A Model-driven Approach to Representing and Checking RBAC Contextual Policies

Ameni Ben Fadhel, Domenico Bianculli, Lionel Briand
SnT Centre - University of Luxembourg, Luxembourg
{ameni.benfadhel,domenico.bianculli,lionel.briand}@uni.lu

Benjamin Hourte
HITEC Luxembourg
benjamin.hourte@hitec.lu

ABSTRACT

Among the various types of Role-based access control (RBAC) policies proposed in the literature, contextual policies take into account the user's location and the time at which she requests an access. The precise characterization of the context in such policies and the definition of an access decision procedure for them are non-trivial tasks, since they have to take into account the various facets of the temporal and spatial expressions occurring in these policies. Existing approaches for modeling contextual policies do not support all the various spatio-temporal concepts and often do not provide an access decision procedure.

In this paper, we propose a model-driven approach to representing and checking RBAC contextual policies. We introduce GemRBAC+CTX, an extension of a generalized conceptual model for RBAC, which contains all the concepts required to model contextual policies. We formalize these policies as constraints, using the Object Constraint Language (OCL), on the GemRBAC+CTX model, as a way to operationalize the access decision for user's requests using model-driven technologies. We show the application of GemRBAC+CTX to model the RBAC contextual policies of an application developed by HITEC Luxembourg, a provider of situational-aware information management systems for emergency scenarios. The use of GemRBAC+CTX has allowed the engineers of HITEC to define several new types of contextual policies, with a fine-grained, precise description of contexts. The preliminary experimental results show the feasibility of applying our model-driven approach for making access decisions in real systems.

1. INTRODUCTION

Several types of Role-based access control (RBAC) policies[1] have been proposed in the literature, together with the corresponding conceptual models that support them (see, for example, the recent taxonomy in [5]). In this paper we focus on *contextual policies*. A contextual policy restricts a user to perform an action depending on her context, e.g., her location and/or the time at which the action should happen. For example, a policy that refers to a temporal context (also called *temporal policy*) can be expressed in English as "assign role *program chair* to user *AP* from March 4, 2015 to March 11, 2016". Similarly, a policy that refers to a spatial context (also called *spatial policy*) can be expressed as "assign role *general chair* to user *RS* if he is located in San Antonio, TX".

Contextual policies play a fundamental role in defining the security level of a system in different types of application domains. Consider, for example, the case of proximity-based payment systems, in which a user can access her credit card details stored on a mobile device only in proximity of a compatible point-of-sale. Another example is represented by enterprise policies that restrict the hours in which an employee can connect to the corporate network when working from home. Finally, we remark that this type of policies is vital in the domain of disaster relief intervention, where HITEC Luxembourg (the partner for the research project in which this work has been carried out) is a provider of situational-aware information management systems for emergency scenarios. In such systems, restricting access to resources (e.g., satellite photos, sensors data) based on the user context is an essential and critical requirement.

Precisely characterizing the context in a contextual policy is non-trivial. For example, temporal policies can refer to individual time instants (e.g., a specific date and/or time) or to time intervals. They can also contain periodic expressions (e.g., "every 3 months") or complex expressions like "in February, from the second Monday to third Friday, from 10:00 to 12:00". In the case of spatial policies, the context can be expressed, for instance, using a distance and a direction with respect to another location (e.g., "6 miles West from coordinates 48.86N, 2.29E") or just with a qualitative attribute (e.g., "100 meters outside the White House"). All these facets have to be considered when defining an RBAC model that supports contextual policies. On a par with the problem of expressing and modeling these policies, there is also the issue of how to make an access decision (i.e., granting/denying access requests) based on such policies.

Several RBAC models have been proposed in the literature to support contextual policies. However, none of them fully support all the facets of these policies. Moreover, only some of them provide algorithms to evaluate contextual policies in order to make an access decision. Furthermore, these

[1]RBAC policies are also referred to as "(authorization) constraints". In this paper we will use the word "policies" to avoid the confusion with "(OCL) constraints".

CODASPY'16, March 9–11, 2016, New Orleans, LA, USA.

DOI: http://dx.doi.org/10.1145/2857705.2857709

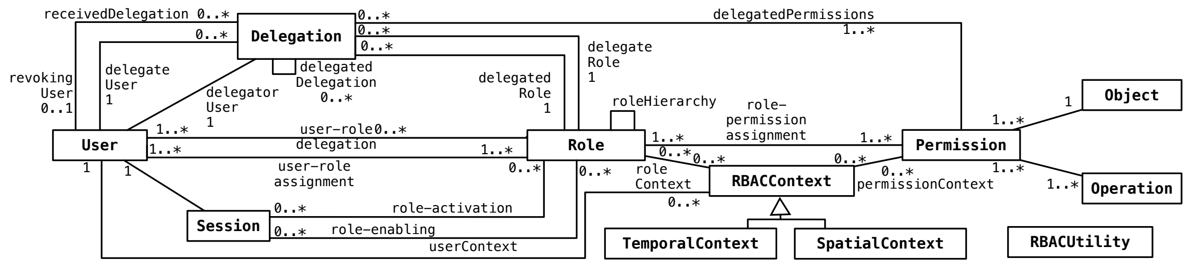

Figure 1: A simplified version of the GemRBAC conceptual model.

models are based on the original RBAC model [23] and do not support advanced non-contextual policies like binding of duty, delegation, revocation.

Our goal is to define a conceptual model significantly more expressive than the state of the art, on top of which we can operationalize the access decision procedure. A more expressive and operational model critically determines its applicability in real scenarios. To achieve this goal, we follow a model-driven approach, based on UML and the Object Constraint Language (OCL) [17].

To represent RBAC contextual policies, we present GemRBAC+CTX, a conceptual model—expressed in UML—that contains all the conceptual entities required to accurately specify temporal and spatial contexts in RBAC policies. GemRBAC+CTX is defined as an extension of GemRBAC [5], our previous proposal for a generalized framework for defining RBAC policies. Since GemRBAC+CTX is an extension of GemRBAC, it inherits all its benefits, in particular the support for all types of (non-contextual) RBAC policies surveyed in [5]. In this way, GemRBAC+CTX supports both complex contextual policies and all types of non-contextual policies (including binding of duty, delegation, revocation).

Regarding the operationalization of the access decision procedure for contextual policies, our model-driven approach is based on the formalization of contextual policies as OCL constraints on the GemRBAC+CTX model. The problem of making an access decision for contextual policies can be thus reduced to checking the corresponding OCL constraints on an instance of the GemRBAC+CTX model. We use OCL since it is the common, standardized language for expressing constraints in model-driven engineering and it is well-supported by industry-strength tools. The proposed OCL-based formalization can facilitate the precise understanding of contextual policies by practitioners and paves the way for the practical verification of these policies, based on UML modeling tools and OCL checkers (such as Eclipse OCL [11]).

Furthermore, we report on the application of the proposed approach to modeling the RBAC contextual policies of a real system. The use of GemRBAC+CTX has allowed the engineers of HITEC to define *19 new types* of contextual policies, with a fine-grained, precise description of contexts. Based on the results of the three policies in the case study section, the time taken by our model-driven approach for making an access decision ranges from few milliseconds to less than three seconds per policy, confirming its suitability for the practical operationalization of access decision procedures.

To summarize, the specific contributions of the paper are: 1) the GemRBAC+CTX conceptual model, to express contextual RBAC policies; 2) the templates for the formalization of OCL constraints over the GemRBAC+CTX model, as a way to operationalize the access decision of contextual RBAC policies; 3) the application of the GemRBAC+CTX model for the specification of real RBAC policies in an industrial setting with high contextual requirements.

The paper is structured as follows. Section 2 briefly illustrates the GemRBAC model. Section 3 shows how to model contextual policies with GemRBAC+CTX. Section 4 illustrates the templates for the formalization of contextual policies as OCL constraints on the GemRBAC+CTX model. Section 5 reports on an industrial application of the proposed approach. Section 6 discusses related work and Section 7 concludes the paper.

2. BACKGROUND: THE GEMRBAC MODEL

The GemRBAC model, previously introduced in [5], is a richer and more expressive extension of the original RBAC model [23]. GemRBAC has been designed after surveying the various types of RBAC policies (and the corresponding model extensions) proposed in the literature. We defined GemRBAC with the goal of filling the gap among these extensions by proposing a generalized model that includes *all* the conceptual entities required to define all the types of constraints classified in the survey. In the rest of this section we describe the main entities of the GemRBAC model providing some background information on RBAC. A simplified version of the GemRBAC model is shown in the class diagram of Figure 1; we refer the reader to [5] for a complete description.

At the basis of GemRBAC there are the concepts of `User`, `Session`, `Role` and `Permission`. A `Permission` is represented as a set of `Operations` on an `Object`. `Permissions` are assigned to `Roles` and `Roles` are assigned to `Users`; these assignments are captured with associations between the respective classes. A `Session` maps a `User` to a subset of the `Roles` that have been assigned to her; this mapping activates the role(s) for a certain user. A role that can be activated is called *enabled*. Role enabling and activation relations are modeled as associations between the `Role` and `Session` classes. A permission is enabled if the user is allowed to perform its associated operations. By analogy to role assignment and enabling, we model the enabled permissions of a given role with the `enabledPermissions` association between the `Role` and `Permission` classes. A user can delegate her role or a subset of its permissions to another user via a `Delegation`.

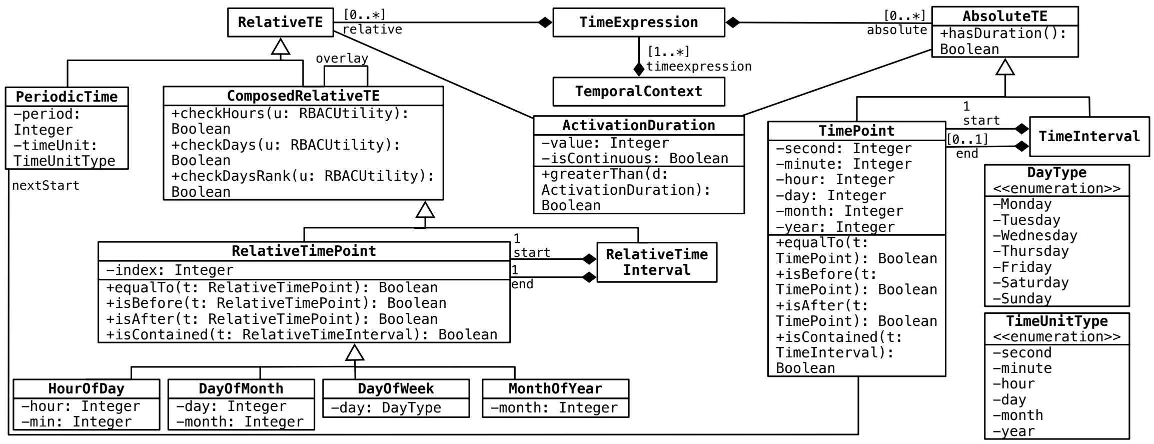

Figure 2: Temporal context in GemRBAC+CTX.

For a `Delegation` we keep track of the delegator, the delegate (the user that receives the delegation), their roles at the time of the delegation, and the delegated role, with associations between classes `Delegation` and `User`, and classes `Delegation` and `Role`. A delegation is put to an end through a revocation action, which can be explicitly performed by a user or automatically triggered depending on the context. Contextual information is modeled with the class `RBACContext` and its subclasses `TemporalContext` and `SpatialContext`. A temporal (respectively, spatial) context models the time (location) on which a given role, or permission, can be enabled/assigned; a role or permission can be enabled/assigned if its contextual information matches the user's one.

3. MODELING CONTEXTUAL POLICIES WITH GEMRBAC+CTX

The GemRBAC model has limited support for contextual policies. More specifically: (1) the context assigned to a role (or a permission) restricts either its assignment or its enabling but not both; (2) temporal and spatial information are represented in a symbolic way, without an explicit characterization of the actual context. Limitation (1) implies that two policies such as "assign role r to user u in context ctx_1" and "enable role r in context ctx_2", which respectively restricts the assignment and the enabling of the same role r, cannot be defined on the same model instance. As for limitation (2), the GemRBAC model cannot be used to model explicitly the specific aspects of temporal and spatial context, because GemRBAC represents contexts in a symbolic way (i.e., with identifiers). For example, from the point of view of temporal context, one cannot explicitly refer to time instants (e.g., "(on) *January 21, 2014 at 8:00*") or periodic expressions (e.g., "*every Monday, from 9.00 to 11.00*"). From the point of view of spatial context, in GemRBAC, for instance, one cannot define a geo-fence, i.e., a precise geometric characterization of a context (e.g., "*within a radius of 20 miles from the main building*") or a relative location (e.g., "*100 meters outside the White House*").

To overcome these limitations, we introduce the GemRBAC+CTX model, an extension of the GemRBAC model that supports the definition of richer contextual policies. To address limitation (1), in the GemRBAC+CTX model we separate contextual assignment and enabling, both for role and permission. More explicitly, the context in which a role should be assigned (as prescribed by a contextual policy) is modeled with the `roleContextAssignment` association between the `RBACContext` and `Role` classes; similarly, the context in which a role should be enabled (as prescribed by a contextual policy) is modeled with the `roleContextEnabling` association between these two classes. The context for permission enabling and assignment is modeled in a similar way with the `permissionContextAssignment` and `permissionContext-Enabling` associations between the `RBACContext` and `Permission` classes. To tackle limitation (2), we enrich the GemRBAC model with new entities that support the specification of more detailed temporal and spatial context in policies. We illustrate these new entities in the rest of this section.

3.1 Modeling Temporal Context

We support richer temporal context specification in the GemRBAC+CTX model by introducing a new class hierarchy under class `TemporalContext` of the GemRBAC model. The new classes and their associations are shown in the class diagram in Figure 2.

We extend class `TemporalContext` by introducing (with a composition relation) the concept of `TimeExpression`. A time expression is composed of absolute and/or relative time expressions; these concepts are modeled as classes `AbsoluteTE` and `RelativeTE`.

An absolute time expression refers to a concrete point or interval in the timeline. An absolute time point, modeled with the class `TimePoint`, corresponds to a given time instant, e.g., "*January 21, 2014 at 8:00:00*". Hereafter, to improve the readability we will omit the hours from a time point when we refer to midnight. A time interval, modeled with class `TimeInterval`, corresponds to a segment in the timeline; a time interval can be either of type bounded or unbounded. A bounded time interval corresponds, for example, to the expression "*from January 21, 2014 to April 25, 2015*". This interval has a start `TimePoint` (*January 21, 2014*) and an end `TimePoint` (*April 25, 2015*). An unbounded interval corresponds to the expression "*starting from October 15, 2013*"; it has only the start `TimePoint` (*October 15, 2013*) and is unbounded to the right.

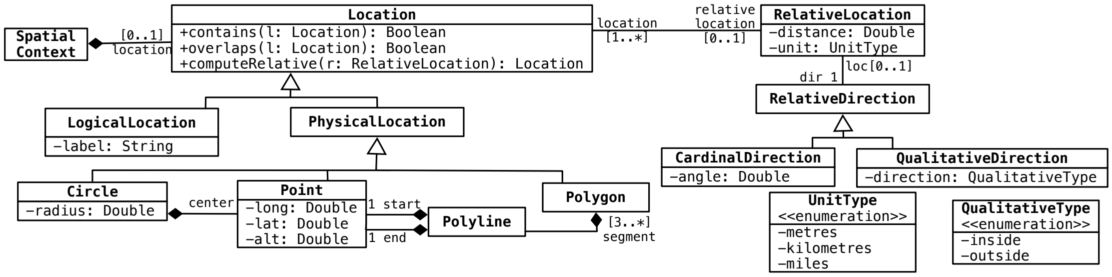

Figure 3: Spatial context in GemRBAC+CTX.

A relative time expression is an expression that cannot be mapped *directly* to a point or an interval in the timeline. For example, the common expression "*(at) 9 a.m.*" by itself cannot be directly mapped to a point in the timeline unless another expression, e.g., "*(on) May 2, 2015*" is specified. Class `RelativeTE` has two subclasses, `RelativeTime` and `PeriodicTime`. By analogy with the class `AbsoluteTE`, the class `RelativeTime` has two subclasses, `RelativeTimePoint` and `RelativeTimeInterval`. Class `RelativeTimePoint` has four subclasses: `HourOfDay` refers to a specific hour of the day, e.g., "*(at) 9 a.m.*"; `DayOfWeek` corresponds to a given day of the week, e.g.,"*(on) Monday*" refers to any Monday; `DayOfMonth` refers to a day in a month such as "*(on) April, 5*"; `MonthOfYear` refers to a given month, e.g., "*(in) April*". Unlike class `TimeInterval`, class `RelativeTimeInterval` always refers to a bounded time interval, whose start and end points have both the same type (a subclass of `RelativeTimePoint`).

Class `ComposedRelativeTE` can be recursively composed with itself through the association `overlay`, to represent composite time expressions. These composite expressions are required to have composite elements of different granularity. We enforce this requirement by defining a structural constraint on the model. Informally, a `MonthOfYear` can overlay either a `DayOfWeek` or an `HourOfDay`; a `DayOfWeek` or a `DayOfMonth` can overlay only an `HourOfDay`. The same constraint applies if any subclass c of `RelativeTimePoint` mentioned in it is replaced with a `RelativeTimeInterval` with bounds of type c. An example of an expression that can be modeled by composing different instances of `ComposedRelativeTE` by means of the `overlay` association is "*in February, from the second Monday to third Friday, from 10:00:00 to 12:00:00*". This expression is modeled by an instance of `MonthOfYear` (*February*) overlaid with an instance of `RelativeTimeInterval`, with start- and end-point of type `DayOfWeek` (*from Monday to Friday*), overlaid with an instance of `RelativeTimeInterval`, with start- and end-point of type `HourOfDay` (*from 10:00:00 to 12:00:00*). The indexes that refer to a specific occurrence of Monday and Friday are modeled with the `index` attribute, which is defined only for class `DayOfWeek`.

Class `RelativeTE` can also represent periodicity in temporal expressions such as "*every 3 months*". The periodicity is modeled with its subclass `PeriodicTime`. Its attribute `period` is a numeric value associated with a time unit (e.g., day, hour, month) modeled with the attribute `timeUnit`. A `PeriodicTime` is always part of a `TimeExpression` that has exactly one `AbsoluteTE`; the latter defines either the starting time of the period (as in "*every 3 months, starting from April 5, 2015*") or the time interval in which it applies (as in "*every 3 months, from April 5, 2015 to June 8, 2017*"). We assume that each `PeriodicTime` has a `nextStart` association with a `TimePoint` corresponding to the beginning of the next period.

In the context of RBAC, a temporal context can have a time-based policy that represents a bound for the sum of activation durations of a given role (or permission). For instance, a security engineer could enable a certain role from Monday to Friday but allow users to activate it only for two hours over the five days. We keep track of this duration with class `ActivationDuration`, which is associated with classes `RelativeTE` and `AbsoluteTE`. Moreover, this duration can be cumulative (i.e., related to multiple sessions) or non-cumulative (i.e., related to the current session); this concept is represented by the boolean attribute `isContinuous` of the class `ActivationDuration`.

3.2 Modeling Spatial Context

Similarly to what we have done for temporal context, we support richer spatial context specification in the GemRBAC+CTX model by introducing a new class hierarchy under class `SpatialContext` of the GemRBAC model. The new classes and their associations are shown in the class diagram in Figure 3.

We extend class `SpatialContext` by introducing (with a composition relation) the concept of `Location`. At a very high-level, a location represents a specific bounded area or point in space. A location can be either physical or logical; these concepts are modeled as classes `PhysicalLocation` and `LogicalLocation`.

A physical location identifies a precise position in a geometric space. We consider three possible ways to express a physical location and we model them as subclasses of `PhysicalLocation`. Class `Point` represents a geographic coordinate with latitude, longitude and altitude. Class `Circle` represents a circular area, characterized by a radius and a center. Class `Polygon` is an area enclosed by at least three segments, which are modeled with class `Polyline`; each `Polyline` is a segment composed of a start and an end `Point`. Notice that a `Polygon` (as a set of `Polylines`) can model areas with complex shapes, such as the border of a city.

A logical location is an abstraction of one or many physical locations. For instance, the logical location "*offices on the second floor*" refers to the set of physical locations corresponding to the actual office rooms in the second floor of a building. A logical location can also be a convenient shorthand to identify a geographical landmark without providing its coordinates. The concept of logical location is modeled with class `LogicalLocation`. We assume that there is a geocoding function that maps each `LogicalLocation` to the corresponding `PhysicalLocation`(s). A location can be defined *relatively* to another location by providing a direction and optionally a distance. We model these concepts with class `RelativeLocation`, which is associated with class `RelativeDirection`, and has a `distance` attribute. The latter has two subclasses, `CardinalDirection` and `QualitativeDirection`. Class `CardinalDirection` represents the degrees of rotation based on cardinal points on a compass. An example of a location using a relative location denoted with a cardinal direction is "*6 miles West from the Tour Eiffel*". This expression contains a distance (*6 miles*), a cardinal direction (*Southwest, i.e., 225°*) and a logical location (*Tour Eiffel*). Class `QualitativeDirection` represents a relative proximity to a location, such as "*inside*" or "*outside*". An example of a location using a relative location denoted with a qualitative direction is "*100 meters outside the White House*". This expression contains a distance (*100 meters*), a qualitative direction (*outside*) and a logical location (*White House*).

Class `Location` provides some operations that check for topological relations between locations: operation `contains` checks if a location is a part of another one; operation `overlaps` checks if two locations share a common area.

In the GemRBAC+CTX model, we model the user's position with an association between classes `User` and `SpatialContext`. This association provides more precise information than the association between `User` and `RBACContext` that was included in the GemRBAC model (and that is not present in the GemRBAC+CTX model anymore).

4. CHECKING CONTEXTUAL POLICIES WITH OCL

The GemRBAC+CTX model can be used to represent the state of the system from the point of view of RBAC. As explained in Section 3, the context in which a `Role` (or a `Permission`) can be enabled or assigned (as prescribed by a contextual policy), is captured on the UML model. RBAC contextual policies can then be checked by verifying OCL constraints on the GemRBAC+CTX model. In this way, an access decision (e.g., allowing a user to activate a role) can be performed by checking whether an instance of the GemRBAC+CTX satisfies the OCL constraints associated with it. In the rest of this section we provide several templates that can be used to formalize contextual RBAC policies for role enabling or assignment as OCL constraints on the GemRBAC+CTX model. These RBAC policies are based on real policies defined in our industrial case study.

In the definition of the OCL constraints, we make some working assumptions. We assume that each snapshot contains the time at which it was taken (modeled as an association between classes `RBACUtility` and `TimePoint`) and the current day of week (modeled as an association between classes `RBACUtility` and `DayOfWeek`). This assumption can be guaranteed by applying a timestamp to each snapshot. We also assume that the position of the user is always known, by means of a GPS; this is very reasonable nowadays. Lastly, we assume that policies are not conflicting with each other; e.g., we avoid the case of having two policies, one enabling (assigning) and another one disabling (unassigning) the same role/permision in the same context; consistency check of RBAC policies (which guarantees conflict-free policies) is outside the scope of the paper.

All OCL constraints have been made publicly available, together with an Ecore version of the GemRBAC+CTX model, at
https://github.com/AmeniBF/GemRBAC-CTX-model.git.

4.1 Policies with temporal context

A policy on role enabling with an absolute time expression restricts the time interval at which a role can be enabled, as in "role r_1 is enabled from January 21, 2014 to April 25, 2015". This policy can be checked by verifying the following OCL invariant of the class `Session`:

```
1  context Session inv AbsoluteBTIRoleEnab:
2  let u : RBACUtility = RBACUtility.allInstances(),
3      r : Role = Role.allInstances() ->
4         select(r : Role|r.idRole = 'r1'),
5      temporalContext: Set(RBACContext) =
6         r.roleContextEnabling -> select(c |
7         c.oclIsTypeOf(TemporalContext)),
8      timeE: Set(AbsoluteTE) = temporalContext.
9         oclAsType(TemporalContext).timeexpression.
10        absolute ->flatten()->asSet(),
11     timeI: Set(AbsoluteTE) = timeE -> select(e |
12        e.oclIsTypeOf(TimeInterval) and
13        e.oclAsType(TimeInterval).end->notEmpty())
14 in if timeI.oclAsType(TimeInterval) ->
15    exists(i| u.getCurrentTime().isContained(i)) then
16             self.enabledRoles -> includes(r)
17             or self.activeRoles -> includes(r)
18    endif
```

In this OCL expression, we first select the instance corresponding to role r_1 (lines 3–4). Then, we retrieve the list `temporalContext` of temporal contexts in which the role should be enabled (lines 5–7) and compute, over the elements of this list, the list `timeE` of absolute expressions assigned to them (lines 8–10). In this example, since there is only one `TemporalContext` object containing one `AbsoluteTE` object, the `timeE` list will include only one instance of `TimeInterval` whose start and end `TimePoints` corresponds to "January 21, 2014" and "April 25, 2015". Since the enabling temporal context in the policy is expressed as a bounded time interval, we have to select, among the elements of `timeE`, the list `timeI` of expressions in the form of a time interval (lines 11–13) with a bounded end point; this last condition is checked with the expression at line 13. Afterwards, we check if the time when the snapshot was taken—obtained by calling the operation `getCurrentTime` of class `RBACUtiliy`—is contained in one of the time intervals in list `timeI` (lines 14–15). If this is the case, we check whether role r_1 is in the list of enabled or active role of the current session (lines 16–17).

A policy on permission assignment with a relative time expression restricts the time at which a permission can be assigned to a role. As explained in Section 3, we support different forms of relative time expression. For the purpose of illustration, we consider a relative time expression structured as a `DayOfWeek` (or a `RelativeTimeInterval` with bounds of type `DayOfWeek`), which, subsequently, can overlay an `Hour` (or a `RelativeTimeInterval` with bounds of type `Hour`). An example of a policy with a relative time expression of this form is "assign role r_1 to user u_1 *only* from Wednesday to Friday, from 10:00 to 14:00". Such a policy can be checked by verifying the following OCL invariant of the class `Permission`:

```
1 context Permission inv DayOfWeekHourPermAssign:
2 if self.idPermission = 'p1' then
3  let u: RBACUtility = RBACUtility.allInstances(),
4    day: RelativeTimePoint = u.getDayOfWeek(),
5    r: Role = Role.allInstances() ->
6        select(r : Role | r.idRole = 'r1'),
7    temporalContext: Set(RBACContext) = self.
8        permissionContextAssignment -> select(c |
9        c.oclIsTypeOf(TemporalContext)),
10   timeE: Set (ComposedRelativeTE) = temporalContext.
11       oclAsType(TemporalContext).timeexpression.
12       relative.oclAsType(ComposedRelativeTE)
13       -> flatten() -> asSet(),
14   days: Set (ComposedRelativeTE) = timeE ->select(t|
15      (t.oclIsTypeOf(RelativeTimeInterval) and
16       t.oclAsType(RelativeTimeInterval).start.
17       oclIsTypeOf(DayOfWeek) and day.isContained
18      (t.oclAsType(RelativeTimeInterval)))
19      or (t.oclIsTypeOf(DayOfWeek) and
20      day.equalTo(t.oclAsType(DayOfWeek))))
21  in if days -> exists (t| t.checkHours(u)) then
22            self.roles -> includes (r)
23      else
24            self.roles -> excludes (r)
25      endif
26 endif
```

In this OCL expression if the current permission is p_1, we select the day corresponding to the day of week at which the snapshot was taken, by calling the `getDayOfWeek` operation of the class `RBACUtility` (lines 3–4). Then, we select the instance corresponding to role r_1 (lines 5–6). We retrieve the list of temporal contexts `temporalContext` in which the permission should be assigned to role r_1 (lines 7–9) and compute, over the elements of this list, the list `timeE` of relative time expressions assigned to them (lines 10–13). Based on the type of policy described above, we have to select, among the elements of `timeE`, the list `days` of relative time expressions having a `ComposedRelativeTE` of type `DayOfWeek` or of type `RelativeTimeInterval` with bounds of type `DayOfWeek` (lines 14–20). While selecting the time expressions in this list, we check whether the day at which the snapshot was taken is contained in the selected `TimeExpression`. To do so, we check separately for the `DayOfWeek`, by calling operation `equalTo` of class `RelativeTimePoint` (line 20), and for the `RelativeTimeInterval` by calling operation `isContained` of class `RelativeTimePoint` (line 17). In this specific example, list `days` will include a `TimeExpression` that contains two `ComposedRelativeTE`. These objects are: a `RelativeTimeInterval` (whose start and end `RelativeTimePoints` correspond to "Wednesday" and "Friday"); and a `RelativeTimeInterval` (whose start and end `RelativeTimePoints` correspond to "10:00" and "14:00"). We remark that the first object `overlays` the second. We check whether the time at which the snapshot was taken is contained in one of the `TimeExpressions` in `days`. To do so, we check the hours overlaid by the day(s) of the week by calling operation `checkHours` of class `ComposedRelativeTE` (line 21). If the check succeeds, we require role r_1 to belong to the list of roles of permission p_1 (line 22). Otherwise, we require the role not to be in this list (line 24). Because of space limitations, in the remaining of this section we focus only on the specification of policies at the role level.

A policy on role assignment with a relative time expression containing an index of a specific `DayOfWeek` restricts the day in which a given user can acquire a given role, as in "*assign role r_1 to user u_1 on the 2nd Monday of June*". This policy can be checked by verifying an OCL invariant of the class `Role`:

```
1 context Role inv indexRoleAssign:
2 let u: RBACUtility = RBACUtility.allInstances(),
3    month: ecore::EInt = u.getCurrentTime().month,
4    day: RelativeTimePoint =  u.getDayOfWeek(),
5    u1: User = User.allInstances() ->
6        select(m : User | m.idUser = 'u1'),
7    temporalContext: Set(RBACContext) = self.
8        roleContextAssignment -> select(c |
9        c.oclIsTypeOf(TemporalContext)),
10   timeE: Set(ComposedRelativeTE) = temporalContext.
11       oclAsType(TemporalContext).timeexpression.
12       relative.oclAsType(ComposedRelativeTE)
13       -> flatten() -> asSet()
14 in self.idRole = 'r1'
15    and self.users -> includes (u1) implies
16    timeE -> exists(t | t.oclIsTypeOf(MonthOfYear)
17    and t.oclAsType(MonthOfYear).month  = month
18    and t.checkDayIndex(u))
```

In this invariant we first select the month and day of week at which the snapshot was taken by calling the `getCurrentTime` and `getDayOfWeek` operations of class `RBACUtility` (lines 3–4). Then, we select the instance corresponding to user u_1 (line 6). We retrieve the list `temporalContext` of temporal contexts in which the role should be assigned (lines 7–9) and compute, over the elements of this list, the list `timeE` of time expressions assigned to them (lines 10–13). The implication at lines 14–18 states that if the current role is r_1 and user u_1 is a member of this role, the temporal context for role assignment should match the current `DayOfWeek`; this condition is verified by calling operation `checkDayIndex` of class `ComposedRelativeTE`.

A policy on role assignment with time expression containing a periodic expression restricts the time at which a role can be assigned to a user as in "user u_1 acquires role r_1 every 5 days starting from July 10, 2014 at 16:00". This policy can be checked in OCL as an invariant of class `Role`:

```
1 context Role inv periodicUnboundTIRoleAssign:
2 let u: RBACUtility = RBACUtility.allInstances(),
3     u1: User = User.allInstances() ->
4         select(m : User | m.idUser = 'u1'),
5     temporalContext: Set(RBACContext) =
6        self.roleContextAssignment -> select(c |
7        c.oclIsTypeOf(TemporalContext)),
8     timeE: Set (TimeExpression) =
9         temporalContext.oclAsType(TemporalContext).
10        timeexpression.absolute
11        -> flatten() -> asSet(),
12    absoluteE: Set (TimeExpression) = timeE ->
13      select (t | t.absolute.oclAsType(TimeInterval)
14     ->exists(a| a.start.equalTo(u.getCurrentTime())
15      or a.start.isBefore(u.getCurrentTime()))),
16    periodicE: Set(PeriodicTime)= absoluteE.
17       relative.oclAsType(PeriodicTime)
```

```
18        -> flatten() -> asSet()
19 in self.idRole= 'r1' and self.users->includes(u1)
20     implies periodicE.nextStart->select( a |
21     a.equalTo(u.getCurrentTime()))->notEmpty()
```

In this invariant, we first select the instance corresponding to user u_1 (lines 3–4). We retrieve the list `temporalContext` of temporal contexts in which the role should be assigned to user u_1 (lines 5–7) and compute, over the elements of this list, the list `timeE` of time expressions assigned to them (lines 8–11). In this example, the list `timeE` will include a `TimeExpression` with an unbounded `TimeInterval` whose start `TimePoint` corresponds to "July 10, 2014 at 16:00". Then we select among the element of list `timeE`, the list (`absoluteE`) of expressions having an absolute `TimeInterval` that contains the `TimePoint` at which the snapshot was taken (lines 12–15). We check this containment by comparing the time at which the snapshot was taken with the start `TimePoint` of the unbounded`TimeInterval`. Afterwards, we retrieve the list of `PeriodicTime` objects in each expression in list `absoluteE` (lines 16–18). The implication at lines 19–21 states that if the current role is r_1, and user u_1 is member of this role, the time at which the snapshot was taken should match the starting time (derived from the `nextStart` association) of the next period.

A policy on role enabling with a duration associated with an absolute time expression restricts the activation of a role up to a specific duration, as in "enable all roles on April 23, 2015 from 8:00 to 18:00; each role can be active for 3 hours cumulatively". This policy can be checked in OCL as an invariant of class `Session`:

```
1 context Session inv DurationAbsoluteBTIRoleEnab:
2 let u : RBACUtility = RBACUtility.allInstances(),
3    rolesA: Set(Role) = self.enabledRoles ->
4         select (r:Role| r.getCurrentAbsoluteTE(u)
5         -> notEmpty() and
6         r.getCurrentAbsoluteTE(u).hasDuration())
7 in rolesA -> forAll(r: Role |
8      r.getCurrentAbsoluteTE(u).duration.
9      greaterThan(u.getCumulativeActiveDuration
10     (r,self.user, r.getCurrentAbsoluteT(u).
11     duration.timeUnit)))
```

In this OCL constraint, we select a subset (list `rolesA`) of the roles enabled in the current session (lines 3–6). This subset includes the roles whose temporal context for enabling contains an absolute time expression that matches the time at which the snapshot was taken (checked by calling the operation `getCurrentAbsoluteTE` of the class `Role`). For each role in `rolesA`, this absolute time expression should be associated with a duration (checked by calling the operation `hasDuration` of the class `AbsoluteTE`). Then, we check whether the duration of each role in the list is less than the duration specified in its temporal context for enabling (lines 7–11). We assume that the duration of the activation of each role for each user is recorded in a database and made available through the operation `getCumulativeActiveDuration` of class `RBACUtility`.

4.2 Policies with spatial context

A policy on role assignment with a physical location forbids the role assignment when the user is not located in a physical location belonging to the role spatial context for assignment, as in "role r_1 is assigned to user u_1 *only* if the latter is in location loc_1". We assume that loc_1 is of type `PhysicalLocation`. This policy can be checked in OCL as an invariant of class `Role`:

```
1 context Role inv physicalLocationRoleAssign:
2 let  u1 : User = User.allInstances() ->
3         select(m: User | m.idUser = 'u1'),
4      spatialContext: Set(RBACContext) = self.
5         roleContextAssignment -> select(c |
6         c.oclIsTypeOf(SpatialContext)),
7      locPh: Set(PhysicalLocation) = spatialContext.
8        oclAsType(SpatialContext).location.
9        oclAsType(PhysicalLocation)->flatten()
10        ->asSet()
11 in if self.idRole = 'r1' and loc -> exists(l|
12       l.contains(u1.userLocation.location.
13       oclAsType(PhysicalLocation))) then
14            self.users -> includes(u1)
15    else
16            self.users -> excludes(u1)
17    endif
```

In this OCL expression, we first select the instance corresponding to user u_1 (lines 2–3). Then, we retrieve the list `spatialContext` of spatial contexts at which the role should be assigned to user u_1 (lines 4–6) and compute, over the elements of this list, the list `locPh` of physical locations assigned to them (lines 7–10). We check if the current role is r_1 and if a physical location in list `locPh` matches the user's location, by calling the operation `contains` of class `Location`. If this is the case, the list of roles assigned to user u_1 should contain role r_1 (lines 11–14). If it is not the case, the role should not be included in this list (line 16).

A policy on role assignment with a logical location is checked in a similar way by replacing the instances of `PhysicalLocation` with instances of `LogicalLocation`.

A policy on role assignment with a relative location forbids the role assignment when the user is not located in a relative location belonging to the role spatial context for assignment, as in "enable role r_1 *only* within 3 meters outside location loc_1". Location loc_1 can be either of type `PhysicalLocation` or `LogicalLocation`. This policy is checked in OCL as an invariant of class `Session`:

```
context Session inv relativeLocationRoleEnabling:
let r1 : Role = Role.allInstances() ->
     select(r : Role| r.idRole = 'r1'),
   spatialContext: Set(RBACContext) = self.
       roleContextEnabling -> select(c |
       c.oclIsTypeOf(SpatialContext)),
   loc: Set(Location) = spatialContext.
       oclAsType(SpatialContext).location
       ->select(l|l.relativelocation->notEmpty())
       ->flatten()->asSet(),
    relativeLoc: Set(Location)= loc -> collect(l|
       l.computeRelative(l.relativelocation))
       ->flatten()->asSet()
in if relativeLoc -> exists(l|self.user.
       userLocation.location -> exists(pos|
       l.contains(pos))) then
           self.enabledRoles -> includes(r1)
           or self.activeRoles -> includes(r1)
   else
```

```
20            self.enabledRoles -> excludes(r1)
21            and  self.activeRoles -> excludes(r1)
22    endif
```

In this OCL invariant, we first select the instance corresponding to role r_1 (lines 2–3). We retrieve list `spatialContext` of spatial contexts at which the role should be enabled (lines 4–6) and compute, over the locations assigned to each element in this list, the list `loc` of all locations associated with a relative one (lines 7–10). For each location in list `loc`, we compute in `relativeLoc` the location resulting from the call to operation `computeRelative` of class `Location` (lines 11–13). This operation takes in input `RelativeLocation` and is applied to a `PhysicalLocation` or `LogicalLocation`, hereafter called *base location*. It returns the location resulting from the application to the base location of the parameters (distance and direction) of the relative location. The resulting location is always of type `PhysicalLocation`. We check if any of locations in `relativeLoc` matches the user's position (lines 14–16). If it is the case, the role r_1 should be enabled or active (lines 17–18). Otherwise, the role should be disabled (lines 20–21).

Closing remarks. In this section we have shown how the access decision for spatial and temporal RBAC policies defined according to the GemRBAC+CTX model can be reduced to the verification of OCL constraints of an instance of the GemRBAC+CTX model. For space reasons, we have considered temporal and spatial policies in isolation. Nevertheless, we support also *composite context-based policies*, i.e., policies that contain both a temporal and a spatial context. These policies can be checked in OCL by a logical conjunction of the individual OCL constraints corresponding to the composite spatial and temporal policies. The OCL formalization presented here is at the core of a model-driven approach for checking RBAC policies, whose complete description (including technological aspects) is outside the scope of the paper. For example, we have assumed that at any time during the execution of the system for which RBAC policies are defined, we could take a snapshot of the system state and represent it as an instance of the GemRBAC+CTX model. This assumption is based on previous work of some of the authors on model-driven run-time verification [10], which shows how a run-time system can be represented as a "live" instance of a conceptual model, on which to check OCL constraints.

5. INDUSTRIAL APPLICATION

In this section we report on the application of our approach based on GemRBAC+CTX for the modeling of a real application and of its RBAC contextual policies. This application has been developed by a provider of situational-aware information management systems for emergency scenarios. The application allows different (humanitarian) organizations to participate to various missions by providing emergency aid to refugees and casualties. An RBAC system controls the access to mission resources. Due to space limitations, we present a small excerpt of the application and consider only a subset of the actual RBAC entities. Moreover, the description has been sanitized for confidentiality reasons. We assume the system to have two `Users`, *Joe* and *Kim*; three `Roles`, *agencyAdmin*, *missionAdmin* and *missionMember*; one `Permission` *noBandwidthLimit*; one `TemporalContext` (hereafter referred to with the id *freeTime*) that ranges from 00.00 to 06.00 and from 20.00 to 23.59 during weekdays and all-day during the weekend; one `SpatialContext` *Zone1*. The following contextual policies are defined for the system:

PL1: *permission* noBandwidthLimit *is assigned to role* missionMember *only during* freeTime. This policy is typically used to ensure a fair use of the available bandwidth.

PL2: *role* agencyAdmin *is enabled only outside Zone1.* This policy is typically used to ensure that administrative tasks are performed, for security reasons, outside the area of the mission.

PL3: *role* missionAdmin *is enabled only inside Zone1.* This policy is typically used for guaranteeing that mission management is done locally.

The object diagram in Figure 4 depicts a small subset of the instance of the GemRBAC+CTX model that corresponds to a system state during the mission. Roles *agencyAdmin* and *missionMember* are assigned both to *Joe* and to *Kim*. Role `missionAdmin` is assigned to *Joe*. According to policy **PL1**, the temporal context for assignment of permission *noBandwidthLimit* is `freeTime`. It is modeled as a `TimeExpression` composed of four `RelativeTimeIntervals`. Interval `weekend` has a start (`Saturday`) and end (`Sunday`) `RelativeTimePoint` of type `DayOfWeek`. Interval `weekDays` has a start (`Monday`) and end (`Friday`) `RelativeTimePoint` of type `DayOfWeek`. Interval `weekDays` overlays `hours1` and `hours2`: these intervals are of type `HourOfDay`. Let us consider the case in which one wants to check policy **PL1** on this instance. This policy can be checked using the OCL invariant `DayOfWeekHourPermAssign` introduced in Section 4.1. The `if` condition at line 21 is `false` because the time at which the snapshot was taken is not included in the temporal context for enabling permission *noBandwidthLimit*. Hence, we follow the `else` branch, calling operation `excludes` at line 24. Since role *missionMember* is not assigned to permission *noBandwidthLimit*, policy **PL1** is not violated.

According to policy **PL2**, the spatial context for enabling role `AgencyAdmin` is modeled as a `LogicalLocation` (*LLAgencyAdmin*) associated with a `RelativeLocation` (*rloc1*) that contains a `QualitativeDirection` (`inside`). The spatial context for enabling role `MissionAdmin`, indicated in policy **PL3**, is modeled in a similar way (see *LLMissionAdmin*, *rloc2*). The snapshot in Figure 4 includes an instance of `RBACUtility` that captures the `TimePoint` and the `DayWeek` at which it was taken (*Monday, May 4, 2015 at 12:15:23*). In this snapshot, users are connected to the system; we model this with `Sessions`. In session *sesJoe*, role `missionAdmin` is active and role `missionMember` is enabled for user *Joe*. In session `sesKim`, roles `missionMember` and `agencyAdmin` are enabled for user *Kim*. This model instance also captures the location of the two users at the time of their connection. Each of these locations is represented with an association between each `User` and his `SpatialContext`, which contains an object of type `Point`. Objects `pK` and `pJ` refers to the position of users *Kim* and *Joe*. We assume that only *Joe* is located in the defined zone *Zone1*. We now consider the case in which one wants to check policy **PL2** on this model instance. This policy can be checked on both `Sessions`, `sesKim` and `sesJoe`, using the OCL invariant `relativeLocationRoleEnabling` (shown in section 4.2) parametrized with role *agencyAdmin*. For `Session` `sesKim`, the `if` condition at lines 14–16 is true because *Kim*, according to the assumption made above, is outside *Zone1*, meaning that her position (object `pK`) is contained in the lo-

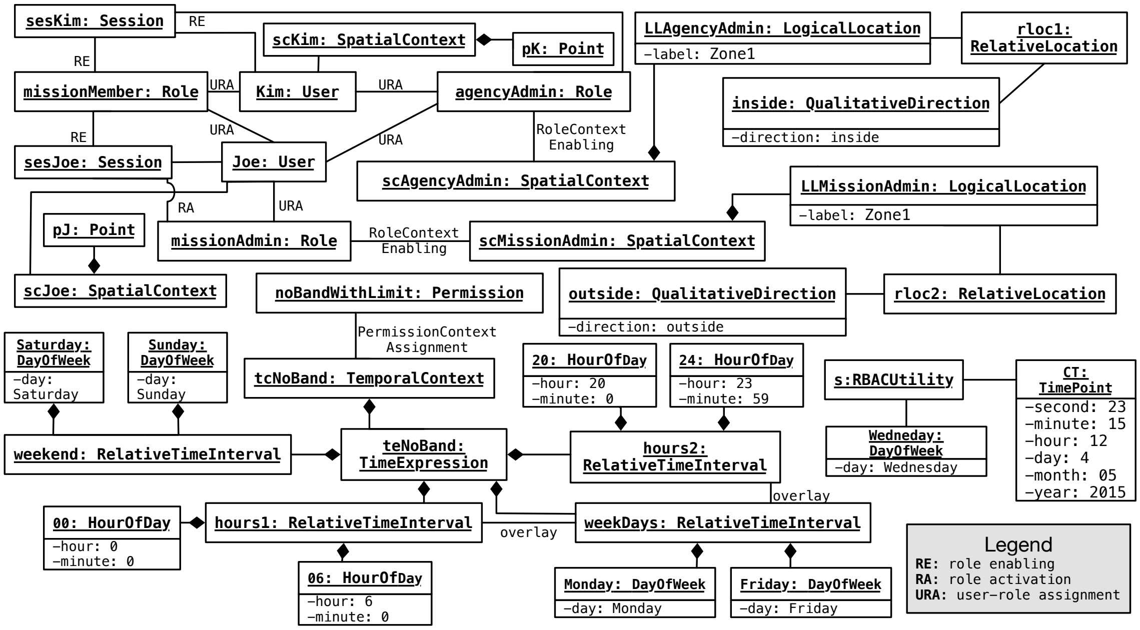

Figure 4: An instance of the GemRBAC+CTX model representing a system state of the example application.

cation `LLAgencyAdmin` associated with the spatial context for enabling role *agencyAdmin* (object `scAgencyAdmin`). Hence, we follow the `then` branch, calling the operation `includes` at lines 17–18. Since role `agencyAdmin` is enabled in the session, this operation returns true, meaning that policy **PL2** is not violated for Kim. Policy **PL3** is checked in a similar way on sessions `sesKim` and `sesJoe`, using the same OCL invariant parametrized with role *missionAdmin*. This policy is not violated since role *missionAdmin* is not enabled for `Kim` (i.e., there is no association between objects `sesKim` and `missionAdmin`) and is active for `Joe` (i.e., there is an association between `sesJoe` and `missionAdmin`).

Evaluation. In the evaluation of the industrial application, we mainly focused on assessing whether all contextual policies required by the application could be expressed with our approach. The new model has allowed the security engineers of our partner to define *19 new types* of RBAC contextual policies. With these new policies, engineers can now tune the definition of fine-grained (from the point of view of context) RBAC policies. This is a major improvement over the previous solution, which granted permissions under any context, for the lack of better specification methods. Moreover, since the GemRBAC+CTX defines structural constraints among entities, it will prevent end-users to define RBAC policies that are not well-formed.

Overall, the application of the modeling approach supported by GemRBAC+CTX has been warmly welcome by the security engineers of HITEC. Nevertheless, the engineers also reported some drawbacks of our current approach. In particular, they remarked that defining RBAC policies as OCL constraints on the GemRBAC+CTX class model was sometimes cumbersome, especially for complex policies (with large corresponding models). To address this limitation, as part of future work, we plan to define a domain-specific language (DSL) on top of the GemRBAC+CTX model, to allow the definition of policies at a higher-level of abstraction, using a syntax close to natural language.

We also assessed the performance of our model-driven approach in terms of the time required to provide an access decision, i.e., the time needed to evaluate an OCL constraint corresponding to a certain contextual RBAC policy. We used a laptop with a 2.2 GHz Intel Dual-Core i7 CPU and 16GB of memory, running Eclipse Mars Service Release 1, JavaSE-1.8, Eclipse OCL v.4.1.0. The evaluation of policies **PL1**–**PL3** on the complete model (with 67 roles, 252 permissions, 914 users, 400 temporal contexts and 700 spatial contexts) took 24 ms for **PL1**, 2847 ms for **PL2**, and 2405 ms for **PL3**. We considered the worst-case scenario when all the roles are active and all the roles are assigned to permission p_1. These results confirm the suitability of our approach for operationalizing access decisions for contextual policies in real applications. Although scalability studies on policy checking are out of the scope of this paper, we are confident in the scalability of our approach also for much larger models. For example, community experience [19] shows that Eclipse OCL can check complex OCL constraints on models with millions of elements in few seconds.

6. RELATED WORK

Several extensions of the original RBAC model [23] have been proposed in the literature to express temporal and/or spatial contexts. The first proposed temporal model, TRBAC [6], introduces temporal policies on role enabling. It supports absolute, relative and periodic time. A generalization of this model, called GTRBAC [13], includes temporal policies on role assignment. It also supports the specification of temporal policies restricting the activation duration of a given role. A limitation of these two models is the lack of support for temporal policies at the permission level. As for spatial extensions of RBAC, GeoRBAC [7] introduces spatial policies on role enabling. LRBAC [20] and SRBAC [12] support policies with spatial context not only for role enabling but also for user-role and role-permission

Table 1: Support of policies in RBAC models

	Contextual policies								Scope				Decision	Non-contextual
	ART	PTE	I	AD	PL	LL	RL	CP	PA	PE	RA	RE	algorithm	policies [5]
TRBAC [6]	+	+	+	-	-	-	-	N/A	-	-	-	+	+	2/18
GTRBAC [13]	+	+	+	+	-	-	-	N/A	-	-	+	+	+	9/18
GeoRBAC [7]	-	-	-	-	+	+	-	N/A	-	-	-	+	+	4/18
LRBAC [20]	-	-	-	-	+	+	-	N/A	+	-	+	+	+	2/18
SRBAC [12]	-	-	-	-	+	+	-	N/A	+	-	+	+	-	3/18
STARBAC [2]	+	+	-	-	+	+	-	+	-	+	-	+	-	0/18
ESTRBAC [1]	+	+	+	-	+	+	-	+	+	-	+	+	+	3/18
STRBAC [22]	+	+	-	-	+	+	-	-	+	-	+	+	+	7/18
GSTRBAC [18]	+	+	-	-	+	+	-	+	+	-	+	+	-	8/18
OrBAC [9]	+	+	-	-	+	+	-	+	-	+	-	+	+	5/18
LotRBAC [8]	+	+	+	+	+	+	+	+	+	-	+	+	-	6/18
GemRBAC+CTX	+	+	+	+	+	+	+	+	+	+	+	+	+	18/18

Legend. ART: Absolute and Relative TE; PTE: Periodic TE; I: Index; AD: Activation Duration; PL: Physical Location; LL: Logical Location; RL: Relative Location; CP: Composite context-based policies, RA: User-role Assignment, RE: Role Enabling, PA: Role-permission Assignment, PE: Permission Enabling.

assignment. However, these models do not support spatial policies for permission enabling and have limited support for role disabling. Among RBAC extensions that support both temporal and spatial policies, there is STARBAC [2], with support for contextual policies for role enabling. ESTRBAC [1] extends STARBAC to support contextual policies for role enabling and both user-role and role-permission assignment. STRBAC [22] does not support the definition of composite context-based policies. This limitation is overcome by GSTRBAC [18]. OrBAC [9] defines context as an additional condition restricting the user access in different organizations, and abstracts away from the usual concepts of spatial and temporal context. STRBAC, GSTRBAC, and OrBAC do not support the concepts of index and activation duration for temporal policies, and the concept of relative location for spatial policies. LotRBAC [8] extends GTRBAC by assigning a location to each user, role and permission. It does not support permission enabling and the specification of the perimeter of physical locations.

Regarding checking contextual policies defined over the models surveyed above, the SRBAC [12], STARBAC [2], GSTRBAC [18] and LotRBAC [8] models do not come with any access decision algorithm.

Table 1 summarizes to which extent the RBAC models discussed above support the various concepts related to contextual policies. It also indicates the scope (assignment/enabling of permissions and/or roles) in which such policies can be used, the availability of an access decision algorithm, and the number of types of non-contextual policies (as identified in the taxonomy presented in [5]) they support. As one can see, the GemRBAC+CTX model proposed in this paper is the only one that supports 1) all the various spatio-temporal concepts for contextual policies; 2) the use of such policies for the assignment and enabling of both roles and permissions; 3) a checking procedure for these policies; 4) a complete support for non-contextual properties.

As for the use of UML and OCL for modeling and checking RBAC policies, several approaches have been proposed (such as [4, 16, 24, 15, 21, 25]); we refer the reader to our previous work [5] for a detailed discussion.

7. CONCLUSION AND FUTURE WORK

Several application domains require the definition of RBAC policies that restrict access based on the location of the user or the time at which she requests the access. These policies are called contextual policies and come with many facets, ranging from complex types of temporal expressions to different types of locations and their topological relations. The conceptual RBAC models proposed so far in the literature do not support all the facets of contextual policies and provide limited support to evaluate these policies in order to make an access decision.

In this paper we presented GemRBAC+CTX, a conceptual model that contains all the entities required to accurately specify temporal and spatial contexts in RBAC policies. We formalized these policies as OCL constraints on the GemRBAC+CTX model, as a way to operationalize the access decision for user's requests. We reported on the application of GemRBAC+CTX to model the RBAC policies of a real application in the domain of disaster relief intervention. The use of GemRBAC+CTX has allowed security engineers to define 19 new types of contextual policies, with a fine-grained, precise description of contexts. The preliminary experimental results show the suitability of our model-driven approach for checking RBAC contextual policies.

As part of future work, we plan to extend GemRBAC+CTX based on the recent proposals that support proximity-based policies [14] and geo-social ones [3]. We also plan to define a domain-specific language on top of the GemRBAC+CTX model, to allow the definition of RBAC policies using a syntax close to natural language. The proposed model-driven approach for policy checking could also be integrated into a platform for model-driven run-time enforcement, tailored for checking policies defined using GemRBAC+CTX.

8. ACKNOWLEDGMENTS

This work has been supported by the National Research Fund, Luxembourg (FNR/P10/03) and by a grant by HITEC Luxembourg. Ameni Ben Fadhel is also supported by the Faculty of Science, Technology and Communication of the University of Luxembourg.

9. REFERENCES

[1] S. Aich, S. Mondal, S. Sural, and A. Majumdar. Role Based Access Control with Spatiotemporal Context for Mobile Applications. In *Trans. on Comput. Sci. IV*, volume 5430 of *LNCS*, pages 177–199. Springer, 2009.

[2] S. Aich, S. Sural, and A. Majumdar. STARBAC: Spatiotemporal Role Based Access Control. In *Proc. of the OTM Conferences 2007*, volume 4804 of *LNCS*, pages 1567–1582. Springer, 2007.

[3] N. Baracaldo, B. Palanisamy, and J. Joshi. Geo-Social-RBAC: A location-based socially aware access control framework. In *Proc. of NSS 2014*, volume 8792 of *LNCS*, pages 501–509. Springer, 2014.

[4] D. Basin, J. Doser, and T. Lodderstedt. Model Driven Security: From UML Models to Access Control Infrastructures. *ACM Trans. on Soft. Eng. and Meth.*, 15:39–91, 2006.

[5] A. Ben Fadhel, D. Bianculli, and L. Briand. A comprehensive modeling framework for role-based access control policies. *Journal of Systems and Software*, 107:110–126, September 2015.

[6] E. Bertino, P. A. Bonatti, and E. Ferrari. TRBAC: A Temporal Role-based Access Control Model. *ACM Trans. Inf. Syst. Secur.*, 4(3):191–233, Aug. 2001.

[7] E. Bertino, B. Catania, M. L. Damiani, and P. Perlasca. GEO-RBAC: A Spatially Aware RBAC. In *Proc. of SACMAT 2005*, pages 29–37. ACM, 2005.

[8] S. Chandran and J. Joshi. LoT-RBAC: A Location and Time-Based RBAC Model. In *Proc. of WISE 2005*, volume 3806 of *LNCS*, pages 361–375. Springer, 2005.

[9] F. Cuppens and N. Cuppens-Boulahia. Modeling contextual security policies. *Int. JIS*, 7(4):285–305, 2008.

[10] W. Dou, D. Bianculli, and L. Briand. Revisiting model-driven engineering for run-time verification of business processes. In *Proc. of SAM 2014*, volume 8769 of *LNCS*, pages 190–197. Springer, September 2014.

[11] Eclipse. Eclipse OCL tools. http://www.eclipse.org/modeling/mdt/?project=ocl.

[12] F. Hansen and V. Oleshchuk. SRBAC: A spatial role-based access control model for mobile systems. In *Proc. of NORDSEC2003*, pages 129–141, 2003.

[13] J. B. D. 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, January 2005.

[14] M. S. Kirkpatrick, M. L. Damiani, and E. Bertino. Prox-RBAC: A proximity-based spatially aware RBAC. In *Proc. of GIS 2011*, pages 339–348. ACM, 2011.

[15] M. Kuhlmann and M. Gogolla. Modeling and validating Mondex scenarios described in UML and OCL with USE. *Formal Aspects of Computing*, 20:79–100, 2008.

[16] M. Kuhlmann, L. Hamann, and M. Gogolla. Extensive validation of OCL models by integrating SAT solving into USE. In *Proc. of TOOLS 2011*, volume 6705 of *LNCS*, pages 290–306, 2011.

[17] OMG. Object Constraint Language. http://www.omg.org/spec/OCL/, 2012.

[18] A. Ramadan, A.-L. Mustafa, R. Indrakshi, and F. Robert B. Specification, Validation, and Enforcement of a Generalized Spatio-Temporal Role-Based Access Control Model. *IEEE Syst. J.*, 7(3):501–515, September 2013.

[19] I. Ràth and E. Willink. Fast, faster and super-fast queries. http://www.eclipse.org/modeling/mdt/ocl/docs/publications/EclipseConEurope2012/FastQueries.pdf, October 2012. EclipseCon Europe.

[20] I. Ray, M. Kumar, and L. Yu. LRBAC: A Location-Aware Role-Based Access Control Model. In *Proc. of ICISS 2006*, volume 4332 of *LNCS*, pages 147–161. Springer, 2006.

[21] I. Ray, N. Li, R. France, and D.-K. Kim. Using UML to visualize role-based access control constraints. In *Proc. of SACMAT 2004*, pages 115–124, 2004.

[22] I. Ray and M. Toahchoodee. A Spatio-temporal Role-Based Access Control Model. In *Proc. of DBSec 2007*, volume 4602 of *LNCS*, pages 211–226. Springer, 2007.

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

[24] K. Sohr, T. Mustafa, X. Bao, and G.-J. Ahn. Enforcing role-based access control policies in web services with UML and OCL. In *Proc. of ACSAC 2008*, pages 257–266. IEEE, 2008.

[25] M. Strembeck and J. Mendling. Modeling Process-related RBAC Models with Extended UML Activity Models. *Information and Software Technology*, 53:456–483, 2011.