securityandtrust.lu

Software Verification and Validation Laboratory: GemRBAC-DSL: a High-level Specification Language for Role-based Access Control Policies

Ameni Ben Fadhel, Domenico Bianculli and Lionel Briand

Interdisciplinary Centre for Security, Reliability and Trust

University of Luxembourg

TR-SnT-2016-4

ISBN: 978-2-87971-151-5

April 27, 2016

Version 1.0

uni.lu

UNIVERSITÉ DU LUXEMBOURG

GemRBAC-DSL: a High-level Specification Language for Role-based Access Control Policies

Ameni Ben Fadhel, Domenico Bianculli, Lionel Briand

May 13, 2016

Abstract

A role-based access control (RBAC) policy restricts a user to perform operations based on her role within an organization. Several RBAC models have been proposed to represent different types of RBAC policies. However, the expressiveness of these models has not been matched by specification languages for RBAC policies. Indeed, existing policy specification languages do not support all the types of RBAC policies defined in the literature.

In this paper we aim to bridge the gap between highly-expressive RBAC models and policy specification languages, by presenting GemRBAC-DSL, a new specification language designed on top of an existing, generalized conceptual model for RBAC. The language sports a syntax close to natural language, to encourage its adoption among practitioners. We also define semantic checks to detect conflicts and inconsistencies among the policies written in a GemRBAC-DSL specification. We show how the semantics of GemRBAC-DSL can be expressed in terms of an existing formalization of RBAC policies as OCL (Object Constraint Language) constraints on the corresponding RBAC conceptual model. This formalization paves the way to define a model-driven approach for the enforcement of policies written in GemRBAC-DSL.

1 Introduction

In a role-based access control (RBAC) system, a user's request to access a resource or perform an operation is allowed or denied based on access control policies (also called authorization constraints) that take into account the role of the requester. Various types of RBAC policies have been proposed in the literature; in this paper, we refer to the policies classified in the taxonomy recently proposed in [7]. This taxonomy identifies eight types of RBAC policies: prerequisite [4, 23], cardinality [2], precedence and dependency [24], role hierarchy [23], separation of duty (SoD) [3, 25], binding of duty (BoD) [27], delegation and revocation [13, 28], and contextual (both temporal and spatial) [10, 19].

Several RBAC models have been proposed to characterize the conceptual entities that are needed to represent these policies. The original, standardized RBAC96 model [23] supports only prerequisite, cardinality, role hierarchy, and simple SoD policies. Various extensions of this model have been defined to support additional policies. For example, support for delegation policies have been added in the models proposed in [13, 26, 28, 29]; the models introduced in [5, 9, 10, 19, 22] have added support for contextual policies. In our previous work [7] we proposed the GemRBAC model, designed with the goal of integrating, in a coherent and comprehensive model, all the

conceptual entities required to express the various types of RBAC polices proposed in the literature. We have also proposed the GemRBAC+CTX model [8], which is an extension of the GemRBAC model that adds support for richer and more expressive contextual policies.

On a par with the definition of complex and more expressive RBAC models, there is the problem of defining *policy specification languages* that are at least as expressive as the policies supported by the existing models. While RBAC models provide the fundamental concepts needed to formalize various types of RBAC policies, policy specification languages represent a means to express RBAC policies that can be used (for both policy definition and enforcement) in practice. One group of proposals to define such languages revolves around XACML [21], the OASIS standard for defining access control policy languages. Since XACML does not support RBAC models natively, it has been extended with profiles specific to RBAC [1, 6]. Other types of RBAC policy languages are ontology-based [15, 16] or logic-based [3, 12, 17] languages. The main problem of existing RBAC specification languages is that they do not support all the types of RBAC policies defined in the literature. For example, a simple delegation transfer policy like "any user with role r_1 can transfer her role to any user assigned to role r_2" cannot be expressed in any of the existing languages. Moreover, the semantics of some of these languages is not executable for the purpose of enforcing the policies specified with them. Furthermore, many of them are not designed to be used by practitioners.

These problems have practical implications, since the lack of expressive policy specification languages limits the adoption, among practitioners, of the more expressive RBAC models proposed in the literature. In turn, this situation makes practitioners use simple(r) RBAC models, resulting in systems underspecified from the point of view of access control. For example, the industrial partner for the research project in which this work has been carried out, is a provider of situational-aware information systems for emergency scenarios; given the criticality of such scenarios, highly-detailed role access control policies are an essential need for them. However, although our partner is aware of state-of-the-art proposals for expressive RBAC models, it could not adopt them in practice, because of the lack of a policy specification language as expressive as them. Besides the expressiveness, another requirement on the specification language stated by our partner is the possibility of interpreting the policies written in the language, with the purpose of automatically generating policy enforcement mechanisms.

In this paper we aim to bridge the gap between highly-expressive RBAC models and policy specification languages, by presenting GemRBAC-DSL, a new specification language for RBAC policies. The language has been designed to cover the various types of RBAC policies captured by the GemRBAC+CTX model. Being based on this model, the language is quite expressive (see Section 3 for a detailed comparison with the state-of-the-art). Moreover, GemRBAC-DSL sports a syntax close to natural language, to encourage its adoption among practitioners. Furthermore, we define semantic checks that can be run on a GemRBAC-DSL policy specification, to detect conflicting and inconsistent policy definitions (e.g., a conflict between two policies, one defining an SoD policy and another one defining a BoD policy *for the same set of permissions*). We have built an editor for the language based on the XText framework and the Eclipse platform, and integrated the semantics checks in it.

The GemRBAC+CTX model and its ancestor GemRBAC, which have inspired the design of GemRBAC-DSL, come with an operationalization of the semantics of the policies they support. This operationalization is defined following a model-driven approach, in which the semantics of each RBAC policy is expressed as an OCL (Object

Constraint Language) constraint on the RBAC model. Since the expressiveness of GemRBAC-DSL is the same as that of the GemRBAC+CTX model, we define the semantics of GemRBAC-DSL by mapping the constructs of the language to the corresponding OCL constraints defined for the GemRBAC+CTX model in [7,8]. This mapping allows users of GemRBAC-DSL to benefit from the model-driven approach for policy enforcement proposed in [7, 8]. Indeed, a policy written in GemRBAC-DSL can be enforced by evaluating the corresponding OCL constraint (as defined in the mapping) on an instance of the GemRBAC+CTX model obtained from the system in which the policy is being enforced. This model-driven approach for policy enforcement can be used both at design time and at run time and relies on standardized technologies, supported by industry-strength tools (such as Eclipse OCL [14]).

Summing up, the main contributions of the paper are: (a) the definition of the GemRBAC-DSL specification language for RBAC policies; (b) the definition of the semantic checks for a GemRBAC-DSL policy specification; (c) a publicly-available implementation of an editor to write policies in GemRBAC-DSL and check for potential conflicts and inconsistencies among them.

The rest of the paper is organized as follows. Section 2 illustrates a motivating example for this work. Section 3 discusses the state of the art. Section 4 presents the language, illustrating the syntax and providing examples for each type of policy. Section 5 defines the semantic checks for policies expressed in GemRBAC-DSL. Section 6 provides a brief overview of the semantics of the language. Section 7 discusses the design trade-offs and the limitations of GemRBAC-DSL, as well as its adoption by our industrial partner. Section 8 concludes the paper and provides directions for future work.

2 Motivating example

In this section we illustrate an example of RBAC policy specifications that motivates our work. The example represents a subset of a real-world case study, defined in collaboration with our industrial partner, a provider of situational-aware information systems for emergency scenarios. The case study deals with the specification of the RBAC policies for a Web application that provides information related to humanitarian missions, ranging from satellite images to highly-confidential data about refugees and casualties. For space and confidentiality reasons we consider a small, sanitized subset of the system, but provide a representative list of policies that covers exhaustively all the types of RBAC policies used in the policy specifications of the case study.

We consider a humanitarian mission taking place from February 12, 2016 to June 8, 2016 in a geographical area symbolically known as "*Zone1*", delimited by four segments with coordinates (longitude and latitude in decimal degrees, elevation in meters): (15:24:200)–(20:27:200), (20:27:200)–(17:27:200), (17:27:200)–(15:27:200), (15:27:200)–(15:24:200). The mission defines five roles (*admin*, *assistant*, *trainee*, *participant*, *analyst*), five permissions (*add_casualty*, *modify_casualty*, *delete_casualty*, *analyse_satellitePhoto*, *save_satellitePhoto*), four operations (*create*, *read*, *update*, *delete*). The access control policies for this mission are:

PL1: To acquire role *trainee*, a user must be assigned to role *participant*.

PL2: Role *assistant* cannot be assigned to more than three users.

PL3: Role *trainee* is enabled only if role *admin* is active. The latter cannot be deactivated if the role *trainee* is still active.

PL4: If a user acquires role *assistant*, she will also acquire all its junior roles.
PL5: A user can acquire either role *assistant* or *trainee*.
PL6: A user can activate roles *assistant* and *admin* at the same time, as long as she does not perform all the operations (*create*, *read*, *update*, *delete*) on the same object (of type "casualty record").
PL7: The operations allowed by permissions *add_casualty*, *modify_casualty*, and *delete_casualty* should be performed by users having the same role.
PL8: In case a user assigned to role *admin* is on leave, she has to delegate all the permissions associated with her role to another user who is assigned to role *assistant*. The delegation lasts for two weeks; during this period the delegator is still allowed to execute the permissions associated with the role she has delegated. Moreover, the delegated role can be further delegated (by a delegate), with a maximum delegation depth of 2.
PL9: The delegation regulated by policy PL8 can be revoked by any user assigned to role *admin*. The revocation will not affect the (further) delegations of role *admin* possibly performed by delegated users. Moreover, the revocation will only remove the affected users from the delegated role *admin*, and will not impact the other roles possibly acquired through a role hierarchy (of the delegated role).
PL10: Role *analyst* is a part-time job; it can be active for a maximum duration of 4 hours per day.
PL11: Role *participant* is enabled for the entire duration of the mission.
PL12: Permission *add_casualty* is assigned to role *trainee* only during weekdays from 8:00 to 17:00.
PL13: Role *admin* is enabled only in zone *Zone1*.
PL14: Role *trainee* is enabled at 100 meters from the boundary inside *Zone1*.

The policies above show that defining the access control requirements of our example requires to deal with several types of policies (see taxonomy in [7]): prerequisite (PL1), cardinality (PL2), precedence (PL3), role hierarchy (PL4), SoD (PL5, PL6), BoD (PL7), delegation (PL8), revocation (PL9), contextual (PL10–PL12). To express these policies security engineers need a policy specification language *expressive enough* to support all of them. In the next section we review existing RBAC specification languages in terms of the policy types they support.

3 State of the art

One of the first policy languages proposed for RBAC is RCL2000 [3], which is a formal language based on first-order predicate logic and defined on top of the RBAC96 model. The language supports only role hierarchy and separation of duty policies. FORBAC [12] is also an extension of RBAC based on first-order logic. It adds support for attributes in policies and numeric constraints; both features enable the definition of more complex policies, like those containing contextual constraints. However, FORBAC does not support role hierarchy, delegation, cardinality, and separation of duty. Furthermore, a limitation shared both by RCL2000 and FORBAC is the difficulty of use by practitioners, since both languages require a strong mathematical background. Tower [17] is a high-level specification language for access control policies; it supports delegation and history-based SoD policies. However, delegation and revocation policies are defined only as administrative operations for role-to-user assignment, i.e., in terms of adding/removing a role to/from a user.

Table 1: Support of policies in RBAC languages

	Prq	RH	Card	Prec	SoD					BoD	Context		Deleg	Rev
					S	D	Obj	Op	His		T	L		
RCL2000 [3]	-	+	+	-	+	+	-	-	-	-	-	-	-	-
FORBAC [12]	+	-	-	-	-	-	-	-	-	-	+	+	-	-
Tower [17]	+	+	+	+	+	+	+	+	+	-	-	-	+/-	+/-
XACML [1, 6]	+	+	+	-	+	+	-	-	-	-	+	+	GT	-
X-RBAC [18]	+	+	+	-	+	+	+	-	-	-	+	+	-	-
X-GTRBAC [11]	+	+	+	-	+	+	+	-	-	-	-	+	-	-
ROWLBAC [16]	+	+	+	-	+	+	+	-	-	-	-	-	GT	-
XACML+OWL [15]	+	+	+	+	+	+	+	+	+	-	-	-	-	-
RBAC-DSL [26]	+	+	+	+	+	+	+	+	+	-	-	-	GT	+

Legend. Prq: Prerequisite; RH: Role Hierarchy; Card: Cardinality; Prec: Precedence and Dependency; S: Static SoD; D: Dynamic SoD; Obj: Object-based DSoD; Op: Operational-based DSoD, His: History-based DSoD, Deleg: Delegation.

Another research stream considers XML-based languages, starting from the definition of XACML (eXtensible Access Control Markup Language) [21]. XACML is a language for access control, standardized by the OASIS community. The XACML standard provides not only the specification language for access control policies but also a reference enforcement architecture. XACML is a general-purpose language for expressing various types of access control models and policies; being general-purpose, it does not support RBAC natively (e.g., sessions are not supported). RBAC support can be added to XACML by means of profiles. The OASIS RBAC profile for XACML [6] supports only role hierarchy and static separation of duty policies. Another RBAC profile of XACML [1] supports separation of duty, delegation, and context-based policies. X-RBAC [18] is an XML-based specification language for RBAC policies in multi-domain environments where authorization policies are distributed over several domains. X-RBAC supports context-based, role hierarchy, cardinality and separation of duty policies. X-GTRBAC [11] is a language defined on top of the GTRBAC model [19] for specifying RBAC policies for heterogeneous and distributed enterprise resources. X-GTRBAC adds the concept of user's credentials to the GTRBAC model: users are grouped according to their credentials. X-GTRBAC supports cardinality, separation of duty, role hierarchy, and temporal policies.

Another language, conceptually similar to XACML, is xfACL (eXtensible Functional Language for Access Control) [20]. xFACL is a general-purpose access control language, which tries to combine the benefits of XACML and RBAC. It is based on the specification of attributes for entities involved in decisions (e.g., users, operations) and supports auxiliary policies to extend its expressiveness. The latter is also its main drawback, since support for each type of policy has to be manually added by means of an auxiliary function.

Other languages deal with the integration of ontologies to provide a semantic interpretation of access control policies across different, heterogenous organizations, and to support advanced access control policies. For instance, ROWLBAC [16] is an ontology-based language that combines OWL (Web Ontology Language) and RBAC properties. The language supports the specification of prerequisite, role hierarchy, SoD, and delegation policies. The XACML+OWL framework [15] combines OWL and XACML. Role hierarchy and separation of duty policies are specified using OWL, while the

XACML engine is used to make decisions for user access requests. The interactions between the XACML engine and the OWL ontology are defined through semantic functions.

RBAC DSL [26] is a domain-specific language for RBAC based on UML diagrams and OCL constraints. The corresponding meta-model includes two levels: the *policy* level and the *user Access* Level. The first level defines the basic RBAC concepts: roles, resources, permissions and operations. At this level, SoD, cardinality, and role hierarchy policies are represented as UML attributes and associations. The second level defines the concepts of user, session, resource access, and snapshot (i.e., an instance of an RBAC model at a specific time point). A predecessor/ successor relation is defined for the concept of user, session and access to identify the individual users, sessions and accesses over time. At this level, authorization policies are defined as OCL constraints based on the information available in the policy level. RBAC DSL supports also delegation and revocation policies. However, as acknowledged also in [8], defining RBAC policies as OCL constraints can be difficult, since it requires a high level of knowledge and expertise with OCL, especially in our case in which OCL constraints tend to be rather complex to express RBAC policies.

Table 1 summarizes the support for the various types of RBAC policies in the policy specification languages discussed above. The types of policies used for the comparison have been taken from the taxonomy in [7] and reflect the ones we have observed in our industrial case study. We remark that the specification of some type of policies, such as context-based and delegation, depends not only on the language but also on the underlying model.

One can see that none of these languages is expressive enough to express all the policies presented in Section 2, related to our industrial case study. Moreover, the analysis has also shown that the majority of existing policy specification languages is based on some formalism (either first-order logic fragments, including OCL, or ontology languages based on description logic) that require a strong theoretical and mathematical background, which is rarely found among practitioners. Hence, we contend that there is a need for an expressive specification language for RBAC policies that can also be used by practitioners.

4 The GemRBAC-DSL language

The GemRBAC-DSL policy specification language has been designed as a domain-specific language built on top of the GemRBAC+CTX model. The choice of the underlying model for the language has been dictated by the need to support a large variety of RBAC policies, like the ones used for the specification of our industrial case study (see Section 2). Hence, the language inherits the expressiveness of the GemRBAC+CTX model (see [7,8]).

The main goal during the design of the language has been to encourage its use among practitioners. Indeed, the language captures the main RBAC concepts that security analysts are familiar with and allows for their specification using a syntax close to natural language. Furthermore, the language design process has incorporated the feedback provided by the security analysts of our industrial partner, who have commented on the expressiveness and the clarity of the language. At the time of writing, the language is being introduced into the security development lifecycle of our partner, to support the top-down definition of access control policies and enforcement mechanisms.

4.1 Syntax

The syntax of GemRBAC-DSL is shown in Fig. 1, using the Backus-Naur Form (BNF) notation: non-terminal symbols are enclosed in angle brackets; terminal symbols are enclosed in single quotes; (derivation) rules are denoted with the ::= symbol; alternatives within a rule are indicated using a vertical bar; a star stands for zero or more occurrences of an element; a plus stands for one or more occurrences of an element; square brackets denote optional elements.

⟨*RBAC-definition*⟩ ::= ⟨*preamble*⟩ ⟨*policies*⟩

⟨*preamble*⟩ ::= ⟨*users*⟩ ⟨*roles*⟩ ⟨*permissions*⟩ ⟨*operations*⟩ ⟨*role-hierarchy*⟩ ⟨*permission-hierarchy*⟩ ⟨*geofences*⟩

⟨*users*⟩ ::= '`users:`' ⟨*user*⟩ ('`,`' ⟨*user*⟩)* '`;`'

⟨*roles*⟩ ::= '`roles:`' ⟨*role*⟩ ('`,`' ⟨*role*⟩)* '`;`'

⟨*permissions*⟩ ::= '`permissions:`' ⟨*permission*⟩ ('`,`' ⟨*permission*⟩)* '`;`'

⟨*operations*⟩ ::= '`operations:`' ⟨*operation*⟩ ('`,`' ⟨*operation*⟩)* '`;`'

⟨*id*⟩ ::= ('`a`'-'`z`' | '`A`'-'`Z`' | '`0`'-'`9`')+

⟨*user*⟩ ::= ⟨*id*⟩

⟨*role*⟩ ::= ⟨*id*⟩

⟨*permission*⟩ ::= ⟨*id*⟩

⟨*operation*⟩ ::= ⟨*id*⟩

⟨*role-hierarchy*⟩ ::= '`role-hierarchy:`' (⟨*rHierarchy*⟩ ('`,`' ⟨*rHierarchy*⟩)* | '`none`') '`;`'

⟨*permission-hierarchy*⟩ ::= '`permission-hierarchy:`' (⟨*pHierarchy*⟩ ('`,`' ⟨*pHierarchy*⟩)* | '`none`') '`;`'

⟨*rHierarchy*⟩ ::= ⟨*role*⟩ '`: {`' ⟨*role*⟩ ('`,`' ⟨*role*⟩)* '`}`'

⟨*pHierarchy*⟩ ::= ⟨*permission*⟩ '`: {`' ⟨*permission*⟩ ('`,`' ⟨*permission*⟩)* '`}`'

⟨*geofence*⟩ ::= '`geofences:`' (⟨*geofence*⟩ ('`,`' ⟨*geofence*⟩)* | '`none`') '`;`'

⟨*geofence*⟩ ::= ⟨*id*⟩

⟨*policies*⟩ ::= '`policies:`' (⟨*policy*⟩'`;`')+

⟨*policy*⟩ ::= ⟨*id*⟩ '`:`' (⟨*Prerequisite*⟩ | ⟨*Cardinality*⟩ | ⟨*PrecEnabling*⟩ | ⟨*Hierarchy*⟩ | ⟨*SSoD*⟩ | ⟨*DSoD*⟩ | ⟨*BoD*⟩ | ⟨*Delegation*⟩ | ⟨*Revocation*⟩ | ⟨*ContextPolicy*⟩)

Figure 1: Grammar of GemRBAC-DSL

A GemRBAC-DSL policy specification (captured by the start symbol ⟨*RBAC-definition*⟩) contains a ⟨*preamble*⟩ and a list of ⟨*policies*⟩. The ⟨*preamble*⟩ contains the declaration of the main entities that will be used in the rest of the specification[1]: the list of users ⟨*users*⟩, the list of roles ⟨*roles*⟩, the list of permissions

[1]Notice that the assignments of users to roles, of permissions to roles, and of operations to per-

⟨*permissions*⟩, and the list of operations ⟨*operations*⟩. The ⟨*preamble*⟩ contains also the list ⟨*role-hierarchy*⟩ of role hierarchy relations, and the list ⟨*permission-hierarchy*⟩ of permission hierarchy relations. Within these lists, each hierarchy relation (⟨*rHierarchy*⟩ for role hierarchy and ⟨*pHierarchy*⟩ for permission hierarchy) declares the parent (role or permission) followed by the list of its junior (roles or permissions, respectively). The absence of role (or permission) hierarchies is explicitly denoted with the keyword '`none`'. The ⟨*preamble*⟩ ends with the list ⟨*geofences*⟩ of logical locations, i.e., symbolic abstractions that refer to real physical locations [8]. All the lists used in the ⟨*preamble*⟩ are comma-separated and contain alphanumeric identifiers. Finally, the list of policies ⟨*policies*⟩ contains the actual policy specifications, where each policy is composed by an identifier and by its body.

The following subsections illustrate each type of policy supported by GemRBAC-DSL; for each policy, we include a short definition, the syntax, its explanation, and an example of specification based on the policies defined in Section 2.

4.2 Prerequisite policy

A prerequisite policy defines a precondition on a role or a permission assignment: to acquire a role (or a permission), a user must have been already assigned to another role (or permission) [4, 23]. The syntax for this policy is:

⟨*Prerequisite*⟩ := ⟨*PrereqRole*⟩ | ⟨*PrereqPermission*⟩ (1)

⟨*PrereqRole*⟩ ::= '`assign-role`' ⟨*role1*⟩ '`prerequisite`' ⟨*role2*⟩ (2)

⟨*PrereqPermission*⟩ ::= '`assign-permission`' ⟨*permission1*⟩ '`prerequisite`' ⟨*permission2*⟩ (3)

The syntax uses keywords for defining a prerequisite policy either at the role (keyword '`assign-role`' in rule 2) or at permission level (keyword '`assign-permission`' in rule 3). In rule 2, ⟨*role2*⟩ corresponds to the precondition for the assignment of ⟨*role1*⟩. Similarly, in rule 3, ⟨*permission2*⟩ corresponds to the precondition for the assignment of ⟨*permission1*⟩. For example, the prerequisite policy on role assignment PL1 is expressed in GemRBAC-DSL as:

```
PL1: assign-role trainee prerequisite participant;
```

4.3 Cardinality policy

A cardinality policy defines a bound on the cardinality of role activation and assignment relations [2]. Its syntax is:

⟨*Cardinality*⟩ ::= ⟨*CardActivation*⟩ | ⟨*CardUser*⟩ | ⟨*CardPermission*⟩ | ⟨*CardRoleToUser*⟩ | ⟨*CardRoleToPermission*⟩ (1)

⟨*CardActivation*⟩ ::= '`maxActiveRoles =`' ⟨*integer*⟩ (2)

⟨*CardUser*⟩ ::= '`maxUsers =`' ⟨*integer*⟩ ['`only-for-role`' ⟨*role*⟩] (3)

⟨*CardPermission*⟩ ::= '`maxPermissions =`' ⟨*integer*⟩ ['`only-for-role`' ⟨*role*⟩] (4)

⟨*CardRoleToUser*⟩ ::= '`maxRoles-User =`' ⟨*integer*⟩ ['`only-for-user`' ⟨*user*⟩] (5)

missions *are not* specified with GemRBAC-DSL. We assume that these assignments are defined in the RBAC system on which the policies are going to be enforced.

⟨*CardRoleToPermission*⟩ ::= '`maxRoles-Permission =`' ⟨*integer*⟩ (6)
['`only-for-permission`' ⟨*Permission*⟩]

GemRBAC-DSL supports five types of cardinality policies: maximum number of active roles within a session (rule 2), maximum number of users assigned to a role (rule 3), maximum number of permissions assigned to a role (rule 4), maximum number of roles assigned to a user (rule 5), maximum number of roles assigned to a permission (rule 6). In rules 2–6, ⟨*integer*⟩ represents the cardinality bound. In rules 3–6, if the optional element is omitted, it means that the bound will apply, respectively, to all roles (rules 3–4), all users (rule 5), all permissions (rule 6). For example, the cardinality policy on user-to-role assignment PL2 is expressed in GemRBAC-DSL as:

```
PL2: maxUsers = 3 only-for-role assistant;
```

4.4 Precedence and dependency policies

A precedence policy establishes a precedence relationship between the enabling of a role and the activation of another one. A dependency policy restricts the deactivation of a role if another one is already active [24]. The syntax is:

⟨*PrecEnabling*⟩ ::= '`enable`' ⟨*role1*⟩ ' `if active`' ⟨*role2*⟩ (1)
['`,`' ⟨*timeShift*⟩] ['`deactivation-dependency`']

⟨*timeShift*⟩ := '`after`' ⟨*integer*⟩ ⟨*timeUnit*⟩ (2)

⟨*timeUnit*⟩ ::= '`second`' | '`minute`' | '`hour`' | '`day`' | '`week`' | '`month`' | '`year`' (3)

In rule 1, ⟨*role2*⟩ denotes the role whose activation has to precede the enabling of the role denoted by ⟨*role1*⟩. An optional ⟨*timeShift*⟩ can be specified to define the amount of time that has to pass between the role enabling and the role activation events (rules 2–3). The optional keyword '`deactivation-dependency`' is used to express a dependency policy. For example, the precedence and dependency policy PL3 is expressed in GemRBAC-DSL as:

```
PL3: enable trainee if active admin deactivation-dependency;
```

4.5 Role hierarchy policy

A hierarchy policy states that assigning a role r (respectively, a permission p) to a user u (respectively, a role s) implies assigning to u (respectively, s) also all the junior roles of r (respectively, the sub-permissions of p) [23]. Its syntax is defined as:

⟨*Hierarchy*⟩ ::= '`trigger-`' (⟨*RoleHierarchy*⟩ | ⟨*PermissionHierarchy*⟩) (1)

⟨*RoleHierarchy*⟩ ::= '`role-hierarchy`' ⟨*role*⟩ (2)

⟨*PermissionHierarchy*⟩ ::= '`permission-hierarchy`' ⟨*permission*⟩ (3)

The syntax uses two different keywords for distinguishing between role hierarchy (rule 2) and permission hierarchy (rule 3). Notice that while the preamble of a GemRBAC-DSL specification declares the role and permission hierarchy relations for the system, a security analyst has to explicitly define a role hierarchy policy (for a role or permission) to put the hierarchy relation(s) into effect. For example, the role hierarchy policy PL4 can be expressed as:

```
PL4: trigger-role-hierarchy assistant;
```

4.6 Separation of duty policy

A separation of duty (SoD) policy defines a mutual exclusion relation between users, roles, or permissions; mutually-exclusive entities involved in a SoD relation are called *conflicting*. SoD can be static or dynamic.

4.6.1 Static Separation of duty (SSoD)

An SSoD policy restricts the assignment of mutually exclusive roles, users, or permissions [2,3]. Its syntax is:

⟨*SSoD*⟩ ::= ⟨*SSoDCR*⟩ | ⟨*SSoDCU*⟩ | ⟨*SSoDCP*⟩ (1)

⟨*SSoDCR*⟩ ::= '`conflicting-roles-assignment`' ⟨*role*⟩ ('`,`' ⟨*role*⟩)+ ['`on permission`' ⟨*permission*⟩] (2)

⟨*SSoDCU*⟩ ::= '`conflicting-users-assignment`' ⟨*user*⟩ ('`,`' ⟨*user*⟩)+ ['`on role`' ⟨*role*⟩] (3)

⟨*SSoDCP*⟩ ::= '`conflicting-roles-assignment`' ⟨*permission*⟩ ('`,`' ⟨*permission*⟩)+ ['`on role`' ⟨*role*⟩] (4)

SSoD policies can define conflicting roles (rule 2), conflicting users (rule 3), and conflicting permissions (rule 4). Rules 2–4 have an optional block that indicates that the SSoD policy is applied only when the roles are assigned to a specific permission (rule 2) and when the users (rule 3) or the permissions (rule 4) are assigned to a specific role. For example, the SSoD policy on conflicting roles PL5 is expressed in GemRBAC-DSL as:

```
PL5: conflicting-roles-assignment assistant, trainee;
```

4.6.2 Dynamic Separation of duty (DSoD)

A DSoD policy allows the assignment of conflicting roles but forbids their activation in the same session [25]. GemRBAC-DSL supports the specification of four types of DSoD: simple, object-based, operational-based, and history-based DSoD. We refer the reader to [7,25] for more details about these types of policies. The syntax for DSoD policies is similar to the one for SSoD policies but uses different keywords:

⟨*DSoD*⟩ ::= ⟨*DSoDCU*⟩ | ⟨*DSoDCP*⟩ | ⟨*DSoDCR*⟩ (1)

⟨*DSoDCU*⟩ ::= '`conflicting-users-activation`' ⟨*user*⟩ ('`,`' ⟨*user*⟩)+ ['`on role`' ⟨*role*⟩] (2)

⟨*DSoDCP*⟩ ::= '`conflicting-permissions-activation`' ⟨*permission*⟩ ('`,`' ⟨*permission*⟩)+ ['`on role`' ⟨*role*⟩] (3)

⟨*DSoDCR*⟩ ::= '`conflicting-roles-activation`' ⟨*role*⟩ ('`,`' ⟨*role*⟩)+ ['`depending-on-business-task-list`' ⟨*operation*⟩ ('`,`' ⟨*operation*⟩)+] ['`on-same-object`'] (4)

The optional keyword '`on-same-object`' in rule 4 is used to express an object-based DSoD policy. Similarly, the keyword '`depending-on-business-task-list`' followed by a list of ⟨*operation*⟩s is used to specify an operational-based DSoD. A history-based DSoD is defined by combining these two keywords. For example, the history-based DSoD policy PL6 is expressed in GemRBAC-DSL as:

```
PL6: conflicting-roles-activation assistant, admin
    depending-on-business-task-list create,read,update,delete
    on-same-object;
```

4.7 Binding of duty policy

A binding of duty (BoD) policy states that the operations of bounded permissions should be performed by the same role or subject [27]. Its syntax is:

⟨*BoD*⟩ ::= '`bounded-permissions`' ⟨*permission*⟩ ('`,`' ⟨*permission*⟩)+
('`role-BoD`' | '`subject-BoD`')

The syntax distinguishes between a role- or a subject-based policy with the two keywords '`role-BoD`' and '`subject-BoD`'. The bounded permissions are specified as a list of ⟨*permission*⟩s. For instance, the role-based BoD policy PL7 is expressed in GemRBAC-DSL as:

```
PL7: bounded-permissions add_casualty, modify_casualty,
    delete_casualty  role-BoD;
```

4.8 Delegation policy

A delegation policy allows a *delegator* (a user or any user assigned to a specific role) to delegate her role to *delegates* (the users or roles receiving the delegation). GemRBAC-DSL adopts the concepts of delegation presented in [13, 28] and integrated into the GemRBAC model [7], in which a delegation can be *single* or *multi-step*, *total* or *partial*, of type *grant* or *transfer*. A delegation of type *transfer* can be either *strong* or *weak*. Moreover, a *weak transfer* delegation can be of type *static* or *dynamic*. The syntax of a delegation policy is defined below:

⟨*Delegation*⟩ ::= ('`user` '⟨*user*⟩ | '`role` '⟨*role*⟩) '`can-delegate`' ⟨*role*⟩ (1)
('`to users`' ⟨*user*⟩ ('`,`' ⟨*user*⟩)* | '`to roles`'⟨*role*⟩ ('`,`' ⟨*role*⟩)*) '`as`'
('`total`' | '`partial with permissions (`'⟨*delegated-permissions*⟩'`)`') '`,`'
('`grant`' [⟨*duration*⟩] ('`single`' | '`multi-step`' ⟨*integer*⟩)
|'`transfer`' ('`strong`'|'`weak-static`'|'`weak-dynamic`'))

⟨*delegated-permissions*⟩ ::= ⟨*permission*⟩ ('`,`' ⟨*permission*⟩)* (2)

⟨*duration*⟩ ::= '`for`' ⟨*integer*⟩ ⟨*timeUnit*⟩ (3)

In the syntax, keywords '`user`' and '`role`' are used to denote the delegator. The keyword '`can-delegate`' denotes the ⟨*role*⟩ being delegated. The list of delegate ⟨*user*⟩s is denoted by the keyword '`to users`'; similarly, the keyword '`to roles`' denotes the list of delegate ⟨*role*⟩s. If the delegation is partial, the keyword '`partial-with-permissions`' denotes the list of ⟨*permission*⟩s being delegated. In the case of a multi-step delegation, the syntax requires to indicate the ⟨*integer*⟩ corresponding to the maximum number of delegation steps allowed. If the delegation is of type grant, a duration (denoted with the keyword '`for`', rule 3) can be optionally specified to indicate the amount of time after which the delegation is automatically revoked. For example, the delegation policy PL8 defines a delegation that is *multi-step* (with a maximum delegation depth of 2), *total* (because all the permissions of the delegated role have to be delegated), of type *grant* (because the delegator is still allowed to execute the permissions associated with the delegated role), with a *duration* of at most two weeks. This policy is expressed in GemRBAC-DSL as:

```
PL8: role admin can-delegate admin to roles assistant as total,
     grant for 2 week, multistep 2;
```

4.9 Revocation policy

A revocation policy allows a user or a role to revoke a delegation. GemRBAC-DSL supports the concept of revocation presented in [28] and integrated into the GemRBAC model [7], in which a revocation can be *grant-dependent* or *grant-independent*, *strong* or *weak* and, *cascading* or *non-cascading*. Its syntax is defined as:

⟨*Revocation*⟩ ::= (‘`user`’ ⟨*user*⟩ | ‘`role`’ ⟨*role*⟩ | ‘`delegator`’)
‘`can-revoke-delegation`’ ⟨*id*⟩
(‘`from users`’ ⟨*user*⟩ (‘`,`’ ⟨*user*⟩)* | ‘`from roles`’ ⟨*role*⟩ (‘`,`’ ⟨*role*⟩)*)) ‘`as`’
(‘`strong`’ | ‘`weak`’) ‘`,`’ (‘`nonCascading`’ | ‘`cascading`’)

The syntax allows for specifying who can revoke a certain delegation; the keywords ‘`user`’ and ‘`role`’ denote, respectively, an explicit user or role, while the keyword ‘`delegator`’ implicitly refers to the user or role that originally performed the delegation. The delegation that is being revoked is referenced through its identifier, preceded by the keyword ‘`can-revoke-delegation`’. The keyword ‘`from users`’ denotes the list of ⟨*users*⟩ from which the delegation is revoked; similarly, the keyword ‘`from roles`’ denotes the list of ⟨*roles*⟩ from which the delegation will be revoked. The additional keywords that come after the keyword ‘`as`’ indicate the type of revocation. For example, the revocation policy PL9 is defined as *weak* (because it will not impact the other roles possibly acquired through a role hierarchy) and as *non-cascading* (because it will not affect the further delegations performed along a delegation chain). This policy is expressed in GemRBAC-DSL as:

```
PL9: role admin can-revoke-delegation PL8 from roles assistant as
     weak, nonCascading;
```

4.10 Contextual policy

A contextual policy allows (or disallows) a user to be a member of a role or to perform an operation according to her context, i.e., depending on the current time [19] and/or location [10]. The syntax for this policy is defined as follows:

⟨*ContextPolicy*⟩ ::= ⟨*RoleContextPolicy*⟩ | ⟨*PermContextPolicy*⟩ (1)

⟨*RoleContextPolicy*⟩ ::= ‘`role-context`’ ⟨*role*⟩ (2)
(⟨*activeDuration*⟩
| ((‘`assign`’ | ‘`unassign`’) [‘`to user`’ ⟨*user*⟩] ⟨*context*⟩
| (‘`enable`’ | ‘`disable`’) ⟨*context*⟩
[‘`,`’ (‘`assign`’ | ‘`unassign`’) [‘`to user`’ ⟨*user*⟩] ⟨*context*⟩])
[‘`,`’ ⟨*activeDuration*⟩]))

⟨*activeDuration*⟩ :: = ‘`activation`’ (3)
(‘`duration`’ ⟨*integer*⟩ ⟨*timeUnit*⟩
| ‘ `cumulative duration =` ’ ⟨*integer*⟩ ⟨*timeUnit*⟩ ‘`,`’
‘`reset =`’ (‘`none`’ | ⟨*periodicTime*⟩) ‘`,`’
‘`duration-per-session =`’ (‘`unlimited`’ | ⟨*integer*⟩ ⟨*timeUnit*⟩))

⟨*periodicTime*⟩ ::= '`every`' [⟨*integer*⟩] ⟨*timeUnit*⟩ (4)

⟨*PermContextPolicy*⟩ ::= '`permission-context`' ⟨*permission*⟩ (5)
(('`assign`' | '`unassign`') ['`to role`' ⟨*role*⟩] ⟨*context*⟩
| ('`enable`' | '`disable`') ⟨*context*⟩
['`,`' ('`assign`' | '`unassign`') ['`to role`' ⟨*role*⟩] ⟨*context*⟩])

⟨*context*⟩ ::= '`@`' (⟨*temporalContext*⟩ | ⟨*spatialContext*⟩ (6)
| ⟨*SpatioTemporalContext*⟩ ('`&&`' ⟨*SpatioTemporalContext*⟩)*)

⟨*SpatioTemporalContext*⟩ ::= ⟨*spatialContext*⟩ ⟨*temporalContext*⟩ (7)

A contextual policy can be specified either at the role (rule 2) or at the permission level (rule 5). At the role level, a contextual policy can define a) a bound for the sum of activation durations of a given role and/or, b) the context of role assignment and/or role enabling. An *activation duration* represents the amount of time during which a role is active. As shown in rule 3, an activation duration can be specified for a single session (denoted with keyword '`duration`') or for multiple sessions (denoted with keyword '`cumulative duration =`'). In the second case, a security analyst should specify a reset period (line 3 of rule 3) and a bound for the maximum duration per single session (line 4 of rule 3). The reset period corresponds to a specific period of time after which the cumulative duration is reinitialized to zero. This period is represented by a periodicity expression as indicated in rule 4. The keyword '`none`' is used to indicate the absence of a reset period (rule 3). Similarly, the keyword '`unlimited`' is used to indicate the absence of a bound for the activation duration per session (rule 3). In addition to the activation duration, a security analyst can specify if a role should be assigned/unassigned (possibly to a specific user, as denoted by the optional keyword '`to user`'), or if a role should be enabled/disabled in a specific ⟨*context*⟩(rule 2). Notice that the same policy can restrict both role enabling/disabling and assignment/unassignment as indicated by the optional part in line 4 of rule 2. Rule 5 is structured similarly to rule 2 (lines 1–4) but it is used for specifying the enabling/disabling and/or assignment/unassignment of permissions. As shown in rule 6, GemRBAC-DSL supports temporal, spatial and spatio-temporal context specifications preceded by the '`@`' symbol. Temporal and spatial policies will be illustrated in the next subsections, using the concepts of the GemRBAC+CTX model introduced in [8]. Since spatio-temporal specifications can be seen as the conjunction of a temporal policy and a spatial one, we will omit their description.

An example of a contextual policy on role activation with a reset period is PL10, which can be expressed in GemRBAC-DSL as:

```
PL10: role-context analyst activation cumulative duration = 4
    hour, reset = every day, duration-per-session = unlimited;
```

4.10.1 Policies with temporal context

The syntax for defining a temporal context is:

⟨*temporal*⟩ ::= '`time`' (⟨*absoluteTime*⟩ | ⟨*relativeTime*⟩
| (⟨*compositeTime*⟩ ('`&`' ⟨*compositeTime*⟩)*)

⟨*compositeTime*⟩ ::= ⟨*absoluteTime*⟩ ⟨*relativeTime*⟩

The type of temporal context supported by GemRBAC-DSL corresponds to the one defined in [8], which distinguishes between absolute and relative time expressions.

An absolute time expression refers to a concrete point or interval in the timeline; conversely, a relative time expression cannot be mapped directly to a concrete point or interval in the timeline. Furthermore, absolute time and relative expressions can also be composed. The syntax of an absolute time expression is:

⟨*absoluteTime*⟩ ::= (1)
((⟨*date*⟩ [‘`at`’ ⟨*hour*⟩] | ‘`(`’ ⟨*date*⟩ (‘`,`’⟨*date*⟩)+‘`)`’)
|(‘`starting from`’ ⟨*date*⟩ [‘`at`’ ⟨*hour*⟩]
| ‘`[`’⟨*date*⟩‘`,`’⟨*date*⟩‘`]`’
| ‘`(`’ ‘`[`’⟨*date*⟩‘`,`’⟨*date*⟩‘`]`’ (‘`, [`’⟨*date*⟩‘`,`’⟨*date*⟩‘`]`’)+‘`)`’)
[⟨*periodicTime*⟩])

⟨*periodicTime*⟩ ::= ‘`every`’ [⟨*integer*⟩] ⟨*timeUnit*⟩ (2)

⟨*date*⟩ ::= ⟨*sDayOfMonth*⟩(‘`1`’-‘`9`’)(‘`0`’-‘`9`’)(‘`0`’-‘`9`’)(‘`0`’-‘`9`’) (3)

⟨*sDayOfMonth*⟩ ::= ⟨*integer*⟩ ⟨*sMonth*⟩ (4)

⟨*sMonth*⟩ ::= ‘`Jan`’ | ‘`Feb`’ | ‘`Mar`’ | ‘`Apr`’ | ‘`May`’ (5)
| ‘`June`’ | ‘`July`’ | ‘`Aug`’ | ‘`Sept`’ | ‘`Oct`’ | ‘`Nov`’ | ‘`Dec`’

⟨*hour*⟩ ::= ((‘`0`’-‘`1`’)(‘`0`’-‘`9`’) | (‘`2`’)(‘`0`’-‘`3`’)) ‘`:`’ (6)
(‘`0`’-‘`5`’) (‘`0`’-‘`9`’) ‘`:`’ (‘`0`’-‘`5`’) (‘`0`’-‘`9`’)

An absolute time expression can have different forms. The simplest form is captured by ⟨*date*⟩, which is composed of a day of the month ⟨*sDayOfMonth*⟩ and a year (rule 4). An ⟨*sDayOfMonth*⟩ denotes a day, represented as an ⟨*integer*⟩, and a month, represented as an ⟨*sMonth*⟩. The latter corresponds to the abbreviation for a specific month (rule 6). A ⟨*date*⟩ can be optionally followed by the ‘`at`’ keyword and an ⟨*hour*⟩, to represent a specific hour during a day[2]. An absolute time expression can also correspond to a list of ⟨*date*⟩s enclosed in round brackets. Another type of absolute time expression is represented by intervals. An unbounded time interval is specified with a ⟨*date*⟩ prefixed by the keyword ‘`starting from`’. A bounded time interval is represented as two ⟨*date*⟩s enclosed in square brackets. Lists of bounded intervals are enclosed in round brackets. Unbounded and bounded time intervals as well as lists of bounded time intervals can be followed by a periodicity expression (denoted with the keyword ‘`every`’, see rule 2), which specifies how often, during the selected interval(s), the action determined by the policy (e.g., enabling a role) should be in effect. For example, the role enabling policy PL11 can be expressed as:

```
PL11: role-context  participant enable @time  [12 Feb 2016, 8 Jun
    2016];
```

A relative time expression is a time expression that cannot be mapped directly to a concrete point or interval in the timeline. The syntax of a relative time expression is:

⟨*relativeTime*⟩ ::= ((⟨*iHour*⟩ (‘`,`’ ⟨*iHour*⟩)*)
| (⟨*dayOfMonthH*⟩ (‘`and @ time`’ ⟨*dayOfMonthH*⟩)*)
| (⟨*dayOfWeekH*⟩ (‘`and @ time`’ ⟨*dayOfWeekH*⟩)*)
| (⟨*monthDayOfWeekH*⟩
(‘`and @ time`’ ⟨*monthDayOfWeekH*⟩)*))

[2]The current version of GemRBAC-DSL does not support the concept of time zone.

A relative time expression can have different forms. The first form is as a list of hour intervals, which are intervals whose start and end points are hours. The syntax of an hour interval is:

⟨*iHour*⟩ ::= '`from`' ⟨*hour*⟩'`to`' ⟨*hour*⟩ (1)
[('`excluding (`' ⟨*exHour*⟩ ('`,`' ⟨*exHour*⟩)* '`)`']

⟨*exHour*⟩ ::= '`from`' ⟨*hour*⟩'`to`' ⟨*hour*⟩ (2)

Within the definition of an ⟨*iHour*⟩, one can also specify a list of hour intervals to be excluded, denoted with the keyword '`excluding`' (rule 2).

A relative time expression can be also defined as a list of expressions starting with a day of month (⟨*dayOfMonthH*⟩s). This expression corresponds to a day of month (⟨*dayOfMonth*⟩) that optionally overlays an hour interval. The syntax of a relative expression with a day of month is:

⟨*dayOfMonthH*⟩ ::= ⟨*dayOfMonth*⟩('`,`'⟨*dayOfMonth*⟩)* (1)
[(⟨*iHour*⟩ ('`,`' ⟨*iHour*⟩)*)]

⟨*dayOfMonth*⟩ ::= ⟨*sDayOfMonth*⟩ | ⟨*iDayOfMonth*⟩ (2)

⟨*iDayOfMonth*⟩ ::= '`from`' ⟨*sDayOfMonth*⟩ '`to`' (3)
⟨*sDayOfMonth*⟩ ['`excluding (`' ⟨*exDayOfMonth*⟩
('`,`' ⟨*exDayOfMonth*⟩)* '`)`']

⟨*exDayOfMonth*⟩ ::= ⟨*sDayOfMonth*⟩|⟨*exIDayOfMonth*⟩ (4)

⟨*exIDayOfMonth*⟩ ::= '`from`' ⟨*sDayOfMonth*⟩ '`to`' (5)
⟨*sDayOfMonth*⟩

A day of month can correspond to a single day (⟨*sDayOfMonth*⟩, see page 15) or an interval of days of month (⟨*iDayOfMonth*⟩) (rule 2). The latter can also be defined to exclude a single day of month or an interval of days of month ⟨*exIDayOfMonth*⟩; notice that exclusion is not recursive.

A relative time expression can also have the form of a list of ⟨*dayOfWeekH*⟩s. The latter is a day of week that optionally overlays an hour interval. The syntax of a relative expression with a day of week is:

⟨*dayOfWeekH*⟩ ::= ⟨*dayOfWeek*⟩ ('`,`' ⟨*dayOfWeek*⟩)* (1)
[⟨*iHour*⟩ ('`,`' ⟨*iHour*⟩)*]

⟨*dayOfWeek*⟩ ::= ⟨*sDayOfWeek*⟩ | ⟨*iDayOfWeek*⟩ (2)

⟨*sDayOfWeek*⟩ ::= [['`on`'] '`the`' ⟨*integer*⟩] ('`Monday`' (3)
| '`Tuesday`' | '`Wednesday`' | '`Thursday`' | '`Friday`' | '`Saturday`' | '`Sunday`')

⟨*iDayOfWeek*⟩ ::= '`from`' ⟨*sDayOfWeek*⟩ '`to`' (4)
⟨*sDayOfWeek*⟩ ['`excluding (`' ⟨*exDayOfWeek*⟩
('`,`' ⟨*exDayOfWeek*⟩)* '`)`']

⟨*exDayOfWeek*⟩ ::= ⟨*sDayOfWeek*⟩ | ⟨*exIDayOfWeek*⟩ (5)

⟨*exIDayOfWeek*⟩ ::= '`from`' ⟨*sDayOfWeek*⟩ '`to`' (6)
⟨*sDayOfWeek*⟩

This syntax follows a pattern similar to the ones seen above. For example, the time-based policy on permission assignment PL12 is expressed in GemRBAC-DSL as:

```
PL12: permission-context add_casualty assign to role trainee
    @time from Monday to Friday from 08:00:00 to 17:00:00;
```

A relative time expression can be also defined as a set of ⟨*monthDayOfWeekH*⟩s. The latter is a list of ⟨*month*⟩s that optionally overlays a ⟨*dayOfMonthH*⟩ or an ⟨*iHour*⟩. The syntax of ⟨*monthDayOfWeekH*⟩ is:

⟨*monthDayOfWeekH*⟩ ::= ⟨*month*⟩ ('`,`' ⟨*month*⟩)* (1)
[('`#`' ⟨*dayOfWeekH*⟩)+
|(⟨*iHour*⟩ ('`,`' ⟨*iHour*⟩)*)]

⟨*month*⟩ ::= ⟨*sMonth*⟩ | ⟨*iMonth*⟩ (2)

⟨*iMonth*⟩ ::= '`from`' ⟨*sMonth*⟩ '`to`' ⟨*sMonth*⟩ (3)
['`excluding (`' ⟨*exMonth*⟩ ('`,`' ⟨*exMonth*⟩)* '`)`']

⟨*exMonth*⟩ ::= ⟨*sMonth*⟩ | ⟨*exIMonth*⟩ (4)

⟨*exIMonth*⟩ ::= '`from`' ⟨*sMonth*⟩ '`to`' ⟨*sMonth*⟩ (5)

Also this syntax follows the same structure of the previous definitions. Notice that in this case, the list of ⟨*month*⟩s can overlay either a list of ⟨*iHour*⟩s or a list of ⟨*dayOfWeekH*⟩s.

An ⟨*sDayOfWeek*⟩ can contain an index (represented as an ⟨*integer*⟩), which refers to a specific occurrence of a day, as in "on the first Monday" (of a month).

4.10.2 Policies with spatial context

The syntax for defining a spatial context is:

⟨*spatial*⟩ ::= '`location`' ⟨*location*⟩ ('`,`' ⟨*location*⟩)* (1)

⟨*location*⟩ ::= [relativeLocation] ('`physical`' ⟨*physicalLocation*⟩ (2)
| '`geofence`' ⟨*geofence*⟩)

⟨*physicalLocation*⟩ ::= ⟨*point*⟩ | ⟨*polygon*⟩ | ⟨*circle*⟩ | ⟨*userPos*⟩ (3)

⟨*point*⟩ ::= '`(lat`' ⟨*float*⟩'`: long`' ⟨*float*⟩'`: alt`' ⟨*float*⟩'`)`' (4)

⟨*userPos*⟩ ::= '`position`' ⟨*user*⟩ (5)

⟨*circle*⟩ ::= '`center`' ⟨*point*⟩ '`radius`' ⟨*float*⟩ ⟨*locUnit*⟩ (6)

⟨*polygon*⟩ ::= ⟨*polyline*⟩ ⟨*polyline*⟩ ('`,`' ⟨*polyline*⟩)+ (7)

⟨*polyline*⟩ ::= '`line {`' ⟨*point*⟩ '`,`' ⟨*point*⟩ '`}`' (8)

⟨*relativeLocation*⟩ ::= [⟨*integer*⟩ ⟨*locUnit*⟩] ⟨*direction*⟩ (9)

⟨*locUnit*⟩ ::= '`miles`' | '`meters`' | '`kilometers`' (10)

⟨*direction*⟩ ::= ⟨*cardinalDir*⟩ | ⟨*qualitativeDir*⟩ (11)

⟨*cardinalDirection*⟩ ::= ('`N`' | '`E`' | '`S`' | '`W`' | '`NE`' | '`SE`' | '`SW`' |'`NW`') (12)
| '`degree`' ⟨*integer*⟩

⟨*qualitativeDirection*⟩ ::= '`inside`' |'`outside`' |'`around`' (13)

The spatial context in GemRBAC-DSL is represented as a set of locations. The concept of location is taken from [8]: it is a bounded area or a point in space. Reference [8] further classifies locations as physical (a precise position in a geometric space) and logical (a symbolic abstraction of one or many physical locations). Physical locations are denoted in GemRBAC-DSL with the keyword '`physical`', while the keyword '`geofence`' denotes logical locations. Notice that the identifiers that can be used as logical locations are those declared in the preamble under the rule ⟨*geofences*⟩.

The simplest type of physical location is a ⟨*point*⟩, i.e., a set of geographic coordinates denoted with the keywords '`lat`', '`long`', and '`alt`', corresponding to latitude,

longitude, and altitude (rule 4). Each coordinate is expressed as a floating-point number. The keyword '`position`' followed by a user id (rule 5) is used to define a location in terms of the coordinates of a user. Bounded physical locations can have the shape of a circle or of a polygon. A ⟨*circle*⟩ is denoted with a '`center`' and a '`radius`'; the latter is specified using units of length (see rules 6 and 10). A polygon is defined in terms of polylines, which are denoted with the keyword '`line`' and a start and an end ⟨*point*⟩ (rules 7–8). For example, the location-based policy on role enabling PL13 is expressed in GemRBAC-DSL as:

```
PL13: role-context enable admin @location physical
line {(lat 15 : long 24 : alt 200),
      (lat 20 : long 27 : alt 200)},
line {(lat 20 : long 27 : alt 200),
      (lat 17 : long 27 : alt 200)},
line {(lat 17 : long 27 : alt 200),
      (lat 15 : long 27 : alt 200)},
line {(lat 15 : long 27 : alt 200),
      (lat 15 : long 24 : alt 200)};
```

As shown in rule 2, both physical and logical locations can be optionally prefixed by ⟨*relativeLocation*⟩, which represents a location defined with respect to another one. A ⟨*relativeLocation*⟩ is expressed with a ⟨*direction*⟩ and an optional distance expressed with a unit of length (rule 9). A direction of type ⟨*cardinalDirection*⟩ is denoted with symbols corresponding to cardinal and ordinal directions or with the degrees of rotation (denoted with the '`degree`' keyword followed by an integer) on a compass (rule 12). A direction of type ⟨*qualitativeDirection*⟩ represents a relative proximity to a location and is defined using the keywords '`inside`', '`outside`', or '`around`' (rule 13). For example, the contextual policy PL14, which contains a relative location, is expressed in GemRBAC-DSL as:

```
PL14: role-context trainee enable @location 100 meters inside
    geofence Zone1;
```

5 Semantic Checks

A security analyst can erroneously write policies that are inconsistent or conflicting. In the following paragraphs we describe all the possible conflicts that can be found in a GemRBAC-DSL specification. We mainly focus on inter-policy conflicts, i.e., global conflicts between different policies. The Eclipse-based editor for GemRBAC-DSL includes semantic checks for these conflicts, which are then reported to the user as errors or warnings.

Prerequisite role and SSoD on conflicting roles policies. Let PR be the set of roles involved in a prerequisite role policy, and SCR be the set of conflicting roles in a SSoDCR policy. If $PR \subseteq SCR$, the two policies are in conflict. The reason is that, while the prerequisite role policy requires the assignment of two roles to the same user (in a certain order), the SSoDCR policy prohibits this assignment. This situation can be avoided by not specifying prerequisite role policies and SSoDCR policies for the same subset of roles. This conflict is reported as an error. The conflict between the prerequisite permission policy and the SSoDCP one is defined in a similar way.

Prerequisite role and Role hierarchy policies. Let PR be the set of roles in a prerequisite role policy, and RH be the set $\{r\} \cup juniors(r)$ in a role hierarchy policy,

where *junior*() is a function that returns the junior roles of its argument. If $PR \subseteq RH$, the prerequisite role and the role hierarchy policies will require the assignment of the same subset of roles. Hence there is no need to define a prerequisite policy between a role and its parent role. This conflict is reported as a warning. The conflict between the prerequisite permission policy and the permission hierarchy one is defined similarly.

Cardinality (role-to-user assignment) and Role hierarchy policies. Let n be the number of juniors of role r in a role hierarchy policy, and *maxRoles* be the maximum number of roles that can be assigned to a user, as specified by a cardinality policy. If $n \geq maxRoles$, the cardinality policy will be violated. This situation can be avoided by having *maxRoles* greater than the number of juniors of any role. This conflict is reported as an error. The conflict between the cardinality (role-to-permission assignment) policy and the permission hierarchy one is defined similarly.

Cardinality (permission-to-role assignment) and Binding of duty policies. Let n be the number of bounded permissions in BoD policy, and *maxPerm* be the maximum number of permissions that can be assigned to a role, as specified by a cardinality policy. If $n > maxPerm$, the cardinality policy will be violated, because the BoD policy will require a role to be assigned to more than *maxPerm* permissions. This situation can be avoided by having *maxPerm* be equal or greater than the number of bounded permissions in a BoD policy. This conflict is reported as an error.

Role hierarchy and SSoD on conflicting roles policies. Let RH be the set $\{r\} \cup juniors(r)$ in a role hierarchy policy, where *junior*() is a function that returns the junior roles of its argument; let SCR be the set of conflicting roles in an SSoDCR policy. If $|RH \cap SCR| > 1$ the two policies are in conflict. Indeed, while the role hierarchy policy requires the assignment of a set of roles, the SSoDCR policy prohibits this assignment. To avoid this situation an SSoDCR policy should not contain a role and its junior(s) or, similarly, two juniors of the same role. This conflict is reported as an error. The conflict between the permission hierarchy policy and the SSoDCP one is defined similarly.

Role hierarchy and Context (role unassignment) policies. Let JRH be the set containing the juniors of role r. If a context policy on role un-assignment is specified for any role $s \in JRH$, the role hierarchy policy will be violated. Indeed, while the role hierarchy requires the assignment of a junior of role r, the role context policy can prohibit this assignment. This conflict is reported as an error. The conflict between the permission hierarchy and context-based (permission assignment) policies is defined similarly.

SSoD and DSoD on conflicting roles policies. Let SCR and DCR be the sets of, respectively, conflicting roles in an SSoDCR policy and a DSoDCR one. If $|SCR \cap DCR| > 1$, the assignment of at least two conflicting roles will be allowed by the DSoDCR policy but forbidden by the SSoDCR policy, generating an inconsistency in the system. This conflict is reported as a warning. The conflict between the SSoD and DSoD on conflicting users (or permission) policies is defined similarly. Notice that an SSoDCU policy and a DSoDCU one with the same list of users on different roles are not conflicting.

SSoD on conflicting permissions and Binding of duty policies. Let SCP be the set of conflicting permissions in an SSoDCP policy and let $PBoD$ be the set of bounded permissions in a BoD policy. If $|SCP \cap PBoD| > 1$, the two policies are in conflict. Indeed, while the SSoDCP restricts the assignment of at least two conflicting permissions, the BoD policy requires this assignment. To avoid this situation, an SSoDCP policy should not contain permissions that are used in a BoD policy. This conflict is reported as an error.

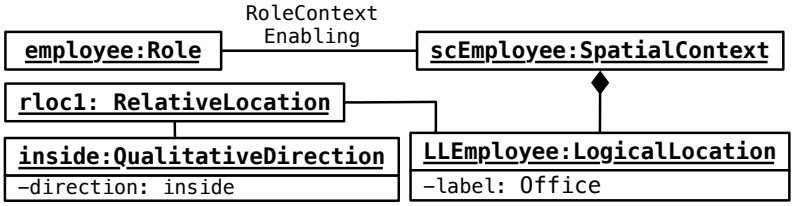

Figure 2: A fragment of an instance of the GemRBAC+CTX model

Delegation and SSoD on conflicting roles policies. Let SCR be the set of conflicting roles in an SSoDCR policy, r be the role being delegated, and $RECR$ be the set of roles that will receive the delegation in a delegation policy. If $(\{r\} \cup RECR) \subseteq SCR$, the two policies are in conflict. The reason is that, while the delegation policy allows the assignment of a set of roles to the same user, the SSoDCR policy prohibits this assignment. This conflict is reported as an error.

Additional checks. The editor also detects overlapping intervals in policies with temporal context, and circular dependencies for role hierarchy and precedence policies.

6 Semantics

The GemRBAC+CTX model (as well as its non-contextual ancestor GemRBAC), which is the conceptual RBAC model on top of which GemRBAC-DSL has been designed, comes with an operationalization of the semantics of the policies it supports. The operationalization follows a model-driven approach, by which the semantics of each RBAC policy is expressed as an OCL constraint on the RBAC model. Since the GemRBAC+CTX model and GemRBAC-DSL have the same expressiveness, we can define the semantics of GemRBAC-DSL by mapping its constructs to the corresponding OCL constraints defined for the GemRBAC+CTX model. In the rest of this section we sketch this mapping; we refer the reader to [7, 8] for the details on the structure of the GemRBAC+CTX model.

Each entity in the ⟨*preamble*⟩ of a GemRBAC-DSL specification corresponds to an instance of a UML class in the GemRBAC+CTX model: users, roles, permissions, operations, and logical locations (⟨*geofences*⟩) are mapped to instances of the homonymous classes in GemRBAC+CTX. Similarly, role and permission hierarchies correspond to the homonymous associations in the GemRBAC+CTX model.

Each type of RBAC policy is mapped to the corresponding OCL constraint template defined in the GemRBAC+CTX model; in each template the symbolic parameters are replaced with the actual entities used in the specification. For instance, the semantics of the object-based DSoD policy

```
objDSoD: conflicting-roles-activation author, reviewer
    on-same-object;
```

can be defined by the OCL invariant `DSoD` of the class `Session` (see [7], §7.5.2), by replacing the parameters `r1` and `r2` with roles `author` and `reviewer`.

Regarding contextual policies, the context to be assigned/enabled (as prescribed by the policy) is represented in the GemRBAC+CTX model, as an association with the corresponding role/permission. For example, consider the policy

```
loc: role-context enable employee only @location inside office;
```

which enables role *employee* only inside the logical location denoted by the label "office". Figure 2 depicts an excerpt of an instance of the GemRBAC+CTX model in which role `employee` is associated to a `SpatialContext` object that contains the object *LLEmployee* of type `LogicalLocation`, which denotes the location "office". This object is associated with object *rloc1* of type `RelativeLocation`, which contains a `QualitativeDirection`. The policy `loc` can be mapped to the OCL invariant `relativeLocationRoleEnabling` of class `Session` (see [8], §4.2), parametrized with role `employee`.

Expressing the semantics of GemRBAC-DSL policies as OCL constraints on the GemRBAC+CTX model enables the users of the language to benefit from the model-driven policy enforcement mechanisms described in [7, 8]. Briefly, making an access decision for a policy can be reduced to checking the corresponding OCL constraint on a instance of the GemRBAC+CTX model, which represents a snapshot of the system at a certain time.

Table 2: Mapping of GemRBAC-DSL constructs to OCL constraints on the GemRBAC+CTX model

Type of policy	OCL constraint	ref
⟨*PrereqRole*⟩	`context User :: assignRole(r:Role): pre PreqRole`	[7]
⟨*PrereqPermission*⟩	`context Role :: assignPermission(p: Permission): pre PreqPermisssion`	[7]
⟨*CardActivation*⟩	`context Session inv Cardinality`	[7]
⟨*CardUser*⟩	`context User inv Cardinality`	[7]
⟨*CardPermission*⟩	This policy is expressed in a similar way as the previous one by replacing the context of `User` with the context of `Permission`.	[7]
⟨*CardRoletoUser*⟩	`context Role inv Cardinality`	[7]
⟨*CardRoletoPermission*⟩	This policy is expressed in a similar way as the previous one by replacing the instances of `users` with instances of `permissions`.	[7]
⟨*PrecEnabling*⟩	`context Session :: enableRole(r:Role): pre RoleEnablingPrecedence`	[7]
Dependency ⟨*PrecEnabling*⟩	`context Session :: deactivateRole(r:Role): pre RoleActivationDependency`	[7]
⟨*RoleHierarchy*⟩	`context User :: assignRole(r:Role): post RoleHierarchy`	[7]
⟨*PermissionHierarchy*⟩	`context Role :: assignPermission(p: Permission): post RoleHierarchy`	[7]
⟨*SSoDCU*⟩	`context Role inv SSoDCU`	[7]
⟨*SSoDCR*⟩	`context User inv SSoDCR` `context Role inv SSoDCP2`	[7]
⟨*SSoDCR*⟩	`context User inv SSoDCR` `context Role inv SSoDCP2`	[7]
⟨*SSoDCP*⟩	`context Role inv SSoDCP1`	[7]
⟨*DSoDCR*⟩	`context Session inv DSoD`	[7]

⟨*DSoDCU*⟩	`context Role inv DSoDCU`	web1
⟨*DSoDCP*⟩	`context Role inv DSoDCP`	web1
⟨*DSoDCR*⟩	`context Session :: performOperation(op: Operation, p:Permission, r:Role): pre ObjectDSOD`	[7]
⟨*DSoDCR*⟩	`context Session inv OperationalDSoD`	[7]
⟨*DSoDCR*⟩	`context Session :: performOperation(op: Operation, p:Permission, r:Role): pre HistoryDSOD`	[7]
Role-based ⟨*BoD*⟩	`context Session :: performOperation(op: Operation, p:Permission, r:Role) pre RoleBoD`	[7]
Subject-based ⟨*BoD*⟩	`context Session :: performOperation(op: Operation, p:Permission, r:Role) pre SubjectBoD`	[7]

⟨*Delegation*⟩	`context Delegation inv TotalDelegation`
	`context Delegation inv MultiStepDelegation`
	`context delegation inv PartialDelegation`
	`context Delegation inv StrongTransfer`
	`context Delegation inv StaticWeakTransfer`
	`context Delegation inv DynamicWeakTransfer`
	`context Delegation inv AutomaticRevocation`
		[7]

⟨*Revocation*⟩	`context Delegation :: revoke() pre RevacationDependency`
	`context Delegation :: revoke() post StrongRevocation`
	`context Delegation :: revoke() post CascadingRevocation`
		[7]

TPA with ⟨*absoluteTime*⟩	`context Session inv AbsoluteBTIRoleEnab`
	`context Permission inv AbsoluteBTIPermAssign`
	`context Role inv AbsoluteTPRoleAssign`
	`context Role inv AbsoluteUBIRoleAssign`
		[8] web2

TPA with ⟨*periodicTime*⟩	`context Role inv periodicUnboundTIRoleAssign`	[8]
TPA with ⟨*activeDuration*⟩	`context Session inv DurationAbsoluteBTIRoleEnab`	[8]
TPRInd ⟨*sDayOfWeek*⟩	`context Role inv indexRoleAssign`	[8]
TPRH ⟨*iHour*⟩	`context Role inv RelativeHoursRoleAssign`	web2
TPRDM ⟨*dayOfMonthH*⟩	`context Role inv DayOfMonthHoursRoleAssign` `context Permission inv DayOfMonthHoursPermAssign`	web2
TPRDW ⟨*dayOfWeekH*⟩	`context Permission inv DayOfWeekHourPermAssign`	[8]
TPRMD ⟨*monthDayOfWeekH*⟩	`context Role inv MonthDayOfWeekHourRoleAssign`	web2

TPCT ⟨*compositeTime*⟩	This policy can be checked by a logical conjunction of two temporal policies: one with absolute time and one with relative time.	[8]
SPP ⟨*physicalLocation*⟩	`context Role inv physicalLocationRoleAssign`	[8]
SPL ⟨*geofence*⟩	This policy can be checked in a similar way as the previous one by replacing the instances of `PhysicalLocation` with instances of `LogicalLocation`.	[8]
SPR ⟨*relativeLocation*⟩	`context Session inv relativeLocationRoleEnabling`	[8]
SPT⟨*SpatioTemporal*⟩	This policy can be checked by a logical conjunction of the spatial and temporal policies.	[8]

Legend. TP: temporal policy; TPA: TP with absolute time; TPR: TP with relative time; TPRInd: TPR containing an index; TPRH: TPR of type hour interval; TPRDM: temporal policy with a relative time of type day of month that optionally overlays hours; TPRDW: TPR of type day of week that optionally overlays hours; TPRMD: TPR of type day of month that optionally overlays days of week (the days of week may optionally overlay hours); TPCT: TP with composite time; SP: spatial policy; SPP: SP with a physical location; SPL: SP with a logical location; SPR: SP with a relative location; SPT: spatio-temporal policy.

Table 2 describes the mapping of each RBAC policy supported by GemRBAC-DSL to its corresponding OCL constraint(s) defined on the GemRBAC+CTX model. The first column indicates the type of policy and the corresponding grammar rule. The second column denotes the corresponding OCL constraints, whose full definition can be found in the reference indicated in the third column. The reference "web1" and "web2" are the websites `https://github.com/AmeniBF/GemRBAC-model` and `https://github.com/AmeniBF/GemRBAC-CTX-model.git`, respectively.

7 Discussion

Policy specification languages vs RBAC models. GemRBAC-DSL is a domain-specific specification language, built on top of the GemRBAC+CTX model, with the goal of providing a high-level specification language for the policies that can be defined using GemRBAC+CTX. The constructs included in the language have been derived from the corresponding concepts defined in GemRBAC+CTX. In this sense, GemRBAC-DSL does not define new concepts related to RBAC; instead, it provides a practical way to express RBAC policies using the concepts provided by an expressive model like GemRBAC+CTX. Although in our previous work [8] we reported on the use of OCL for the specification of RBAC policies based on GemRBAC+CTX, we also mentioned the impracticality of such an approach and expressed the need for a higher-level specification language.

Adoption. GemRBAC-DSL has been used by our industrial partner for the specification of the RBAC policies of a production-grade Web application. The adoption of GemRBAC-DSL has allowed its engineers to easily specify all the policies for their system, including 19 new types of contextual policies. Despite the fact that some constructs of the language are non-trivial, the engineers were able to use GemRBAC-DSL confidently after three half-day training sessions.

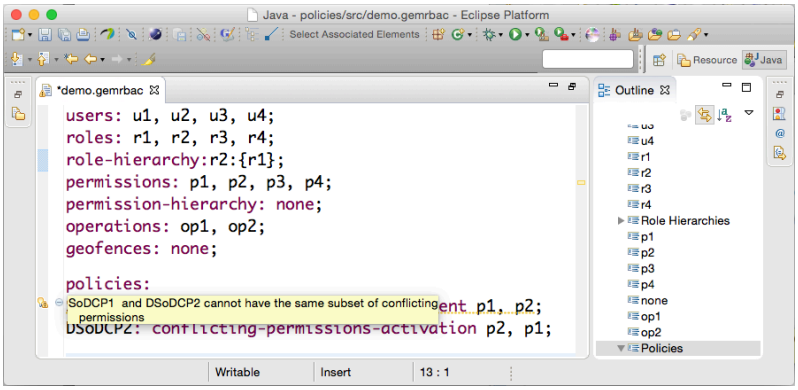

Figure 3: The GemRBAC-DSL editor

Tool Support. The GemRBAC-DSL editor has been implemented as an Eclipse plugin. We used Xtext 2.8 to define the textual syntax and the semantic checks (illustrated in section 5) for the language. As can be seen in Fig. 1, the editor supports syntax highlighting and conflict detection. Furthermore it performs also syntactic checks, such as detecting duplicated items in lists, or verifying that the identifiers of the entities (e.g., roles, users) used in the policies have been declared in the preamble. The editor is publicly available at `https://github.com/AmeniBF/GemRBAC-DSL.git`.

Limitations and Design Trade-offs. GemRBAC-DSL can express *all and only* the types of policies supported by its underlying model, GemRBAC+CTX. Since GemRBAC+CTX is quite an expressive model, GemRBAC-DSL includes many constructs that could have increased its level of complexity, hindering its adoption. Designing a simpler language would have implied providing limited support in terms of policy types, leading to partial fulfillment of our expressiveness requirements and a limited advance in terms of the state of the art. Hence, at the language design stage, we decided to pursue our expressiveness requirements, and to provide a syntax close to natural language to favor the adoption among practitioners and compensate (also by means of a rich editor) for the complexity of the language.

8 Conclusions and Future Work

In this paper we presented GemRBAC-DSL, a domain-specific language that facilitates the specification and consistency checking of policies based on highly-expressive RBAC models. GemRBAC-DSL supports all types of policies captured by the GemRBAC+CTX model, a comprehensive model encompassing all proposed types of policies. We have shown how the language can be used to specify the RBAC policies of an industrial application with complex, context-aware policies. The semantics of GemRBAC-DSL has been defined with a mapping to an existing OCL formalization of the RBAC policies supported by GemRBAC+CTX. This mapping paves the way for automating the enforcement of policies specifications written in GemRBAC-DSL,

using a model-driven approach.

As part of future work, we plan to extend GemRBAC-DSL to support richer contextual policies, as well as administrative policies. We also plan to assess the usability of the language through user studies with practitioners.

9 Acknowledgments

The authors wish to thank Benjamin Hourte and his team from HITEC Luxembourg, as well as the anonymous referees for their valuable feedback. 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.

References

[1] D. Abi Haidar, N. Cuppens-Boulahia, F. Cuppens, and H. Debar. An Extended RBAC Profile of XACML. In *Proc. of SWS 2006*, pages 13–22. ACM, 2006.

[2] G.-J. Ahn. Specification and Cassification of Role-based Authorization Policies. In *Proc. of WETICE 2003*, pages 202–207. IEEE, 2003.

[3] G.-J. Ahn and R. Sandhu. Role-based Authorization Constraints Specification. *ACM Trans. Inf. Syst. Secur.*, 3(4):207–226, Nov. 2000.

[4] G.-J. Ahn and M. Shin. Role-based authorization constraints specification using Object Constraint Language). In *Proc. of WETICE 2001*, pages 157–162. IEEE, 2001.

[5] 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.

[6] A. Anderson. XACML profile for role based access control (RBAC). *OASIS Access Control TC committee draft*, 1:13, 2004.

[7] 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.

[8] A. Ben Fadhel, D. Bianculli, L. Briand, and B. Hourte. A Model-driven Approach to Representing and Checking RBAC Contextual Policies. In *Proc. of CODASPY2016*, pages 243–253. ACM, 2016.

[9] 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.

[10] 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.

[11] R. Bhatti, A. Ghafoor, E. Bertino, and J. B. D. Joshi. X-GTRBAC: An XML-based Policy Specification Framework and Architecture for Enterprise-wide Access Control. *ACM Trans. Inf. Syst. Secur.*, 8(2):187–227, May 2005.

[12] C. Cotrini, T. Weghorn, D. Basin, and M. Clavel. Analyzing first-order role based access control. In *Proc. of CSF2015*, pages 3–17. IEEE, July 2015.

[13] J. Crampton and H. Khambhammettu. Delegation in Role-based Access Control. *Int. J. Inf. Secur.*, 7(2):123–136, 2008.

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

[15] R. Ferrini and E. Bertino. Supporting RBAC with XACML+OWL. In *Proc. of SACMAT 2009*, pages 145–154. ACM, 2009.

[16] T. Finin, A. Joshi, L. Kagal, J. Niu, R. Sandhu, W. Winsborough, and B. Thuraisingham. ROWLBAC: Representing Role Based Access Control in OWL. In *Proc. of SACMAT 2008*, pages 73–82. ACM, 2008.

[17] M. Hitchens and V. Varadharajan. Tower: A Language for Role Based Access Control. In *Proc. of POLICY 2001*, volume 1995 of *LNCS*, pages 88–106. Springer, 2001.

[18] J. Joshi. Access-control language for multidomain environments. *Internet Computing, IEEE*, 8(6):40–50, Nov 2004.

[19] 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.

[20] Q. Ni and E. Bertino. xfACL: An Extensible Functional Language for Access Control. In *Proc. of SACMAT 2011*, pages 61–72. ACM, 2011.

[21] OASIS. eXtensible Access Control Markup Language (XACML) Version 2.0, 2005.

[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] B. Shafiq, A. Masood, J. Joshi, and A. Ghafoor. A Role-based Access Control Policy Verification Framework for Real-time Systems. In *Proc. of WORDS 2005*, pages 13–20. IEEE, February 2005.

[25] R. T. Simon and M. E. Zurko. Separation of Duty in Role-based Environments. In *Proc. of CSFW 1997*, pages 183–194. IEEE, 1997.

[26] K. Sohr, M. Kuhlmann, M. Gogolla, H. Hu, and G.-J. Ahn. Comprehensive two-level analysis of role-based delegation and revocation policies with UML and OCL. *Inf. Softw. Technol.*, 54(12):1396 – 1417, 2012.

[27] M. Strembeck and J. Mendling. Modeling Process-related RBAC Models with Extended UML Activity Models. *Inf. Softw. Technol.*, 53(5):456–483, May 2011.

[28] L. Zhang, G.-J. Ahn, and B.-T. Chu. A Rule-based Framework for Role-based Delegation and Revocation. *ACM Trans. Inf. Syst. Secur.*, 6(3):404–441, 2003.

[29] Z. Zhang, J. Xiao, H. Li, and Y. Geng. An Extended Permission-based Delegation Authorization Model. In *Proc. of CSSE 2008*, volume 3, pages 696–699, December 2008.