Common architecture. http://www.opengeospatial.org/standards/sfa.
[32] S. Osborne, editor. *Fifth ACM Workshop on Role-Based Access Control*, Berlin, 2000. ACM.
[33] J. Park and R. S. Sandhu. The UCON_{ABC} usage control model. *ACM Trans. Inf. Syst. Secur.*, 7(1):128–174, 2004.
[34] 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.
[35] 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.
[36] 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.
[37] 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.
[38] 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.
[39] R. S. Sandhu, editor. *Second ACM Workshop on Role-Based Access Control*, Fairfax, VA., 1997. ACM.
[40] R. S. Sandhu, editor. *Third ACM Workshop on Role-Based Access Control*, Fairfax, VA., 1998. ACM.
[41] 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.
[42] V. Solutions. JTS topology suite. http://www.vividsolutions.com/jts/jtshome.htm.
[43] 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.
[44] Sun Microsystems. Sun's xacml implementation. http://sunxacml.sourceforge.net.
[45] 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.

Appendix

Table 8
Glossary

Function Name/Symbol	Meaning	Defined in
PL	Set of Physical Locations	Sec. 4.1.2
LL	Set of Logical Locations	Sec. 4.1.2
LT	Set of Location Types	Sec. 4.1.2
$\subset_{LL}$	Partial order over logical locations	Sec. 4.1.2
$\preceq_{LT}$	Partial order over location types	Def. 6
$TCond$	Temporal Condition	Def. 10
$SCond$	Spatial Condition	Def. 11
$ECond$	Event Condition	Def. 12
$Eloc : E \to PL \cup \{\bot\}$	Physical location in which an event is generated	Def. 9
$Eprios : E \to Prios$	Priority of an event	Def. 9
$EV(t)$	Set of prioritized event expressions at time t.	Def. 25
$EVisible : E \to LL \cup \{\bot\}$	Set of logical loc. in which an event is visible	Def. 9
$generatedBy : E \to U \cup \{\bot\}$	User who generated an event	Def. 9
$generatedFor : E \to 2^U \cup \{\bot\}$	Users to whom an event is addressed	Def. 9
$PtoL : PL \to 2^{LL} \cup \{\bot\}$	Maps physical locations to logical ones	Def. 5
$LofT : LT \to 2^{LL} \cup \{\bot\}$	Maps locations type to logical locations	Def. 5
$LtoT : LL \to 2^{LT}$	Maps logical locations to location types	Def. 5
$MostSpecific(S)$	Event expressions in S that are minimal w.r.t. $<$	Def. 24.
$NonBlocked(S)$	Mutually compatible event expressions in S	Def. 21.
$Q(t)$	Subset of REB whose conditions are met at t	Def. 26
$SEXP$	Simple Event Exp.: “enable/disable r [for u]”	Def. 16
$PEXP$	Priorit. Event Exp.: “p:enable/disable r [for u]”	Def. 16
$CEXP$	Conditional event expression $\langle cond, ev\rangle$	Def. 18
$<$	Partial order defined over CEXP	Def. 23
$conf(x)$	Denotes the SEXP conflicting with x	Def. 17
$Sol(I, P)$	Set of intervals identified by the exp. (I, P)	Def. 3
$ST(t)$	Set of roles that are globally enabled at time t	Def. 25
$time()$	Current time	Def. 3
$UserPos : U \to PL$	The physical locations of a device	Def. 5