Comprehensive Two-level Analysis of Static and Dynamic RBAC Constraints with UML and OCL

Mirco Kuhlmann
University of Bremen
Computer Science Department
Database Systems Group
D-28334 Bremen, Germany
mk@informatik.uni-bremen.de

Karsten Sohr
University of Bremen
Center for Computing Technologies
D-28334 Bremen, Germany
sohr@tzi.de

Martin Gogolla
University of Bremen
Computer Science Department
Database Systems Group
D-28334 Bremen, Germany
gogolla@informatik.uni-bremen.de

***Abstract*—Organizations with stringent security requirements like banks or hospitals frequently adopt role-based access control (RBAC) principles to simplify their internal permission management. Authorization constraints represent a fundamental advanced RBAC concept enabling precise restrictions on access rights. Thereby, the complexity of the resulting security policies increases so that tool support for comfortable creation and adequate validation is required. We propose a new approach to developing and analyzing RBAC policies using UML for modeling RBAC core concepts and OCL to realize authorization constraints. Dynamic (i. e., time-dependent) constraints, their visual representation in UML and their analysis are of special interest. The approach results in a domain-specific language for RBAC which is highly configurable and extendable with respect to new RBAC concepts and classes of authorization constraints and allows the developer to validate RBAC policies in an effective way. The approach is supported by a UML and OCL validation tool.**

***Index Terms*—RBAC, Security, Reliability, Modeling, UML/OCL, Analysis**

I. Introduction

Role-based access control (RBAC) offers a framework for managing permissions respecting access to many kinds of resources. RBAC is today an established standard and is used in many areas with stringent security demands. One central advantage of RBAC is that high-level organizational rules can be described in a natural way [1]. The basic RBAC concepts embody a simple configuration of users, roles and permissions as well as user and permission assignments to roles. In order to address practical requirements, RBAC has been extended with respect to more advanced concepts like role delegation, role hierarchies or formal authorization constraints. In particular, those advanced RBAC concepts are an important means for laying out higher-level organizational rules. Typical rules enforce separation of duty (SoD). As pointed out in the literature [2], [3], history-based SoD is a flexible form of SoD which is often needed in practice. For example, in a banking application, a clerk may have the permissions to prepare and to authorize cheques, but once the clerk has prepared a cheque, she cannot authorize it any more. Various types of dynamic (in particular application specific) authorization constraints have been identified in [4], [5].

Typically, RBAC rules become complex in large organizations such as financial institutes or hospitals so that undesirable properties of the security policies, i. e., the specific role configurations and sets of authorization constraints, may arise. For example, an SoD constraint between two roles may be useless if a user can obtain the security-critical permissions through other roles with no SoD restriction [6]. As a consequence, comprehensive RBAC policies need to be thoroughly analyzed to ensure a correct realization of the underlying requirements. Dynamic properties are of special interest because many important constraints like history-based SoD regard both present as well as past or future activities so that specific sequences of resource accesses must be forbidden, e.g., after a preparation of a cheque by a clerk its authorization by the same clerk is not allowed.

The Unified Modeling Language (UML) in combination with the Object Constraint Language (OCL) provides a promising way for creating and analyzing RBAC policies, since both languages benefit from substantial tool support including model driven methods. We propose a UML description of the core RBAC concepts and supplemental authorization constraints representing an RBAC metamodel. The metamodel is meant as a basis for conceptual explanations within this paper. In practice, it can be enriched by further concepts and constraints depending on specific demands. But even this simple model allows administrators (i. e., security officers) entrusted with the permission management to specify and examine policies with respect to explicit and implicit static and dynamic properties. After a comprehensive validation a policy can be deployed and enforced by an authorization engine [7]. Our RBAC metamodel includes typical SoD constraints. However, as indicated before, the approach supports a broad variety of dynamic access control options including constraints specifically designed for a particular organization.

As will be explained below, we achieve the handling of dynamic constraints on the technical side by introducing snapshots together with particular temporal relationships (predecessor and successor relationships). Thereby, we enable the expression of dynamic constraints which cannot be handled in other approaches based on UML and OCL.

With our RBAC metamodel, we present a domain-specific language (DSL) adapted for RBAC, syntactically based on UML object diagrams. It assists administrators to configure complex policies while hiding the details like the underlying OCL constraints. The administrators design their policies by designing object diagrams, i.e., they create objects and links and define attribute values. At this 'policy level' the administrators are allowed to enforce complex formal constraints without the need to define or adjust the respective textual formulas. Also examinations at the so-called 'user access level' can be carried out in relation to the defined policies without knowing the internal matters of the RBAC metamodel. For validation purposes user activities can be simulated at the user access level in context of a given policy.

In the hidden complexity we see a great advantage compared to formal approaches based on model checkers and temporal (or relational) logic or other methods which require the handling of complex textual structures (like XACML [8]) during the policy specification and validation process. With our RBAC DSL the administrators can focus on realizing their security demands which are visually presented through UML. In an organizational context, a small number of UML and OCL experts can maintain and extend the RBAC DSL providing the administrators with the necessary functionality.

The DSL supports the constructs needed for a formal analysis of the created policy. These constructs including dynamic authorization constraints can be directly handled by any tool allowing for validation of UML models because all temporal requirements are encoded within an ordinary UML class diagram and OCL constraints. More generally, our approach for handling dynamics can be applied to other access control models and even to usage control models which comprise temporal aspects (reactive systems) [9], [10], [11], [12]. The relevance of such stateful security models has often been pointed out in literature [2], [3], [4], [9], [13], [14].

The (further) development of the RBAC UML metamodel, the DSL, is generally a time-consuming task, especially the development of its instances—the organizational RBAC policies—including a comprehensive analysis and evaluation. The development requires a powerful UML and OCL validation tool. An advantageous way for automatically analyzing complex UML models with OCL constraints is the use of our newly developed technique which we call OCL2Kodkod. It is implemented as plug-in for the UML-based Specification Environment (USE) [15] and makes use of the relational model finder Kodkod [16] which in turn makes use of solvers for boolean satisfiability (SAT). In the following sections we will consider and analyze our RBAC description from different perspectives, i.e., at different levels, in order to give a deeper insight into the RBAC metamodel and the given possibilities. OCL2Kodkod allows us to realize this proceeding, as it efficiently executes our validation demands.[1]

In summary, we combine the following aspects with the development and analysis of RBAC policies.

- Support for dynamics through snapshot modeling, enabling the tracing of user activities over time and the expression of dynamic constraints.
- Separation of the RBAC metamodel from concrete policy definitions.
- Hiding the complexity of the authorization constraints from the policy developers (DSL approach).
- Flexibility to extend the RBAC DSL with respect to specific organizational demands.
- Supporting automated analysis of RBAC policies.

The rest of the paper is structured as follows. In Sect. II we introduce needed RBAC concepts. Section III introduces the different levels in the RBAC UML description. The RBAC metamodel itself is explained in Sect. III-A and the supplemental OCL constraints in Sect. III-B. Section IV addresses the different levels of analysis with respect to the RBAC UML description. In Sect. IV-A we briefly introduce the OCL2Kodkod method, before we discuss the analysis of the RBAC metamodel in Sect. IV-B and RBAC policies in Sect. IV-C. Section V presents related works. We conclude with Sect. VI.

II. Sketch of RBAC and Authorization Constraints

Since several years, RBAC is widely used in organizations to simplify access management. Roles are explicitly handled in RBAC security policies. Thereby, security management is simplified and the use of security principles like 'separation of duty' and 'least privilege' is enabled [1]. We now give an overview of (general) hierarchical RBAC according to the RBAC standard [17] which is the fundament of our following RBAC UML approach.

RBAC relies on the following sets: U, R, P, S (users, roles, permissions, and sessions, respectively), $UA \subseteq U \times R$ (user assignment to roles), $PA \subseteq R \times P$ (permission assignment to roles), and $RH \subseteq R \times R$ (partial order called role hierarchy or role dominance relation written as $\leq$). *Users* may activate a subset of the roles they are assigned to in a *session*. P is the set of ordered pairs of *actions* and *resources*[2]. Resources represent all elements accessible in an information technology (IT) system, e.g., files and database tables. Actions, e.g., 'read', 'write' and 'append', are applied to resources. The relation PA assigns a subset of P to each role. Therefore, PA determines for each role the action(s) it may execute and the resource(s) to which the action in question is applicable for the given role. Thus, any user having assumed this role can apply an action to a resource if the corresponding ordered pair is an element of the subset assigned to the role by PA. Role

[1]The application of OCL2Kodkod is though not limited to the RBAC context. It can be applied to other UML and OCL models.

[2]Actions and resources are also called operations and objects in the RBAC context. For disambiguating RBAC and UML concepts, we continuously use the former notion.

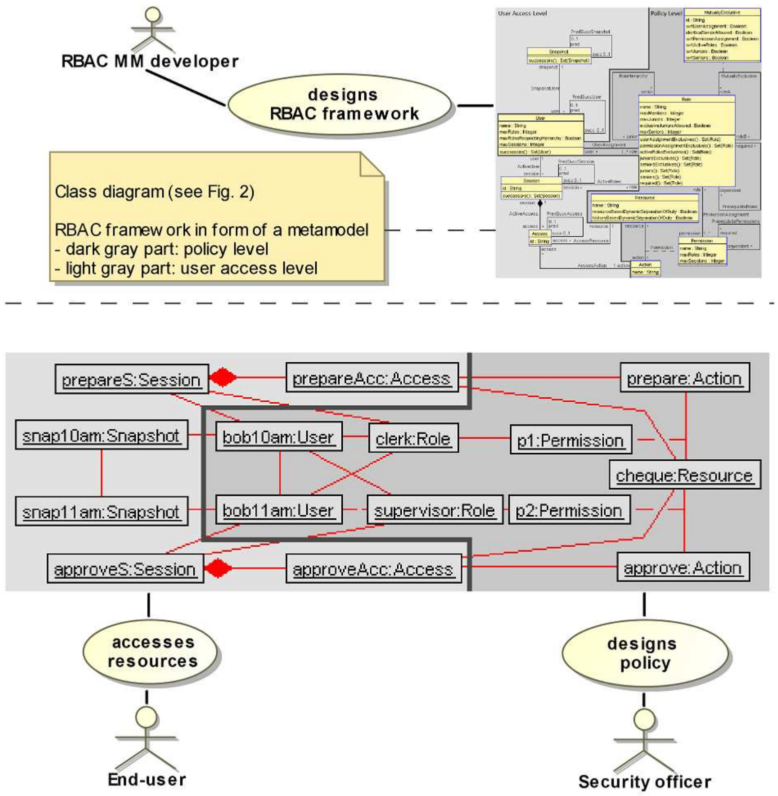

Fig. 1. Policy and user access level of the RBAC UML description

hierarchies can be formed by the RH relation. Senior roles inherit permissions from junior roles through the RH relation, e. g., the role 'chief physician' inherits all permissions from the 'physician' role.

An important advanced concept of RBAC are authorization constraints. Authorization constraints can be regarded as restrictions on the RBAC functions and relations. For example, a (static) SoD constraint may state that no user may be assigned to both the *cashier* and *cashier supervisor* role, i.e., the UA relation is restricted. It has been argued elsewhere [1] that authorization constraints are the principal motivation behind the introduction of RBAC. They allow a policy designer to express higher-level organizational rules as indicated above. In the literature, several kinds of authorization constraints have been identified. In this paper we exemplarily consider static and dynamic SoD [18], [3] and cardinality constraints [1]. Temporal considerations need extra preparation which we introduce later.

III. RBAC UML Description

Three central requirements form the basis of the developed RBAC metamodel. The model must provide for (1) the design of organizational (security) policies with respect to core RBAC concepts including authorization constraints, (2) a comprehensive validation of the specified policies including time-independent (static) and time-dependent (dynamic) aspects, and (3) expandability.

These requirements result in a UML class diagram with two parts describing a *policy level* for the policy design and a *user access level* for the policy analysis. Figure 1 visualizes the basic idea. An object diagram shows an example instance of the RBAC class diagram. The dark grey part represents a rudimentary policy specified by an administrator (security officer) through the creation of role, permission, action and resource objects and insertion of links between the objects. In this example no authorization constraints are involved. The light grey part simulates an IT system with one user ('bob') and his activities. With the help of the object diagram we are able to overview the main principles of RBAC and our UML model which is examined in detail later. The example policy manages the access to just one resource, a (bank) cheque.[3] Users in the role of a 'clerk' are entitled to prepare cheques. Users in the role of a 'supervisor' are allowed to approve them. As mentioned before, policy designers (administrators) normally aim to prevent situations in which the same user prepares and approves a critical resource like a cheque (SoD requirement).

The user access level—which is either manually instantiated by an administrator or automatically by a UML validation tool (USE) during the analysis process—is exclusively designed for the analysis of policies including authorization constraints like

[3]This is a simplified view to an RBAC permission management. RBAC policies usually abstract from individual resources.

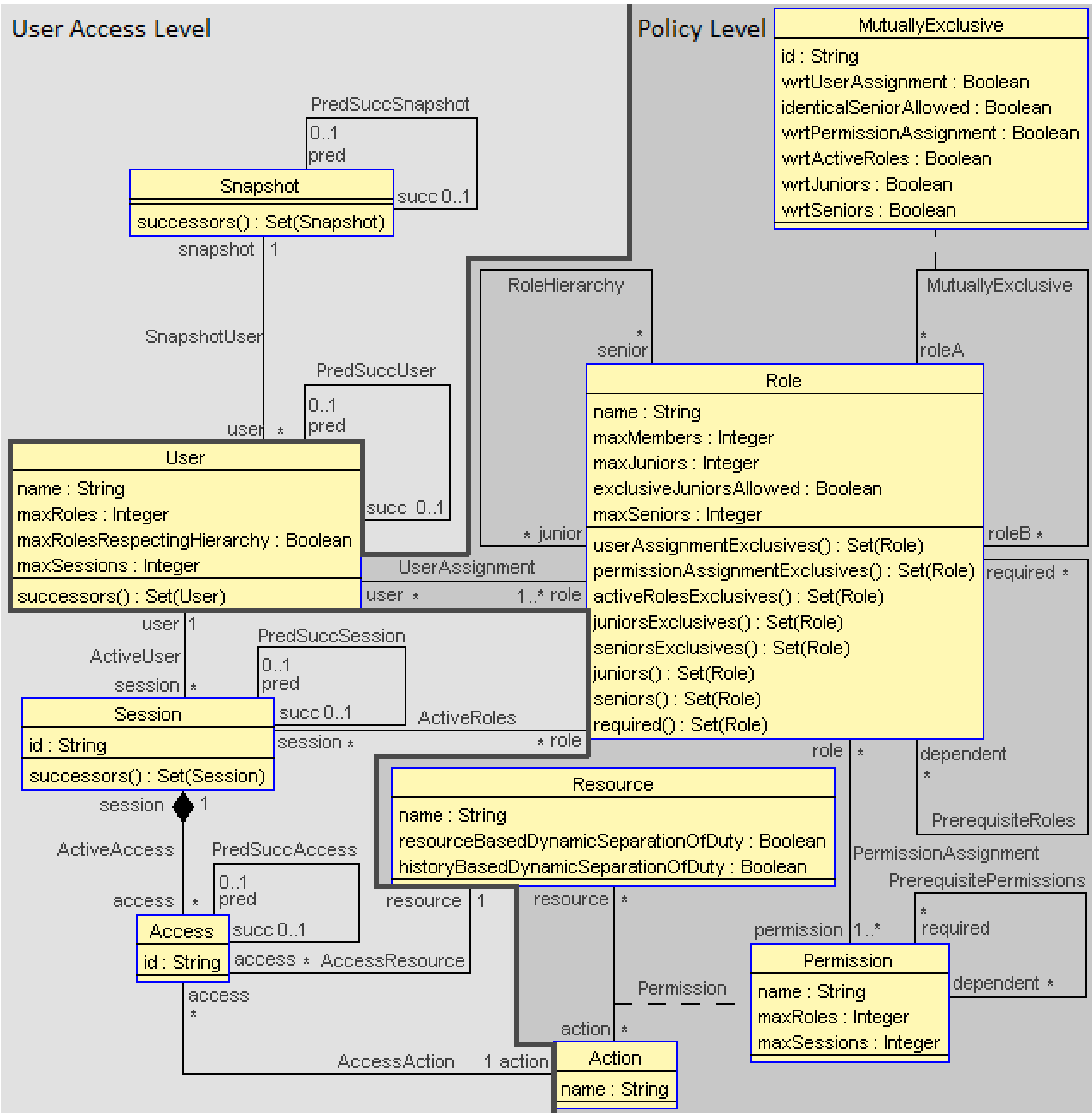

Fig. 2. The RBAC metamodel

the aforementioned SoD requirement. It simulates concrete user activities in the context of a policy, i.e., the actor 'End-user' in Fig. 1 represents real users defined by an administrator, but the users' activities are simulations of real events. In the present case the following situation is at hand. The user 'bob' prepares a cheque at 10 am and approves this cheque in a different session at 11 am, thus, violating the SoD requirement. Speaking more precisely, 'bob' accesses the real resource 'cheque' via the action 'prepare' and later in the context of another access via the action 'approve'. We call a point in time a *snapshot* and a sequence of snapshots a *scenario*.

The user activity can be checked with respect to the policy. It is either valid, i. e., the whole object diagram fulfills all underlying UML and OCL constraints specified with the RBAC metamodel, or invalid, i. e., the object diagram violates at least one UML or OCL constraint. The UML and OCL constraints are controlled by the policy part as the policy determines the set of active authorization constraints. For example, if the administrator activates the respective SoD authorization constraint (a boolean UML attribute belonging to 'Resource' objects which is currently hidden in the diagram) for the 'cheque', the OCL invariant enforcing the SoD requirement will come into effect. Thus, the present scenario will not be valid in the context of the restricting policy.

The distinction between the actors, i.e., the RBAC metamodel developers (the authors of this paper), security officers (administrators) and end-users, is helpful later when we address the various possibilities to analyze the RBAC description.

A. RBAC Metamodel

The object diagram shown in Fig. 1 is based on the RBAC metamodel shown in Fig. 2. Classes and associations belong analogously to the policy level or the user access level.

1) Policy Level: The dark grey policy part features the basic RBAC concepts. Users are assigned to at least one role. Roles entail a particular set of permissions which are needed for applying actions to resources. The role hierarchy and RBAC authorization constraints form the realized advanced concepts. Roles may have junior roles implying the inheritance of permissions. The authorization constraints are based on the fundamental paper of Sandhu [1] supplemented by dynamic constraints discussed in [7]. In our approach the constraints are realized as UML attributes and associations.

While integrating the authorization constraints into the RBAC metamodel, we adhered to the principle of strictly separating the RBAC metamodel from concrete policies. That is, concrete policies should be exclusively defined in object diagrams so that their specification does not require adjustments at metamodel level. Generally speaking, our approach allows the policy administrators to freely configure the needed authorization constraints by setting attribute values and inserting links between objects. While the attribute and association names are chosen to suggest the meaning of the corresponding constraint, we provide a short description for each realized authorization constraint within Tab. I. The OCL invariants implementing the authorization constraints are considered in Sect. III-B.

2) User Access Level: As explained before, the user access level displayed in the light grey part of Fig. 2 is an essential means for policy analysis. On the one hand, the class 'User' and related authorization constraints belong to the policy level because administrators create users and configure their access rights through the assignment to roles and the determination of the respective attribute values. On the other hand, a user represents a central element at the user access level because we model the users' activities via sessions and resource accesses at this level. In other words, a user object is part of a concrete policy, but the activated sessions and accesses related to the user object simulate an IT system which underlies the designed policy. This way, during the analysis process, we can for example identify user activities which are forbidden by the given policy specification, but are valid in the eyes of the administrators, or identify constellations which are allowed wrt. the policy but should actually be forbidden.

The policy level of the RBAC UML description follows the principles of an *application model*, whereas the user access level follows the principles of a *snapshot model* [19], [6]. That is, one object diagram for Fig. 2 describes exactly one policy, but several situations on the user access level, i.e., points in time in a IT system. The class 'Snapshot' and the associations with 'PredSucc' prefix enable the corresponding dynamics. A scenario consists of one chain of successive snapshots. Analogously, users, sessions and accesses can have successors. These predecessor/successor relationships allow for identifying the individual users, sessions and accesses over time (snapshots). For example, the user 'bob' is represented by one object per snapshot so that we can follow 'Bob's activities within the whole scenario. This aspect is not explicitly treated in [6].

This snapshot modeling of the user access level with pred/succ associations allows us the analysis of time-dependent (dynamic) constraints.

B. Supplemental OCL Constraints

The RBAC class diagram is supplemented by OCL invariants which serve three purposes. They (1) represent the fundament of the authorization constraints by formalizing the respective requirements, (2) check for reasonable policy designs, and (3) regulate the snapshot concepts.

The OCL invariants make use of OCL query operations displayed in the operation parts of the classes (see Fig. 2). The query operations represent auxiliary functions simplifying the invariant bodies or calculating transitive closures. For example, the operation 'successors' (Snapshot) returns all direct and indirect successors of the snapshot under consideration, or the operation 'required' (Role) calculates all directly and indirectly required roles in the context of the calling role object.

1) Formalizing Authorization Constraints: Each authorization constraint produces an OCL invariant which checks whether a user access scenario complies with the authorization constraint. The administrators determine for which objects the authorization constraint should be activated, i. e., the invariant should be applied. For example, consider the invariant 'MaximumNumberOfMembers' stated below. It corresponds to the authorization constraint which is configured with the attribute 'maxMembers' of class 'Role'. After determining a value for 'maxMembers' in the context of a role object in the policy, the related invariant is activated which checks the requirement for the role object.

```
context r:Role inv MaximumNumberOfMembers:
  r.maxMembers.isDefined implies
    r.user->size() <= r.maxMembers
```

This invariant expresses a static, time-independent property because it must hold in each point in time. In contrast, the invariant 'NoExclusiveRolesActive' related to the (switch) attribute 'wrtActiveRoles' of class 'MutuallyExclusive' has to respect the snapshot framework. It ensures that no pair of roles exists which is characterized as mutually exclusive with respect to the activation in a single session (i. e., the attribute 'wrtActiveRoles' is set to 'true'; this attribute is used for the definition of the query operation 'activeRolesExclusives' used in the invariant defined below).

```
context s:Session inv
NoExclusiveRolesActive:
  let activeRoles =
    s.successors().role->union(s.role) in
  activeRoles->excludesAll(
    activeRoles.activeRolesExclusives())
```

As sessions are active in an arbitrary time frame, they often persist several snapshots until the respective user terminates them. Hence, the invariant must regard the whole time frame wrt. a session, i.e., the sequence of successive session objects (s.successors()), representing the single considered session

TABLE I
REALIZED AUTHORIZATION CONSTRAINTS

Constraint	Description	Reference
User:: `maxRoles`	maximum number of roles the user is assigned to (respecting or ignoring the role hierarchy, depending on the boolean value of attribute 'maxRolesRespectingHierarchy')	Sandhu [1], page 11, lines 29–30
`maxSessions`	maximum number of simultaneously active sessions wrt. a user	Sandhu [1], page 12, lines 15–16
Role:: `maxMembers`	maximum number of assigned users	Sandhu [1], page 11, lines 27–28
`maxJuniors`	maximum number of inheriting junior roles (mutually exclusive juniors allowed or prohibited, depending on the boolean value of attribute 'exclusiveJuniorsAllowed')	Sandhu [1], page 12, lines 30–31
`maxSeniors`	maximum number of senior roles	Sandhu [1], page 12, lines 30–31
`PrerequisiteRoles` (Assoc.)	dependent role postulates required role wrt. user assignment	Sandhu [1], page 11, lines 36–38
MutuallyExclusive::		
`wrtUserAssignment`	a user must not be assigned to both of the connected roles (identical seniors can be explicitly allowed by setting the boolean attribute 'identicalSeniorAllowed' to true)	Sandhu [1], page 11, lines 6–7
`wrtPermissionAssignment`	a permission must not be assigned to both roles	Sandhu [1], page 11, lines 10–12
`wrtActiveRoles`	the connected roles must no be both activated in a session (possibly involving several snapshots)	Sandhu [1], page 12, lines 14–15
`wrtJuniors`	the connected roles must not have the same junior roles	Sandhu [1], page 12, lines 31–32
`wrtSeniors`	the connected roles must not have the same senior roles	Sandhu [1], page 12, lines 31–32
Permission:: `maxRoles`	maximum number of roles the permission is assigned to	Sandhu [1], page 11, lines 30–32
`maxSessions`	maximum number of sessions simultaneously activating the permission (i. e., within the same snapshot)	Sandhu [1], page 12, lines 16–17
`PrerequisitePermissions` (Assoc.)	assignment of the dependent permission postulates the assignment of the required permission	Sandhu [1], page 12, lines 1–3
Resource::		
`resourceBasedDynamic-SeparationOfDuty`	a user may not apply more than one action to the resource	Simon & Zurko [3], page 4, line 16–20
`historyBasedDynamic-SeparationOfDuty`	a user may not apply all available actions to the resource	Simon & Zurko [3], page 4, line 28–39

over time. Further dynamic authorization constraints are discussed within Sect. IV.

2) Checking for Reasonable Policies: The model comprises further invariants assisting the administrators (at a syntactical level) to design correct policies. Thus, structurally inconsistent policies, e. g., showing self excluding roles or roles which simultaneously require and exclude themselves, can be avoided in the first place. The aim is to allow the administrators to focus on semantical aspects, like assigning the end-users to proper roles so that they achieve a policy which matches their intended security properties.

3) Constraining User Access Scenarios: Finally, a set of OCL invariants is created to maintain valid sequences of snapshots. For example, only one scenario is allowed within an object diagram and the set of snapshots must be properly ordered.[4]

IV. Analyzing the RBAC Description

If we consider the complexity of real RBAC policies and the extensive possibilities of designing a policy by means of the RBAC metamodel, on the one hand, and if we respect the resulting possibilities of missing the security holes, on the other hand, we see that computer-aided analysis is essential with respect to the policy level.

As an adequate RBAC metamodel is the precondition for designing accurate policies, the model itself must be sound. Regarding the number of classes, associations and attributes as well as the number of OCL constraints, the RBAC UML model has reached a size which makes pure manual validation impossible. Thus, the UML and OCL experts who maintain the RBAC metamodel (the DSL) within an organization (as well as the authors of this paper) also need tool support. Table II shows the different approaches to analyzing the RBAC artifacts including the RBAC metamodel, RBAC policies and the user access. In the following, the user access level can be disregarded because user activities are restricted by a policy. Consequently, a complete and correct policy is sufficient with respect to the enabling of only valid user activities.

A. The Method OCL2Kodkod

Our RBAC description provides diverse interfaces for analysis so that any UML and OCL tool with analysis functionality can help to ensure a sound RBAC metamodel and well-designed policies. We follow the approach of the UML-based Specification Environment (USE) [15]. In order to ensure properties of the metamodel or the policies, we search system state spaces, i. e., sets of objects diagrams. The existence of an object diagram fulfilling specified conditions gives information about the model or the policy characteristics.

The success of this approach strongly depends on the performance of the underlying search engine. In [7], we employ the ASSL generator [15] integrated into USE to analyze RBAC policies in order to detect missing and conflicting static authorization constraints. The enumerative generator has to consider all possible object diagrams in the worst case, i.e., if there is no state having the required properties. Hence, it

[4]All sources related to the RBAC metamodel can be found in [20].

TABLE II
DIFFERENT PERSPECTIVES OF ANALYZING RBAC

RBAC level	Focus	Analyzed by	Considered subject
RBAC metamodel	class diagram and OCL constraints	RBAC DSL developers	all instantiable policies all possible RBAC **scenarios**
RBAC policy	**static** policy aspects	policy administrators	one specific (partial) policy all possible RBAC **snapshots**
	dynamic policy aspects	policy administrators	one specific (partial) policy all possible RBAC **scenarios**
User access	resource access	RBAC policy	one specific RBAC **scenario**

is not able to handle models in size of the present RBAC metamodel with acceptable answer times. The developed method OCL2Kodkod resolves this problem. It is based on the relational model finder Kodkod representing the successor of the Alloy Analyzer [21]. Both tools provide a relational logic for specifying and analyzing models. Internally, they translate the model and properties to be checked into a SAT problem which can be handled by any SAT solver. Kodkod is designed as a Java API making the integration into other tools easy.

Our method OCL2Kodkod includes a translation from UML and OCL concepts into relational logic. The current version comprises all important UML class diagram and OCL features. As the RBAC metamodel is completely supported, it can be taken as an example for the successful use of OCL2Kodkod (see [20] for details). All examinations presented in the following sections complete within seconds using an ordinary laptop.

B. Analyzing the RBAC Metamodel

A comprehensive analysis of the RBAC metamodel during development helped us to discover several unwanted properties of which we present two as an example in this section. Also future extensions of the model with respect to further RBAC features will benefit from further analysis of the model properties. Our examinations presented here are based on the core concepts *independence* and *reasoning* discussed in [22].

1) Independence: The independence of constraints describes the fact that each defined constraint adds essential information to the model, i. e., it further restricts the space of valid object diagrams. This property can be checked by searching an object diagram fulfilling all constraints but the constraint under consideration. If such a diagram exists, the respective constraint is independent from the others because it does not follow from them. This check has to be executed 30 times, as the RBAC metamodel currently comprises 30 OCL constraints. We automated the sequential checks with OCL2Kodkod so that no further manual interaction is needed. Each check results in an object diagram or yields no solution. The latter case indicates dependencies between the constraints which have to be further examined, e. g., by temporarily disabling not involved constraints.

Within the RBAC metamodel all constraints are independent. However, the consideration of the generated object diagrams is a part of a valuable analysis because they can reveal erroneous constraints. For example, OCL2Kodkod returned the object diagram shown in Fig. 3 in an early development

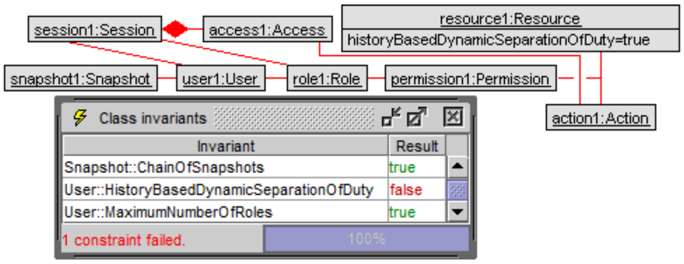

Fig. 3. Object diagram revealing an erroneous constraint definition

phase of the RBAC metamodel. As expected, the diagram proves the independence of the invariant 'HistoryBasedDynamicSeparationOfDuty' because all invariants but this are fulfilled.[5] However, the object diagram shows a situation which should normally not violate the respective invariant. The SoD constraint states that a user must not apply all available operations to a resource. But in this case, there was just one action which should be available for application. Thus, the object diagram pointed out that we forgot to handle the particular case of exactly one available action. After correcting the invariant we got an adequate, yet more complex result. The results for each invariant are presented in [20].

2) Reasoning: Reasoning stands for the universal examination of model properties. Properties under consideration are often complex, but in many cases simple properties already lead to the desired information. For example, in order to check a specific RBAC metamodel invariant we can configure OCL2Kodkod to search a valid object diagram in which the authorization constraint corresponding to the invariant is activated. This way, we discovered a further erroneous invariant during development. We searched for an object diagram showing a simple policy with one resource and one action. The permission corresponding to the action resource pair had to define a maximum number of active sessions (maxSessions = 1). Additionally, the diagram had to simulate a user access scenario with at least five snapshots and at least five sessions with user accesses. Although there should be many valid object diagrams, we got no solution with respect to this search space. After deactivating the invariants which had no effect on the result we found out that two invariants ('ActionsPermitted' and 'MaximumNumberOfSessions') were responsible for this unwanted behavior. The former invariant

[5]The shown class invariant view of the USE tool displays an extract of all invariants.

ensures that only permitted accesses to resources exist. The latter realizes the authorization constraint which controls the maximum number of sessions:

```
context p:Permission inv
MaximumNumberOfSessions:
  p.maxSessions.isDefined implies
    p.role.session->asSet()->size()
    <= p.maxSessions
```

Within a session, permissions are indirectly activated by activating a role the permission is assigned to. Thus, the expression `p.role.session->asSet()->size()` returns the number of all (distinct) sessions which activated the current permission disregarding the time. That is, also closed sessions, which do not activate a permission any more, are involved. Hence, we adjusted the invariant to respect the dynamics resulting from the scenarios (see the constraint below). The maximum number of sessions is now calculated in the context of the individual snapshots so that only simultaneously active sessions are counted.

```
context p:Permission inv
MaximumNumberOfSessions:
  p.maxSessions.isDefined implies
    Snapshot.allInstances()->forAll(snap |
      p.role.session->asSet()->select(s |
        s.user.snapshot = snap)->size()
      <= p.maxSessions)
```

C. Analyzing RBAC policies

Complex security policies usually become intransparent with respect to their implicit properties, i. e., the combination of the explicitly stated authorization constraints often yields new properties which have to be analyzed. Consequently, changes to a policy may have various effects. Even simple policies like the ones presented in this section can reveal unanticipated characteristics. In the context of our RBAC metamodel and OCL2Kodkod these characteristics can be uncovered by searching specific object diagrams. In contrast to the analysis at the metamodel level, the analysis of policies is normally based on a given object diagram representing the policy under consideration or a partial policy which may be automatically adapted during the search. That is, administrators can determine which parts of the designed policy should be fixed (e. g., permission ‘p1’ must be assigned to role ‘clerk’ and the number of roles must not change) or are variable (e. g., the user assignment to roles can arbitrarily be changed during the search). In many cases, at least some parts of a policy remain variable.

The analysis with OCL2Kodkod needs two artifacts, an object diagram – the (partial) policy – and a property to be checked. The property can be formulated in form of a usually non-complex OCL expression and by explicitly stating the bounds with respect to the number of objects and links for each class and association as well as the definition of attribute values. Let us take the object diagram shown in Fig. 4 which presents the first artifact, a partial policy (grey

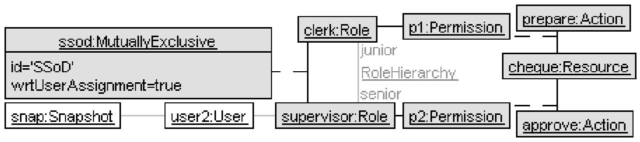

Fig. 4. Partial policy and partial search results (white objects, grey links)

objects and black links) with some fixed elements (e. g., the existing objects must not be deleted, users do not change their roles, and the attribute value of ‘wrtUserAssignment’ must remain ‘true’, i.e., a user may not have both roles ‘clerk’ and ‘supervisor’ at the same time). The white objects and grey links are not part of the policy. They are addressed later.[6] The second artifact represents the following property to be checked (informally): ‘Does the policy allow a user to apply both actions (‘prepare’ and ‘approve’) to the resource in the context of a snapshot, although a user cannot have both roles?’ Modeling this property with OCL we require (among other requirements) the following statement to be fulfilled.

```
User.allInstances()->exists(u |
  u.session.access.action->asSet()->size()
  = 2)
```

These kind of statements normally have specific patterns which are often reused in case of other properties. Thus, the administrators do not need a deep insight into the OCL semantics. Moreover, the patterns could be respected and implemented in the used UML tool (e.g., USE) in order to allow property configurations through a graphical user interface.

Giving both artifacts to OCL2Kodkod, it returns a completed object diagram fulfilling all constraints. It is partly shown with the white objects and grey associations in Fig. 4. We hide the overhead like the session in which the user accesses the resource via both actions. We see that the static SoD property is circumvented, if the role ‘clerk’ becomes the junior role of ‘supervisor’ because a supervisor will in turn inherit all permissions from a clerk.

The former property can be checked in the context of one point in time (snapshot) because it does not depend on dynamic activities. The following example considers a whole scenario. The starting point is again the policy shown in Fig. 4 with small changes. Instead of setting ‘wrtUserAssignment’ to true we set ‘wrtActiveRoles’ to true, thus activating the related OCL invariant shown in Sect. III-B1, i.e., both roles must not be activated in the same session. Additionally, we forbid the creation of role hierarchy links. Now we would like to check the same property as before. Does a user have the rights to execute both actions? OCL2Kodkod returns the object diagram shown in Fig. 5. It is similar to the exemplary diagram in Fig. 1. We see that a user can access the resource via both actions within two different successive sessions (there is no pred/succ link between the objects ‘session1’

[6] Please note that we manually adapted the displayed object diagram to combine the elements existing before and after the search.

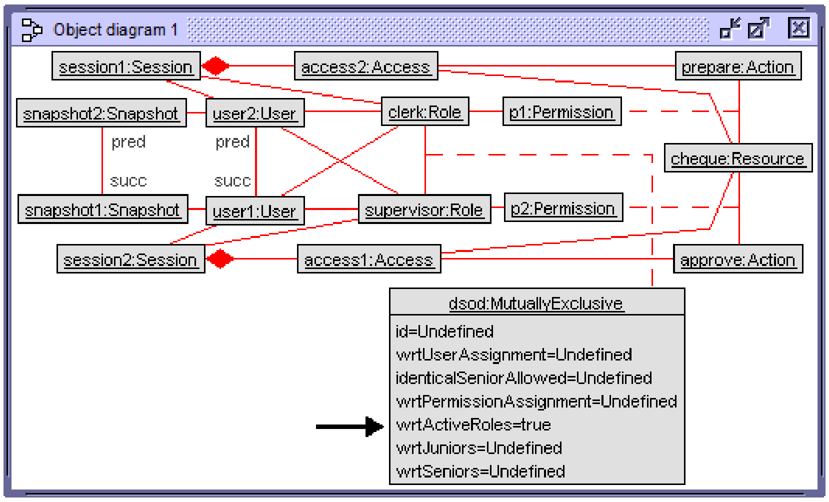

Fig. 5. Automatically generated scenario fulfilling the stated properties

and 'session2').[7] The authorization constraint prohibiting the simultaneous activation of both roles does not apply.

This result shows the policy developer that both attributes 'wrtUserAssignment' and 'wrtActiveRoles' must be set to true in order to prevent a user to execute both actions ('prepare' and 'approve') to the considered resource, i.e., both corresponding authorization constraints must be activated in context of the roles 'clerk' and 'supervisor'. Additionally, we discovered that both roles must not be related trough a role hierarchy. OCL2Kodkod approves this assumption, as it does not find a solution which on the one hand allows a user to execute both actions to the resource and on the other hand fulfills all authorization constraints. For this search task we instructed OCL2Kodkod to check all object diagrams which have up to 30 user, snapshot, session and access objects. Consequently, there is no scenario consisting of 1 to 30 snapshots in which the unwanted activity can be performed. These results base on the aforementioned precondition that users do not change their roles within a scenario. If we allowed changes, e.g., a role switch from 'clerk' to 'supervisor' in two successive snapshots, a user would still have the rights to execute both actions.

Beside the automatically generated object diagrams, it is also often very helpful to manually specify scenarios of user activities. They can, for example, be used as positive (valid object diagrams) and negative (invalid object diagrams) test cases during the development of policies. When a reasonable set of test cases is available, it can be periodically checked during the development process because a failed test can indicate the existence of a new unwanted property within the policy, possibly resulting from the interplay of several authorization constraints. However, if a policy undergoes great structural changes the test cases must be adapted accordingly.

[7]The numbers within the generated object names do not have further meaning, i.e., they do not indicate an order within a scenario. The order is determined by the role names 'pred' and 'succ'.

V. Related Work

There is a plethora of works integrating security policies into system models based on UML such as [7], [23], [24], [25], [26], [27]. Some of the approaches do not particularly address RBAC like UMLsec [23]. Basin et al. [27] present the modeling language SecureUML for integrating the specification of access control into application models and for automatically generating access control infrastructures for applications. They also deal with authorization constraints, but do not support SoD constraints. In [7] we explicitly model RBAC SoD constraints with UML and OCL. There, we have no means for handling dynamic aspects and we do not strictly separate the presented RBAC metamodel from concrete policy definitions. Ray et al. [24] solve the latter problem by generically designing the authorization constraints. We follow their approach with respect to the RBAC description presented in this paper and extend it in terms of dynamic aspects.

Several works on the validation of RBAC policies based on UML and OCL have been presented [28], [6], [7], [29]. Based upon SecureUML, Basin et al. propose an approach to analyzing RBAC policies by stating and evaluating queries like 'Which permissions can a user perform with a given role?' or 'Are there two roles with the same set of permissions?' [28]. Although not explicitly addressed in this paper, our approach allows the same kind of queries through the query facility of the USE tool [15] into which the method OCL2Kodkod is integrated.

In [6], a scenario-based approach to analyzing UML models is presented which is exemplified by an elementary RBAC UML model. In this context, a policy is considered as a dynamic artifact which evolves through administrator activities. Hence, it can be examined whether a sequence of administrative RBAC operations such as assigning users to roles can violate static SoD constraints. In contrast, we realize dynamics at the end-user level, enabling dynamic SoD. Administrative actions are implicitly involved in our approach when analyzing partial policies. In addition, our RBAC metamodel consists of both a static and a dynamic part.

Our OCL2Kodkod approach developed and applied for comprehensive validation is related to UML2Alloy [21], a method for translating UML and OCL into Alloy specifications because the 'Alloy Analyzer' is the predecessor of Kodkod. However, UML2Alloy does not yet support some frequently used UML and OCL features like n-ary associations, association classes or standard operations on integer values which are provided by OCL2Kodkod.

VI. Conclusion

The paper presented an RBAC metamodel as a basis for an RBAC DSL allowing security officers to design complex policies and to analyze explicit and implicit properties without handling the often very complex underlying textual constraints. Properties can be time-dependent (dynamic) and time-independent (static) corresponding to the nature of the authorization constraints which may relate past, present and future activities at the end-user level. We discussed the need for analysis and validation. Even small changes to a model can imply new implicit properties the developer may not think of, regardless of whether one considers the design of a policy or the development of the RBAC metamodel itself.

Our current RBAC metamodel features basic concepts and authorization constraints. It is bulit as a groundwork for versatile extensions like advanced role delegation and revocation concepts. We plan to extend the model in different ways in order to achieve a mature RBAC framework including all concepts for practical application. Our proposal must be improved by performing larger case studies. Another direction of our work concerns polishing the user interface. Currently, many tasks require UML and OCL knowhow. An improved user interface may hide such irrelevant details and let the user concentrate on security concerns.

Acknowledgment

We would like to thank the referees for their helpful comments.

References

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

[2] Nash, M.J., Poland, K.R.: Some conundrums concerning separation of duty. In: Proc. IEEE Symposium on Research in Security and Privacy. (1990) 201–207

[3] Simon, R., Zurko, M.: Separation of duty in role-based environments. In: 10th IEEE Computer Security Foundations Workshop (CSFW '97). (1997) 183–194

[4] Bertino, E., Ferrari, E., Atluri, V.: The specification and enforcement of authorization constraints in workflow management systems. ACM Trans. Inf. Syst. Secur. **2**(1) (1999) 65–104

[5] Schaad, A., Lotz, V., Sohr, K.: A model-checking approach to analysing organisational controls in a loan origination process. In: Proc. of the 11th ACM Symposium on Access Control Models and Technologies, New York, ACM Press (2006)

[6] Yu, L., France, R.B., Ray, I.: Scenario-Based Static Analysis of UML Class Models. In: Model Driven Engineering Languages and Systems, 11th International Conference, MoDELS 2008. Volume 5301 of LNCS., Springer, Berlin (2008) 234–248

[7] Sohr, K., Drouineaud, M., Ahn, G.J., Gogolla, M.: Analyzing and Managing Role-Based Access Control Policies. IEEE Trans. Knowl. Data Eng **20**(7) (2008) 924–939

[8] Abi Haidar, D., Cuppens-Boulahia, N., Cuppens, F., Debar, H.: An extended RBAC profile of XACML. In: Proceedings of the 3rd ACM workshop on Secure web services. SWS '06, New York, NY, USA, ACM (2006) 13–22

[9] Barth, A., Datta, A., Mitchell, J.C., Nissenbaum, H.: Privacy and Contextual Integrity: Framework and Applications. In: IEEE Symposium on Security and Privacy, IEEE Computer Society (2006) 184–198

[10] Dougherty, D.J., Fisler, K., Krishnamurthi, S.: Specifying and Reasoning About Dynamic Access-Control Policies. In Furbach, U., Shankar, N., eds.: IJCAR. Volume 4130 of Lecture Notes in Computer Science., Springer (2006) 632–646

[11] Hilty, M., Pretschner, A., Basin, D.A., Schaefer, C., Walter, T.: A Policy Language for Distributed Usage Control. In Biskup, J., Lopez, J., eds.: ESORICS. Volume 4734 of Lecture Notes in Computer Science., Springer (2007) 531–546

[12] Zhang, X., Parisi-Presicce, F., Sandhu, R., Park, J.: Formal model and policy specification of usage control. ACM Transactions on Information and System Security **8**(4) (November 2005) 351–387

[13] Clark, D.C., Wilson, D.R.: A comparison of commercial and military security policies. In: Proc. IEEE Symp.on Security and Privacy, Washington DC. (1987)

[14] Sandhu, R.: Transaction control expressions for separation of duties. In: Proc. of the Fourth Computer Security Applications Conference. (1988) 282–286

[15] Gogolla, M., Büttner, F., Richters, M.: USE: A UML-Based Specification Environment for Validating UML and OCL. Science of Computer Programming **69** (2007) 27–34

[16] Torlak, E., Jackson, D.: Kodkod: A Relational Model Finder. In: Tools and Algorithms for the Construction and Analysis of Systems - 13th International Conference, TACAS 2007. Volume 4424 of LNCS., Springer, Berlin (2007) 632–647

[17] American National Standards Institute Inc.: Role Based Access Control (2004) ANSI-INCITS 359-2004.

[18] Gligor, V.D., Gavrila, S.I., Ferraiolo, D.: On the formal definition of separation-of-duty policies and their composition. In: 1998 IEEE Symposium on Security and Privacy (SSP '98), IEEE (1998) 172–185

[19] Kuhlmann, M., Gogolla, M.: Modeling and Validating Mondex Scenarios Described in UML and OCL with USE. Formal Aspects of Computing **20**(1) (2008) 79–100

[20] Kuhlmann, M., Sohr, K., Gogolla, M.: RBAC Metamodel: Sources and Validation Results (2010) `http://www.db.informatik.uni-bremen.de/publications/Kuhlmann_2010_RBAC_sources.pdf`.

[21] Anastasakis, K., Bordbar, B., Georg, G., Ray, I.: UML2Alloy: A Challenging Model Transformation. In: Model Driven Engineering Languages and Systems, 10th International Conference, MoDELS 2007. Volume 4735 of LNCS., Springer, Berlin (2007) 436–450

[22] Gogolla, M., Kuhlmann, M., Hamann, L.: Consistency, Independence and Consequences in UML and OCL Models. In: Proc. 3rd Int. Conf. Test and Proof (TAP'2009), Springer, Berlin, LNCS 5668 (2009) 90–104

[23] Jürjens, J.: UMLsec: Extending UML for secure systems development. Lecture Notes in Computer Science **2460** (2002) 412–425

[24] Ray, I., Li, N., France, R.B., Kim, D.K.: Using UML to visualize role-based access control constraints. In: Proc. of the 9th ACM symposium on Access control models and technologies, ACM Press New York, USA (2004) 115–124

[25] Ahn, G.J., Shin, M.E.: Role-Based Authorization Constraints Specification Using Object Constraint Language. In: Proc. of the 10th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, IEEE (2001) 157–162

[26] Fernández-Medina, E., Piattini, M.: Extending OCL for secure database development. In: Proc. of UML 2004 - The Unified Modeling Language: Modeling Languages and Applications. Volume 3273 of LNCS., Springer (2004) 380–394

[27] Basin, D.A., Doser, J., Lodderstedt, T.: Model driven security: From UML models to access control infrastructures. ACM Trans. Softw. Eng. Methodol **15**(1) (2006) 39–91

[28] Basin, D.A., Clavel, M., Doser, J., Egea, M.: Automated analysis of security-design models. Information & Software Technology **51**(5) (2009) 815–831

[29] Höhn, S., Jürjens, J.: Automated checking of SAP security permissions. In: 6th Working Conference on Integrity and Internal Control in Information Systems (IICIS), Lausanne, Switzerland, Kluwer (2003)