Towards Effective Verification of Multi-Model Access Control Properties

Bernhard J. Berger
University of Bremen
Bremen, Germany
bernhard.berger@uni-bremen.de

Christian Maeder
University of Bremen
Bremen, Germany
c.maeder@uni-bremen.de

Rodrigue Wete Nguempnang
University of Bremen
Bremen, Germany
wete@uni-bremen.de

Karsten Sohr
University of Bremen
Bremen, Germany
sohr@uni-bremen.de

Carlos Rubio-Medrano
Arizona State University
Tempe, AZ, USA
crubiome@asu.edu

ABSTRACT

Many *existing* software systems like logistics systems or enterprise applications employ data security in a more or less ad hoc fashion. Our approach focuses on access control such as permission-based discretionary access control (DAC), variants of role-based access control (RBAC) with delegation, and attribute-based access control (ABAC). Typically, software systems implement hybrid access control making an effective security analysis and assessment rather difficult.

We propose an analysis methodology to reconstruct access control using a novel *modular* access control model. Our modular approach allows us to flexibly model exactly those access properties that are relevant for a given system. As formalism we use the Object Constraint Language (OCL) with Ecore from the Eclipse Modeling Framework (EMF).

We demonstrate the suitability of our access control model for three software systems: a port community system (PCS), a clinical information system (CIS), and an identity management system (IdMS). For the PCS and CIS we model concrete roles and policies. For the IdMS we evaluate our analysis methodology in-depth by reconstructing access control policies from byte code using the Soot analysis framework as well as model transformation techniques (QVTo). The resulting model helped us to identify design deficiencies. Violated OCL invariants such as for mutually exclusive roles or cardinality constraints revealed non-trivial security vulnerabilities.

CCS CONCEPTS

• **Security and privacy → Access control**; • **Information systems →** *Enterprise applications*; • **Software and its engineering →** *Specification languages*; *Software reverse engineering*.

SACMAT '19, June 3–6, 2019, Toronto, ON, Canada

ACM ISBN 978-1-4503-6753-0/19/06…$15.00
https://doi.org/10.1145/3322431.3325105

KEYWORDS

Access Control Model; Permissions; RBAC; ABAC; Delegation; OCL; Reverse Engineering

ACM Reference Format:
Bernhard J. Berger, Christian Maeder, Rodrigue Wete Nguempnang, Karsten Sohr, and Carlos Rubio-Medrano. 2019. Towards Effective Verification of Multi-Model Access Control Properties. In *The 24th ACM Symposium on Access Control Models and Technologies (SACMAT '19), June 3–6, 2019, Toronto, ON, Canada.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3322431.3325105

1 INTRODUCTION

Established software systems have been initially developed without security in mind. Only when the software gets larger and more complex, security becomes a topic for the software vendor. It often happens that early security efforts are carried out exploratory, since many companies have no deep security knowledge. This leads to solutions that do not fit the approaches that have been researched. Nevertheless, many companies finally come to the point where they take the subject of software security more seriously. The attempt to use standard mechanisms for security is made difficult at this point by historically grown solutions, i.e. old code that needs to be maintained for the functionality but is based on outdated or no security concepts.

In particular, we observed that in practice different access control concepts are combined, for example, role-based access control (RBAC) [13, 29] is often mixed with permission-based discretionary access control (DAC) [11] using access control lists (ACLs). Both access control models are still widespread and contrast recommendations to use pure attribute-based access control (ABAC) [19].

In the concrete sense, existing modeling approaches for authorization do not support such combined models, and fall short of providing guarantees for accuracy, model-level as well as implementation-level verification and validation, thus potentially opening the door for the existence of serious security vulnerabilities.

To better understand such *implemented* access control policies, one needs a modeling approach that (i) captures those concepts usually occurring in access control implementations and (ii) integrates these access control concepts into a unified model. Research showed that even for simple programs the implemented and planned access control policies differ [33]. Ultimately, reconstructed access control

policies can be thoroughly assessed and potentially verified to make an application more secure.

We use Ecore from the Eclipse Modeling Framework (EMF) [35]—an alternative meta-modeling architecture with concepts similar to the Unified Modeling Language (UML) for class modeling—in conjunction with the Object Constraint Language (OCL) for the representation of our access control model.

According to Ferraiolo et al. [13] the term access control *model* denotes an intermediate abstraction level between a high-level access control *policy* and a low-level access control mechanism. We use the term *model* in the broader (EMF) sense. Our modular access control model allows us to express access control policies that are implemented in different domains, e.g. healthcare, logistics, large enterprises, or banking.

Software models can be (re-)created by hand, but the main idea is to obtain semi-automatically (UML or Ecore) models and/or object diagrams from real software artifacts like requirements specifications, documentation, source code and configuration files. Sometimes, specifying UML models already exist for a system, but it is unclear if such models still reflect a deployed implementation. Certainly, an existing UML model can serve as basis for our modeling of access control, but we want to reconstruct accurate models from source or byte code. For analyzing byte-code and extracting *relevant* access information we use the Soot analysis framework [38] with specifically written plugins. For adjusting extracted models to fit our access control formalism we use QVTo [37] for model-to-model (M2M) transformations. Basic hand-written parsers are also employed to extract access information from concrete configuration files as we elaborate in Section 5.3.

RBAC is an important part of our unified model because we still often need to capture role-based policies. In contrast to prior work [23, 34, 36] our access control model is more *modular* regarding various RBAC concepts including role-based *delegation*. Our model also allows us to disregard roles entirely, for example, to capture DAC or basic ABAC policies. Role-based delegation makes a rather static RBAC policy more flexible. For full flexibility we use *authorization contexts* [18, 25] that support context sensitivity and essentially correspond to basic ABAC.

We evaluate our access control model and modeling approach for three real-world software systems: a port community system (PCS), a clinical information system (CIS), and an identity management system (IdMS) of a large enterprise. The PCS, see Section 5.1, links different stakeholders of seaports, e.g. terminal operators, forwarders, and shipping companies. The clinical information system, see Section 5.2, is—due to critical and sensitive electronic patient's records—a typical example of access control. We create software models with intended access control policies from software artifacts like documentation and business process descriptions. For the IdMS, described in Section 5.3, we actually extracted the implemented access control policy from Java byte code and generated model instances from live data using reverse-engineering techniques as, for example, described by Koschke and Simon [22]. The reconstructed architecture served as a basis for security audits and helped revealing a series of non-trivial vulnerabilities.

Our contributions are threefold. We present:

1. a novel *modular* access control model covering a wide and heterogeneous range of practical access control concepts,
2. a methodology on how to reconstruct concrete access control of *existing* software systems, and
3. an evaluation showcasing the suitability of our modeling and reconstruction approach to discovering access control vulnerabilities.

The paper is organized as follows. We assume familiarity with classical RBAC as given by Sandhu et al. [29]. In Section 2, we present related work. In Section 3, we give a short overview of Ecore. Section 4 explains our approach in detail and elaborates on various forms of authorization constraints. In Section 5, we demonstrate the applicability of our access control model to different domains and evaluate our reconstruction methodology for an identity management system. After a brief discussion of possible disadvantages of our approach in Section 6 we conclude in Section 7.

2 RELATED WORK

The classical RBAC paper of Sandhu et al. 1996 [29] describes roles, role hierarchies and constraints like mutually exclusive roles, cardinality constraints, and prerequisite roles. Ahn and Sandhu investigate much more authorizations constraints and suggest a formal specification language [3, 4]. Shin and Ahn 2000 [30] suggest to use UML as RBAC representation. In 2001, they propose to use OCL to specify authorizations constraints [5]. Barka and Sandhu 2000 [6] add delegation to RBAC. Zhang et al. 2003 [39] present a rule-based delegation model (RDM2000) supporting role hierarchies and multi-step delegation. Kumar et al. 2002 [25] add contexts to RBAC. Hu and Weaver 2004 [18] employ contexts and context types in the healthcare domain that is also a basis of our authorization contexts in Section 4.4.

Jürjens 2002 [21] presents the extension UMLsec of UML that allows one to formally express a larger range of security-relevant information. Without OCL, UMLsec has no support for various RBAC constraints like separation of duty [12]. Basin et al. 2006–2011 [7–9] present the security modeling language SecureUML for modeling RBAC as metamodel using OCL for access control policies to enhance Model Driven Architecture (MDA) with Model Driven Security. From models they can automatically generate access control infrastructures for applications. Based on UML profiles it is unclear how different tools handle this extension mechanism.

Ray et al. 2004 [26] use UML class diagram *templates* and OCL to specify classical RBAC authorization constraints. Constraint violations are visualized using UML object diagram templates. Templates are instantiated with values from applications. Based on the delegation model RDM2000 [39], Sohr et al. integrate delegation and revocation [34]. Kuhlmann et al. 2013 [23] follow the approach of [26, 34]. They add a domain-specific language (DSL) for handling time-dependent dynamic constraints to their RBAC metamodel. Their workbench is USE [15], with a SAT-based model validator.

The modeling approaches presented above exclusively center around RBAC and fail to support permission-based [11] DAC, attribute-based ABAC [19, 24], or a combined model. Existing systems, like the identity management system described in Section 5.3, however, require a combined model to appropriately model and

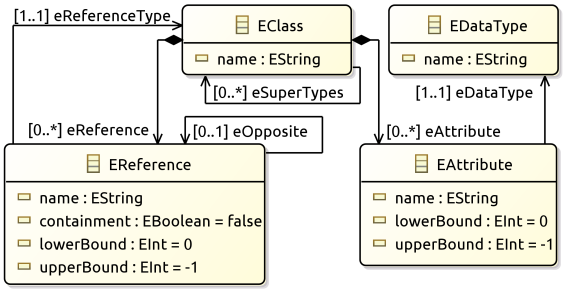

Figure 1: Simplified Ecore

validate access control. Implementing discretionary access control using RBAC [27] or using ABAC for roles [24] would not reflect the underlying software architecture.

We extend the work of Kuhlmann et al. [23] by making the access control model more modular, succinct, and general. Apart from RBAC also permission-based [11] DAC and attribute-based ABAC [19, 24] are integrated. For validation, we exclusively use the EMF [35]. Also Abomhara and Lazrag [2] report that OCL with EMF worked smoother.

Despite the number of static analysis approaches for software security in general, there is little work that addresses the problem of reconstructing and visualizing the security architecture of software systems. One exception is the technique introduced by Abi-Antoun and Barnes [1]. They annotate the source code of an application to extract Ownership Object-Graphs statically. Ownership Object-Graphs represent a hierarchical runtime-architecture of the objects within a system. Furthermore, they compare the extracted graph with a given DFD to identify forbidden data flows. The approach has only been tested for small and non-distributed applications (about 3,000 lines of code) [31]. Inspecting *existing* software for our methodology is based on earlier reverse engineering work [10, 22, 33].

3 BACKGROUND

As the formalism for representing our access control model we use Ecore and OCL from the Eclipse Modeling Framework (EMF) that is similar to UML and OCL. UML models and OCL are typically used to capture requirements and design decisions. OCL is crucial to model desired system properties, invariants or constraints accurately. UML together with OCL allows one to check for model consistency, reachability, and absence of contradictions [17]. Using models has the advantage of being independent of a particular implementation or platform. There are different kinds of diagrams in UML, but we only use class and object diagrams. The latter represent concrete instances of classes and associations that are defined in class diagrams.

Class diagrams with OCL constraints and object diagrams allow for (partial) verification and validation. Thereby verification answers the question “Are we building the product right?”, whereas validation answers the question “Are we building the right product?”.

Ecore is the meta model of the Eclipse Modeling Framework (EMF) for writing models. We use it for representing our access control model and for the software systems we are going to evaluate. A simplified subset of Ecore concepts given by Steinberg et al. [35] is shown in Figure 1 in EMF’s own notation. The class

- `EClass` represents a type with super types and contains references as well as attributes,
- `EAttribute` represents a typed attribute,
- `EDataType` represents the data type of an attribute,
- `EReference` represents an association between the containing class and the class referenced by `eReferenceType`.

Associations are drawn as arrows and containment is a special kind of association that is drawn with a black diamond. Using `eOpposite`, two associations can be declared to be inverse associations of each other.

Both attributes and references of a class may be multi-valued having a cardinality of 0 (meaning optional) or greater or -1 (meaning unlimited). A single (required) attribute or reference would have a lower and upper bound of 1.

In contrast to UML—as supported by the USE tool [15]—, associations in Ecore may be unidirectional largely supporting our modularity. Classes may be referenced without being changed. Like in UML, inheritance is drawn with a hollow triangle. Dashed lines point to interfaces whereas solid gray lines point to possibly abstract super classes as shown in some other class diagrams.

4 APPROACH

This section is subdivided into four subsections. Firstly we present our modeling methodology in Section 4.1. After introducing our RBAC model based on Ecore in Section 4.2, we add role-based *delegation* in Section 4.3 to make the (static) user assignment of roles more dynamic. The final Section 4.4 explains our modeling of fully flexible *authorization contexts* that corresponds to basic attribute-based access control (ABAC).

4.1 Methodology

We view a software system very abstractly as an instance of the software model depicted in Figure 2.

We simulate a finite number of *traces* and *states* of a given system. A single trace is composed of a sequence of *executions*. We use traces to model use cases of the software system. For an execution, a *subject* applies an *operation* to a starting state that leads to an ending state that in turn is the starting state of a subsequent execution within a *trace*. The changes of an operation to the state are modeled using OCL expressions in the operation’s *execute* method. Operations can access resources or call other operations that are resources as well. The underlying states are mere collections of *resource* values. This software model is the foundation of our access control model elaborated later. To create a model instance for a concrete software system we are using the following steps.

Step 1 In the first step, we create an instance model of the software under investigation. Therefore, we start by creating instances for operations of the software. We create resource objects that suitably describe an initial abstract *state* of our underlying software. We also establish the `accesses` relations between operations and resources that again may be

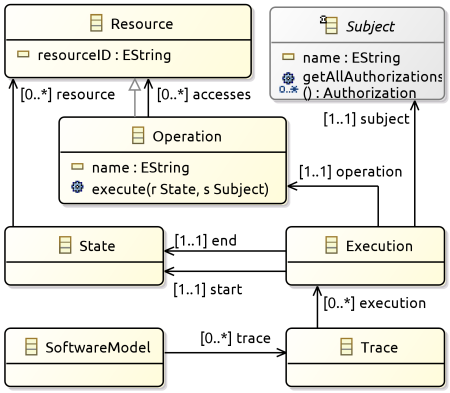

Figure 2: Software Model

operations. Finally, we create OCL expressions describing the state changes caused by operations.

Step 2 In the second step, we add the access control aspects to our instance model by creating instances for all known roles and subjects, see Section 4.2 for details. Furthermore, we construct the *intended* access control *policy* of the software, for example, by concrete role assignments and instances of any RBAC constraints. At this stage, consistency checks (i.e. role exclusion) can already be performed.

Step 3 Then, we model how our application enforces access control decisions. This *mechanism* is extracted from the implementation, i.e. the Java source or byte code. Having object diagrams for the *policy* and the *enforcement* allows further consistency checks and the assessment or improvement of the underlying software. Showing that the enforcement implies the policy for any object constellation amounts to *verification*, which, however, is beyond our current scope.

Step 4 In this step, we create expected *executions* and traces as objects in our object diagram, for example, based on requirement specifications, documentation of business processes, or use cases. From initial states and OCL expressions of Step 1 intermediate and ending states are calculated.

Step 5 In the last step, we identify traces that lead to states violating invariants imposed on state changes or invariants of the access control policy from Step 2.

All aforementioned steps could be done manually for some imagined software. The main idea, however, is to (automatically) extract the objects for *all* above steps from the actually deployed software with associated artifacts or even recorded runs. This is exemplified in Section 5.3 where we use the Soot analysis framework [38] with extra plugins to extract the information we deem to be relevant. The reconstructed security architecture helps not only to detect vulnerabilities via audits, but is also a good basis for a model-driven improved development. The Soot plugins and configuration parsers are specific for application frameworks but may be reused for software systems based on comparable frameworks. Reuse is certainly likely for revised versions of the same software.

4.2 Modeling RBAC

Figure 3 shows parts of our modular access control model as an Ecore class diagram.

The class `Authorization` is introduced as a superclass of `Role` and `Permission` to mediate between subjects and individual permissions for both DAC without roles and RBAC using roles. The class `Permission` is reused to represent the permission-to-role assignment, that is the PA relation of the RBAC96 model [29]. The abstract class `Authorization` is our concession to *modularly* compose DAC and RBAC policies that are both often–and even simultaneously–used in *existing* software. For a pure DAC policy roles could be entirely ignored. The class `Permission` models a description of authorized interactions with *resources*.

RBAC96 features already dynamic/time-dependent *sessions* that require one to check if *activated* roles (given by RBAC96's `roles` function) are a subset of the roles that are *assigned* to a user (given by RBAC96's UA relation). The possibility, to only activate a subset of a user's roles, supports the principle of *least privilege*. Only roles required for a task at hand need to be activated. For tasks in different sessions, other roles can be activated.

In our model, the class `Person` corresponds to *users* of RBAC96 and *sessions* are modeled via the class `Login`. Note that the classes `Person` and `Login` both inherit from `Subject` and that several logins may be associated to a single person. (The opposite reference of `login` corresponds to RBAC96's `user` function.) The *activated* roles of a login are obtained via the method `getAllAuthorizations`, whereas the *same* method applied to a person yields the *assigned* roles. The OCL invariant in Listing 1 checks if all *activated* roles of a login `l` are included in the *assigned* roles.

Listing 1: Session check

```
context Person:
invariant sessionCheck: login → forAll(l |
  getAllAuthorizations() →
    includesAll(l.getAllAuthorizations()));
```

Only the type of the involved role sets is generalized to sets of *authorizations*. Permissions are not regarded as authorizations for a pure RBAC policy.

The method `getAllAuthorizations` has to be distinguished from the reference `authorization` and from the method `getAuthorizations` of the class `Authorization` in that only this method properly collects *all* roles. Further *implicit* roles may be contributed by a role hierarchy, by delegations or *modularly* by future extensions. For the basic $RBAC_0$ of RBAC96 *without role hierarchies and constraints*, the reference `authorization` alone would be enough to know as it represents the *explicitly* associated roles.

In the following two subsections we describe our modeling of role hierarchies and authorization constraints respectively.

Role Hierarchies. We modularly extend $RBAC_0$ with role hierarchies to $RBAC_1$ by adding the subclass `HierarchicalRole` of the class `Role` with an additional `junior` (and inverse `senior`) association. This differs from the usual modeling in the literature [23, 26] where a junior association and numerous auxiliary methods are directly added to the `Role` class. The separate class `HierarchicalRole` allows for a static distinction and modular composition that avoids

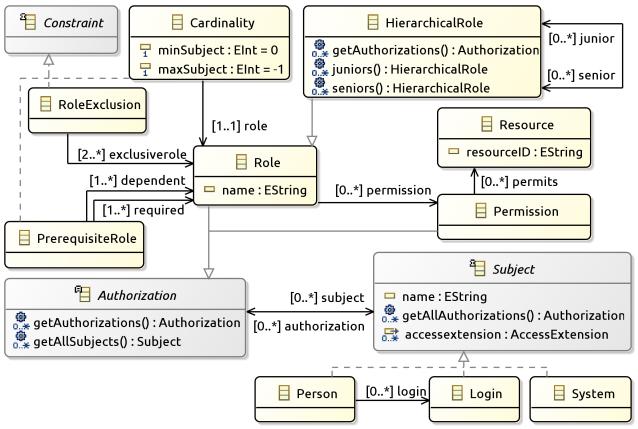

Figure 3: RBAC class diagram

unnecessary dynamic checks during validation for policies without a hierarchy.

Only the usually few or even empty *direct* junior roles need to be given for a hierarchical role. (For a proper hierarchical role either the junior or the inverse senior association should not be empty.) The *transitivity of role hierarchies* can be derived via computing the transitive closure—this is a built-in OCL operation—of all proper juniors and seniors, respectively, as given in Listing 2.

Listing 2: Transitive junior roles

```
operation juniors():HierarchicalRole[*]
{ body: junior → closure(junior); }
```

We compute *all* authorizations of a subject by overwriting the method `getAuthorizations` of class `Authorization` within the subclass `HierarchicalRole` to return also the additional junior roles. The implementation of `getAllAuthorizations` in the class `Subject` picks up all explicit and implicit roles (and further authorizations due to access extensions like delegations) for a proper session check and other access constraints.

We consider this use of overwriting as the least evil for effectively modeling role hierarchies as modular as possible and (mostly) independent from other authorization constraints. An alternative way would be to treat hierarchical roles as an access extension[1] like delegation as explained later.

The *reflexivity* of role hierarchies can be left implicit but not *anti-symmetry*. Anti-symmetry means that if several roles form a cycle via their junior relation, then all roles of that cycle are actually a single identical role. Therefore we ensure that *role hierarchies are non-cyclic* by the invariant in Listing 3. (Another option would be to regard a cycle as a single role with multiple alias names, but that would allow one to specify unintended cycles by accident.) As a side effect the given invariant also prevents explicit reflexivity.

Authorization Constraints. Having covered $RBAC_0$ and role hierarchies for $RBAC_1$, we now present the typical role-based authorization constraints that are all added independently. Other constraints

[1]The method `getAuthorizations` of class `Authorization` could be avoided and the class `HierarchicalRole` would need to implement the interface `AccessExtension`.

Listing 3: Non-cyclic role hierarchy

```
invariant nonCyclicRoleHierarchy:
juniors() → excludes(self);
```

Listing 4: Cardinality constraint

```
invariant cardinalityCheck:
let num:EInt = role.getAllSubjects() → size()
in num ≥ minSubject and
   (maxSubject = −1 or num ≤ maxSubject);
```

may be added as needed. The `Constraint` interface is basically a common purpose indicator. (According to Sandhu, $RBAC_0$ with constraints is $RBAC_2$, whereas $RBAC_1$ and $RBAC_2$ is $RBAC_3$.)

The following invariants are straightforward; they rely on the method `getAllSubjects` from the class `Authorization`. This method returns all subjects that explicitly or implicitly obtained the given authorization/role. The methods `getAllSubjects` and `getAllAuthorizations` are opposites of each other like the explicit references `subject` and `authorization` are between the corresponding two classes. Only for mere $RBAC_0$ or DAC there is no difference between the references and the generalized methods.

Instead of computing invariants, it is also possible for diagnostic output to compute those subjects that are violating an invariant. A generic invariant as part of the `Constraint` interface may then be that the number of violations is zero.

In any case hierarchical roles and access extensions like delegation are transparently considered due to the methods `getAllSubjects` and `getAllAuthorizations`. The following paragraphs show OCL invariants for cardinality constraints, mutually exclusive roles, and prerequisite roles (Listings 4, 5, 6).

Cardinality. An often employed authorization constraint is a restriction of the number of users/subjects having a certain important role. In our access control model with a possibly dynamic user-to-role association we can ensure the validity of a *cardinality constraint* by the OCL invariant given in Listing 4.

Mutually Exclusive Roles. For static or role-based *separation of duty* (SoD) it is important to support *role exclusion*. In port community systems, for example, a shipper must not play the role of customs because otherwise the shipper could control and approve its own goods. In a healthcare context, the role of an internal doctor should be different from a role of an external specialist to ensure independent judgments. To model such scenarios, two or more roles can be specified to be pairwise mutually exclusive so that a subject can have at most one of those roles. The relation `exclusiveRole` relates each role to other excluded roles. The OCL invariant for role exclusion is shown in Listing 5.

Prerequisite Roles. Other desirable authorization constraints are *prerequisite roles*. For certain roles, other roles may be defined to be prerequisites. This is similar to a role hierarchy as this constraint also forms a partial order. In contrast to a role hierarchy where a subject *implicitly* inherits *junior* roles, *prerequisite* roles must have been assigned in advance. Similar to role hierarchies cycles for

Listing 5: Role exclusion

```
invariant noSubjectsAssignedToExclusiveRoles:
exclusiverole.getAllSubjects() →
  forAll(s|s.getAllAuthorizations() →
    intersection(exclusiverole) → size() ≤ 1);
```

prerequisite roles should be avoided since assigning a single role of such a cycle to a subject would directly violate the constraint. Disregarding cycles, the OCL invariant for prerequisite roles is shown in Listing 6.

Listing 6: Prerequisite roles

```
invariant prerequisiteRoles:
dependent.getAllSubjects() →
  forAll(s|s.getAllAuthorizations() →
    includesAll(required));
```

4.3 Delegation

An important access control concept that we discuss here is role-based delegation. Delegation means to *temporarily* transfer rights from one subject to another. We model delegation as presented in Figure 4.

- `Delegation` models a `grantor` that can delegate a role to a `delegate` if the grantor has the `requiredGrantorRole` and the delegate has the `requiredDelegateRole`.
- `DelegationRelation` merely groups together the three roles `requiredGrantorRole`, `requiredDelegateRole`, and the `delegatedRole` in order to make them reusable for different subjects being grantors or delegates.
- `Duration` is the duration of a delegation. The delegation is automatically revoked after the given timeout.
- `MaxNestingLevel` defines how deep delegations (given by the `forwardings` and inverse `backtrace` links) may be nested.
- `MaxDelegations` is a (kind of cardinality) constraint that limits the number of subjects that can be delegates.

The extra delegation relation should stress the view that delegation is a *ternary* relation between roles: the required role of a grantor, the required role of a delegate, and the delegated role. Any member having the required role of a delegate may potentially also have the delegated role—provided that a grantor with the required grantor role actually issued the delegation successfully. A typical constellation, however, is that the required grantor role is equal to the delegated role. In Figure 6 a doctor delegates his role to a specialist that has the required role of a delegate. A required role may have been obtained via an explicit role assignment or implicitly via a senior role or via a *prior* delegation. In fact, the possibility to delegate a role further is an important aspect of delegation that can be controlled via the maximal nesting depth.

An actual delegation object is created whenever a role is successfully delegated. A delegation object points to the grantor and the delegate. We have a delegation chain if a delegate is also the grantor of the same role to another delegate in a subsequent delegation. To obtain the delegated roles of a subject, all delegations need to be collected. If the subject is the delegate, then the subject also has

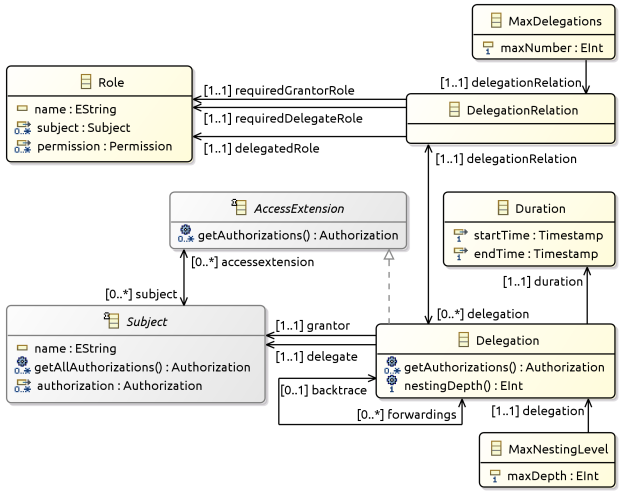

Figure 4: Delegation class diagram

Listing 7: Depth of nested delegations

```
operation nestingDepth():EInt
{ body: if backtrace = null then 1
        else 1 + backtrace.nestingDepth()
        endif }
```

the delegated role. If the delegated role is hierarchical, then also all junior roles are delegated implicitly.

Like hierarchical roles, delegation extends access. The difference is that delegations add roles to subjects, whereas hierarchical roles are only indirectly related to those subjects that have a senior role. This makes treating hierarchical roles and delegation in a uniform way awkward. We preferred two generic ways to extend access. The first generic way is to override the method `getAuthorizations` from `Authorization` as done for hierarchical roles. The other way is by implementing the interface `AccessExtension`. Every instance of an `AccessExtension` adds further authorizations to its connected subjects via the method `getAuthorizations` that needs to be implemented. For a delegation as an access extension the delegate is the only connected subject and the result returned by `getAuthorizations` is the singleton set of the delegated role. When computing all authorizations of a subject or all subjects of an authorization, all instances of access extensions are taken into account.

There seems to be no need to delegate a role that is already explicitly or implicitly assigned to subject, but it may make sense to create a new delegation before an old (otherwise identical) delegation *times out*. Without a reasonable timeout, delegation is unnecessary because roles should be directly assigned (by an administrator) for *long terms*. The duration is also important for *nested* delegations. We require that a forwarded delegation cannot last longer than the original delegation. Otherwise delegation chains would be broken apart if delegation objects are removed after their timeout. This is not crucial but changes the nesting depth of longer lasting

forwarded delegations. The depth of nested delegations can be recursively computed along the `backtrace` links as shown in Listing 7. The constraints `MaxLevel` and `MaxDelegations` are added on top of the core delegation concept. For the maximum number of delegations, we decided to count delegations for every delegation relation separately as shown in Listing 8.

Listing 8: Maximum number of delegations

```
invariant maxDelegations:
delegationRelation.delegation → size() ≤ maxNumber
```

A separate delegation relation is strictly unnecessary but a design choice to increase modularity. The three roles of an delegation relation make up a *kind* for delegations. Each delegation does not need to refer to the three roles directly but only to a single and usually shared delegation relation. The presented delegation model is still experimental as it differs from earlier work [34, 36]. Yet, we consider our design to be more modular and clearer. Not only for the sake of brevity, we omit explicit revocations and entirely rely on timeouts. As mentioned before, classical user-to-role assignments should be used in favor of delegations without timeouts. For exceptional premature revocations, administrators may be allowed to reduce the durations of delegation chains.

4.4 Authorization contexts

As motivated by Kumar et al. [25] and many others [4, 5, 18, 23, 34] we want to support context-awareness. A context constraint is a condition that must be fulfilled whenever a *subject* is going to interact with a *resource* which may also mean to execute an operation. (We view operations as resources, too.) In contrast to the rather static role constraints, context constraints allow one to tune access in an arbitrary, very dynamic and fine-grained way.

For example in healthcare, a doctor may be allowed to only read or modify the records of her own patients. Another scenario may allow a doctor to read the data records of a type that matches the doctor's specialization. In a PCS, a port order may only be administered by those members of a company or department that issued this order; a policy termed *multi tenancy*. Data of different users can be kept apart, despite all users having the same (or overlapping) roles (or even no roles at all).

Generally, granting permissions may depend on any part of a concrete system state (and its history), for instance on the time (of a day), the location (from which a person has logged in), or the type of accessed resources corresponding to history-based, time-based, location-based, or type-based access control, respectively. Hu and Weaver [18] identify the following data as basic for context constraints:

- `PersonID` - the one sending the request;
- `ResourceID` - the particular resource to consult;
- `ResourceType` - the type of the resource;
- `Time` - the date of issuance of this request;
- `Location` - the location of the access request.

A context constraint is an arbitrary Boolean formula taking a subject and a resource (or their identities via IDs) as input; it can be evaluated for a concrete system state, i.e. for concrete values of the time, location, person and resource. We include context constraints

Listing 9: Context constraint

```
abstract class ContextConstraint { interface } {
operation isAllowed(s:Subject, r:Resource):Boolean;
}
```

by merely adding the interface `ContextConstraint` with a single `isAllowed` Boolean operation.

Implementations must be supplied as needed. The Boolean function can be seen as a basic ABAC policy rule that describes for any subject and resource if access is allowed or denied. Relevant and reliable attributes to be queried must be made accessible via domain-specific subclasses of the classes `Subject`, `Resource` (see Figure 5) and `State`. (We assume here that the current state of a system is globally accessible to avoid an extra argument.) An example, demonstrating multi tenancy, is shown in Listing 10 where company attributes of subjects and resources must match. Time-based, location-based, or type-based access control is merely a question of attribute choices. Like time also history[2] may be part of the current state allowing for history-based access control without a temporal logic. With proper attributes we can also simulate *mandatory access control* (MAC) [11, 28], a *Chinese Wall policy*, or *dynamic separation of duty* (DSoD) [4] where privileges may depend on decisions made in the past. Even roles can be seen as attributes with an obvious policy. However, we do not want to combine RBAC with ABAC in an attribute-centric way but in a role-centric way [24]. For a runtime simulation of access control policies or enforcements, all instances of context constraints need to be evaluated. The conjunction of all `isAllowed` results must be true. Beforehand, we additionally check the *static* authorizations via roles.

Like Hu and Weaver [18] we want to group our context constraints by type. A *context type* is just a restriction of the *kind* of formulas in context constraints. We can do so by introducing subclasses of `ContextConstraint` as shown in Figure 5. The implementations `TimeConstraint`, `TypedConstraint`, and `CompanyCheck` (see Listing 10) are application dependent and based on corresponding subclasses `SubjectWithResourceType`, `CompanySubject` and `CompanyResource` of the respective `Subject` and `Resource` known from Figure 3. `TimeConstraint` is supposed to constrain access to opening hours whereas `TypedConstraint` models type-based access control so that subjects can only access specifically typed resources.

Typically, ABAC policies can be specified in sophisticated ways, for example, extra application conditions may be given for either allowing or denying access. Denial is negation of allowance and applicability conditions correspond to implications. In any case, eventually, allowing or denying access is a binary decision made by our `isAllowed` operation. OCL may not be the best formalism to express context conditions, but OCL is crucial for our simulation methodology and we hope to translate ABAC policies automatically to OCL.

Future work. Like our delegation we regard these authorization contexts as experimental and hope to further conform to proper

[2] Storing large parts of the history is usually necessary anyway for mere accountability reasons.

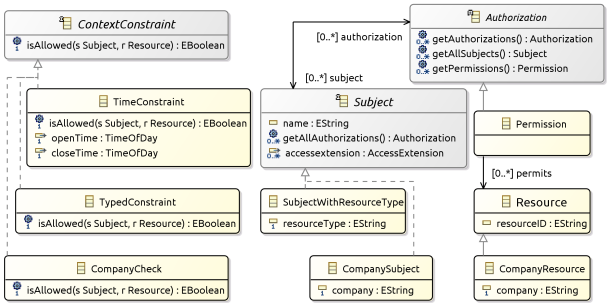

Figure 5: Domain-specific class diagram

ABAC on the one hand and also allow context conditions to be less role-agnostic on the other hand. Already Kumar et al. [25] considered a role context between a subject and a resource context. A role could not only describe a job function but also list a couple of attributes that are guaranteed to exist for role-centric context checks. From role-aware contexts, viewed as roles with contexts, also delegations could profit, allowing roles restricted by a context to be delegated or to be required for a delegation.

5 EVALUATION

The evaluation of our modeling approach is carried out via case studies in three specific domain areas. We evaluate a Port Community System (PCS), a clinical information system (Section 5.2), and an identity management system from industry. These systems employ different access control policies and we show how they can be appropriately modeled using our modular model. According to our methodology, we inspect byte code and extract the implemented access control policy for the identity management system in Section 5.3.

5.1 A port community system

The international port community system association (IPCSA) [20] defines a PCS as a neutral and open electronic platform enabling intelligent and secure exchange of information between multiple systems operated by a variety of organizations that make up a seaport community. Our PCS is an application based on JavaEE technologies and proprietary frameworks. The software administers, records, and controls the communication between the different parties of the port area. Many features like data hosting (cloud service) or software as a service (SaaS) are accessible from anywhere via the web, which naturally poses problems with respect to data access control. Some of the numerous involved parties are listed below.

- Shippers or carriers actually want to export, import, or simple transfer goods via the port. They initiate port orders.
- Shipowners provide the transport capabilities. They receive copies of port orders for their ships and can themselves issue orders (like shippers) to maximally utilize their cargo space.
- The customs has ultimate control over the goods being imported or exported. It does so by selecting and controlling a small sample of containers or conventional goods. The PCS provides movement information to the customs and redirects customs clearances or customs control orders back to the affected parties.
- The port authority is informed about port orders involving dangerous goods to ensure timely and appropriate response in case of hazards.
- The Terminal Operating System (TOS) administers storage locations at the port and controls required clearances before goods are actually moved. All related port orders are transmitted via the PCS. (The storage locations determine the responsible customs office.)
- The tally is responsible for all packing, unpacking, loading, unloading, or (container) movement orders at the port.
- The railway is an alternative for transport orders.

The original port order of a client (i.e. a shipper) is supplemented by the PCS with additional data like ship or storage location data. The completed order will be distributed to the other involved parties but usually without (client) data that are not relevant for them.

Discussion. For our modeling we took the above list of parties as roles and associated permitted tasks to them. An important task of the customs is to supply (or refuse) clearance to a port order. According to Step 1 of our methodology in Section 4.1, port orders are resources that are modified by operations. Especially for a customs clearance we required that only the customs can perform this task and that there is only a single customs at all (which we assume to be true for a single country). Thus for the role customs we introduced the *cardinality constraint* as shown in Listing 4 with `maxSubject` being 1. Furthermore, we added a static role exclusion to prevent that the same subject[3] can both create and supply clearance to port orders. For the invariant of role exclusions see Listing 5.

Apart from creating a port order, shippers may also modify or cancel port orders, but only those that have been created by the shipper's company. This is a classical *context constraint* known as *multi-tenancy* that can be checked by implementing `isAllowed` as given in Listing 10, where we carefully do not restrict access of subjects to resources that are not related via the company subtypes.

Listing 10: Company context constraint

```
class CompanyCheck extends ContextConstraint
operation isAllowed(s:Subject,r:Resource):Boolean
{ body:
 not s.oclIsKindOf(CompanySubject) or
 not r.oclIsKindOf(CompanyResource) or
 s.oclAsType(CompanySubject).company =
  r.oclAsType(CompanyResource).company }
```

We created EMF object diagrams for the PCS based on (semi-) formal specification and documentations of the business processes. This corresponds to Step 2 of our methodology. Source or byte code was not disclosed to us, therefore we could not evaluate if access control is actually *enforced* as intended according to our methodology's Step 3.

Result. The modeling allowed us to design a target *policy* and check its consistency (Step 4 and Step 5). The main result is that

[3]Due to the cardinality constraint there can only be one subject—e.g., the customs system called "Atlas" in Germany—that must not also play the shipper role.

we identified role exclusions, cardinality constraints, and no role hierarchy—this is $RBAC_2$—, but also dynamic *multi-tenancy* context constraints. Due to the separation of hierarchical roles our object diagrams are not cluttered with empty references to junior or senior roles.

5.2 A clinical information system

Gerdes evaluated a role-based security concept for clinics considering trends of electronic health data management [14]. Via interviews she investigated the actual process flow within clinics in order to derive suggestions for improvements, especially with regard to protecting sensitive patient data. Usually, clinics feature four ($RBAC_0$) roles: doctors, nurses, allied healthcare professionals and support staff.

Tietjen took the above work as case study for validating an RBAC model with OCL constraints [36]. He considered two roles: receptionists and doctors. Patients have no roles, they are represented as mere data records in the clinical information system. A receptionist assigns an arriving patient to a hospital ward. Each ward has a responsible doctor. There is a role exclusion between receptionists and doctors. The receptionist has only access to the identifying data of patients, whereas a doctor has only access to electronic health records of patients of her ward. Viewing the ward as a location, we can establish a *location context constraint* similar to the company context constraint of Listing 10.

We also considered a cardinality constraint stating that there is *exactly one* doctor responsible for each ward. However, this is no static cardinality constraint for the number of *all* doctors, therefore our RBAC cardinality constraint in Listing 4 is not directly applicable.

Discussion. Having exactly one responsible doctor poses the question what should happen if a doctor is not able to properly treat all patients? One solution may be to ask the receptionist to assign a patient to another hospital ward. Another way may be to ask a senior doctor responsible for several wards or the whole clinic. A senior doctor would be a part of a role hierarchy ($RBAC_1$) allowing at least the accesses of all junior doctors.

In order to avoid costly changes of user-to-role assignments—to be done by administrators—, permission-to-role assignments tend to include more permissions than necessary to avoid a potential critical lack of permissions. This contradicts to the principle of least privilege. A more appropriate way is to use delegation. A doctor could simply delegate her responsibility to another colleague. However, it makes no sense to delegate the doctor role to another clinician who has already this role. Our role-based delegation model has no idea about doctors for different wards as long as there is only one doctor role. We need different doctor roles for every ward. This would also solve the above cardinality constraint, but such a role inflation might be undesirable. Therefore we are still looking for improvements of our access control model. An idea may be to attach contexts to roles and to obtain something like *parameterized roles*, i.e. roles parameterized by a ward attribute or an arbitrary context condition.

Another typical scenario is a clinical consultation. A patient's data must be made accessible to an external specialist. Figure 6 illustrates how a doctor delegates her role to a specialist to share

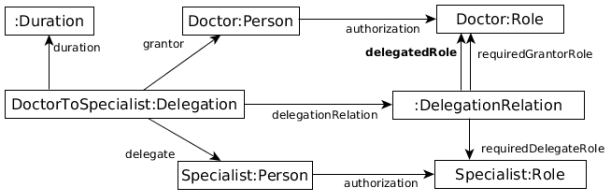

Figure 6: Delegation example object diagram

a patient's data. The delegation relation connects to the two *required* roles and to the *delegated* doctor role (bold connection). The delegation object `DoctorToSpecialist` points to a duration, a delegation relation, the grantor `Doctor`, and the delegate `Specialist`. The doctor has the required grantor role and the specialist has the required delegate role.

Assuming that the specialist is an external person we could restrict the delegation to a maximum nesting depth of one—using an object instance of class `MaxNestingLevel` with a `maxDepth` value of 1 (see Figure 4)—to disallow that the doctor role is delegated further.

An implementation of a clinical information system (CIS) could not be inspected within this case study. Only reports of users about shortcomings of existing systems have been considered. Due to sensitive patient's records the healthcare domain is a particularly challenging field for access control as also Hu and Weaver [18] and [2] show.

Result. The aim of this case study was to demonstrate that our RBAC model with OCL constraints and delegation was suitable to properly model the demanding policies of a clinical information system. Apart from validating consistency for intended target policies of a CIS using EMF [35] we also translated Ecore to UML and employed the USE tool [15] in order to compare validation results. Regarding tool support for OCL there is certainly room for improvements as also Abomhara et al. [2] report. Regarding the modeling, the case study shows that context constraints may clash with role-based delegation for a single doctor role. Using mere context constraints or ABAC-like access rules is certainly possible, but one still wants more or less fine-grained roles and flexible context-sensitive delegations. Otherwise a critical lack of permissions or—the opposite—violating the principle of least privilege are likely problems.

5.3 An identity management system

In a joint project with a large company the aim was to check the company's access control policy for business partners against the information stored in a separate identity management repository. The repository stores information on persons, logins, contracts, permissions, roles and the corresponding assignments. For the corresponding identity management system (IdMS) the company provided us the binaries and sources of their code. The software is divided into three JavaEE components using the technologies Java Server Faces (JSF), Spring and JBoss Seam. Furthermore, the Java Persistence API (JPA) and the Cassandra API are used for database connections. For specifying the policy the company supplied us with business process model and notation diagrams.

Listing 11: Model-to-model transformation excerpt

```
transformation DataModelToAccessControlModel (
  in input: low, out output: acm);
main () {
  input.rootObjects() [login]
    → map toLogin();
  input.rootObjects() [person]
    → map toPerson();
}
mapping login::toLogin()
  : permissions::Login {
}
mapping person::toPerson()
  : permissions::Person {
  logins := self.contracts
    → collect(profiles)
    → collect(logins)
    → resolve(permissions::Login) }
```

The challenge was to automatically translate the implemented RBAC model to our formal approach. Therefore, we (a) used reverse engineering to extract an internal data-model definition, (b) created a transformation for data-model instances into object diagrams of our formal model, (c) identified permissions and roles provided by a web front-end, (d) transformed this enforcement, and (e) semi-automatically extracted expected executions. The tasks (a) and (b) accomplish Step 1 of our methodology in Section 4.1. For Step 2 we manually defined the workflow and the *intended* policy based on supported business process model and notation diagrams. Tasks (c) and (d) correspond to Step 3 and the task (e) matches Step 4 that is the basis for Step 5, namely to check the extracted object diagrams against the specifications of Step 2. In the following, we explain the aforementioned tasks in more detail.

(a) We reverse engineer the software's data-model by evaluating the JPA annotations **@**Entity, **@**Embeddable, **@**Table, **@**Column, **@**ManyToOne, **@**OneToMany, and **@**Transient present in byte code. The static analysis transfers the annotations and the type information into a valid Ecore model with 115 classes. The data stored in the data base can be automatically converted into an object diagram of the extracted Ecore model. The extraction and conversion is transferable to other programs using JPA.

(b) For extracting the resources and creating the initial state, 14 of the above extracted classes hold relevant information for the transformation to model. For these classes we implemented a model-to-model transformation using QVTo [37]. The transformation translates object instances of the internal data-model into resource instances of our software model. Since this transformation depends on the used data-model, it is not transferable to other programs.

(c) In this step, we implemented a static analysis to extract enforced permissions and roles. Starting from identified program entry points we collected calls to the IdMS-API method for checking whether a user has a certain authorization. This method—actually named `hasPermission`—receives a string argument denoting the authorization label that is required to execute the code. The labels are following a naming convention. For example, role names started with the prefix `role`. This led us to introduce the super class `Authorization` for roles and permissions. To reuse this analysis it is necessary to adapt it to other authorization APIs such as Apache Shiro. The `hasPermision` method can be easily replaced by another method to search for; the naming convention, however, is an artifact of the analyzed system.

(d) Similar to (b) we implemented a model-to-model conversion for subjects, permissions and roles. The excerpt in Listing 11 shows how multiple logins of persons are directly attached to the persons that are only indirectly given in the internal data-model via contracts and profiles. The available permissions and roles are stored in the same data class that can be distinguished by the prefix of the name attribute. The data-model provides means to express a hierarchy between the different elements. Since privileges are inherited to children, this may lead to the situation where a role can be the child of a permission, contradicting all approaches from academia. Furthermore, it is possible to directly assign permissions to a user, contradicting the RBAC96 model.

(e) The *executions* corresponding to Step 4 of our methodology were created semi-automatically based on the Seam configuration, the JSF servlets, and their usage dependencies. Therefore, we parsed the JSF files, extracted display masks and their usage relations, and identified calls to the Java implementation. In a manual step we then grouped these masks into higher-level processes and execution paths. The extraction of the masks and their usage relation can be used for any program that employs the aforementioned frameworks.

Results. We used several static analyses of the underlying byte code as well as configurations to obtain a model and used further model-to-model transformations to obtain model instances for validating the user and permissions against a manually defined access control policy. The analyzed identity management system combined RBAC and DAC features. As a preparation step of validation, we had to divide the model instances into independent parts to accelerate the analysis. This was necessary to overcome some memory issues because there were over a hundred thousand assignments in the generated instance model. The preparation also reduced the execution time for validating some constraints below 30 minutes. Generally, validations taking longer have been aborted and considered as failing thus leading only to a partial result.

Due to the modularity of our approach, the model instances could be given succinctly and cleanly without any—even void—connections to concepts like delegations. In particular for validation any avoided boiler plate or overhead is beneficial.

All the aforementioned reverse engineering tasks are implemented as plugins for the Soot analysis framework [38]. The majority of the analyses is transferable without or with only a few adaptions to other programs, so it is possible to transfer this method.

6 DISCUSSION

The approach we have introduced allows us to test the implemented and planned security policies against each other and find execution

paths that lead to insecure states from a policy perspective. Still, there are many possibilities to improve or enhance this approach.

At the moment, we focus on the intended usage of the application by modeling traces from existing use cases or business process model diagrams. Security flaws detected using these traces could be exploited by users in their normal daily work, even by accident. A shortcoming of this approach is that none of our partner companies provide misuse cases for their software. Misuse cases are use cases where a misuser reaches an attack target without the system preventing it [32]. Misuse case would allow us to identify vulnerabilities that an attacker could try to exploit. There are different ways to generate such misuse cases. One could create new executions and traces totally randomly, to simulate an attacker that tries attack steps in any order. Another possibility is to mutate existing traces and executions. This way one could simulate attackers who try to mess with the existing workflows to find possible security flaws. Nevertheless, it is obvious that this can improve the number of tested traces, but it is not possible to guarantee a secure software.

A disadvantage of our approach is the execution time of the model validation. Depending on the number of objects (including traces, operations, subjects, and policy entries) it can increase to a point where the results take longer than about 30 minutes to be calculated. There are different ways to improve this issue. At the moment we calculate all states of all traces before we validate the model. We observed traces where the model was insecure after a very small number of executions of a trace, but we still calculated and checked the complete trace. It would be possible to check each step directly after it has been calculated and stop a trace whenever an execution leads to an insecure state. A more elaborated approach would be to identify equal program states to avoid states that have already been inspected. Another alternative for our *execution traces* may be filmstrip models [16].

Our approach analyzes the implementation of productive software systems. However, this necessitates access to the respective implementation. In some of our projects, access was not so easy to obtain, as companies have security concerns about giving the software to external third parties. Therefore, we designed our methodology so that we can omit reverse engineering steps. As a result, a policy cannot be checked against the implementation, but nevertheless, traces can be checked against the policy.

The reverse engineering part of our approach is heavily dependent on the software frameworks used by the analyzed system. There is quite a large number of existing security frameworks for different languages and technical domains, e.g., JEE, Spring, Apache Shiro, and Android Framework. Since it is nearly impossible to support all frameworks, it is necessary to create new static security analyses for different authorization aspects that can be easily *configured* to support new frameworks instead of being tediously programmed. Our approach would benefit from analyses that extract the effect of operations to the state of our model. Otherwise the *manual* modeling of the behavior is a time-consuming and error-prone process.

7 CONCLUSIONS

We presented a novel modular access control model and a methodology to extract model instances from existing software.

Apart from DAC, we support classical RBAC concepts such as role hierarchies and authorization constraints like cardinality constraints, mutually exclusive roles, and prerequisite roles. Furthermore, role-based delegation as well as highly flexible but role-agnostic context constraints as a basic version of ABAC are covered.

The *modularity* of our access control model ensures the applicability to a wide range of software without any overhead for aspects not present in a given system. In particular for OCL validation, succinct model instances are highly desirable.

We evaluated our approach for three software systems from different domains. A port community system (PCS) and a clinical information system demonstrate our rich and modular modeling capabilities. For an identity management system we actually extracted model instances from Java byte code and configuration files according to our methodology. The violations of OCL invariants indicated vulnerabilities of the underlying system.

The three case studies also revealed certain shortcomings of our access control model left for future work. Despite unifying various concepts their interrelations may be improved. As an example we discussed the combination of role-based delegation and role-agnostic context constraints for the clinical information system. Also using EMF with OCL disclosed limits that are difficult to assess. However, we expect our approach to be rather stable and yet extensible.

ACKNOWLEDGMENTS

This work was supported by the German Federal Ministry of Education and Research (BMBF) under the grant 16KIS0583 (PortSec project) and the German Federal Ministry for Economic Affairs and Energy (BMWi) under the grant ZF4123903ED6 (Certified Applications) and the German Federal Ministry of Transport and Digital Infrastructure (BMVI) under the grant 19H18012E (SecProPort project). The work of Carlos Rubio-Medrano was partially supported by grants from the United States National Science Foundation (NSF-IIS-1527268 and NSF-ACI-1642031).

REFERENCES

[1] Marwan Abi-Antoun and Jeffrey M. Barnes. 2010. Analyzing Security Architectures. In *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering (ASE '10)*. ACM, New York, NY, USA, 3–12. https://doi.org/10.1145/1858996.1859001

[2] Mohamed Abomhara and Mehdi Ben Lazrag. 2016. UML/OCL-based Modeling of Work-Based Access Control Policies for Collaborative Healthcare Systems. In *2016 IEEE 18th International Conference on e-Health Networking, Applications and Services (Healthcom)*. IEEE, Munich, Germany, 1–6. https://doi.org/10.1109/HealthCom.2016.7749461

[3] Gail-Joon Ahn and Ravi Sandhu. 1999. The RSL99 Language for Role-Based Separation of Duty Constraints. In *RBAC '99: Proceedings of the Fourth ACM Workshop on Role-Based Access Control*. ACM, New York, NY, USA, 43–54. https://doi.org/10.1145/319171.319176

[4] Gail-Joon Ahn and Ravi Sandhu. 2000. Role-based Authorization Constraints Specification. *ACM Transactions on Information and System Security (TISSEC)* 3, 4 (April 2000), 207–226. https://doi.org/10.1145/382912.382913

[5] Gail-Joon Ahn and Michael E. Shin. 2001. Role-based Authorization Constraints Specification Using Object Constraint Language. In *Proceedings Tenth IEEE International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises. WET ICE 2001*. IEEE, Cambridge, MA, USA, 157–162. https://doi.org/10.1109/ENABL.2001.953406

[6] Ezedin Barka and Ravi Sandhu. 2000. Framework for Role-Based Delegation Models. In *Proceedings of the 16th Annual Computer Security Applications Conference (ACSAC '00)*. IEEE Computer Society, Washington, DC, USA, 168–176. https://doi.org/10.1109/ACSAC.2000.898870

[7] David Basin, Manuel Clavel, Jürgen Doser, and Marina Egea. 2009. Automated analysis of security-design models. *Information and Software Technology* 51, 5

(2009), 815–831. https://doi.org/10.1016/j.infsof.2008.05.011

[8] David Basin, Manuel Clavel, and Marina Egea. 2011. A Decade of Model-driven Security. In *Proceedings of the 16th ACM Symposium on Access Control Models and Technologies (SACMAT '11)*. ACM, New York, NY, USA, 1–10. https://doi.org/10.1145/1998441.1998443

[9] David Basin, Jürgen Doser, and Torsten Lodderstedt. 2006. Model Driven Security: From UML Models to Access Control Infrastructures. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 15, 1 (Jan. 2006), 39–91. https://doi.org/10.1145/1125808.1125810

[10] Bernhard J. Berger, Karsten Sohr, and Rainer Koschke. 2013. Extracting and Analyzing the Implemented Security Architecture of Business Applications. In *17th European Conference on Software Maintenance and Reengineering (CSMR 2013)*. IEEE, Genova, Italy, 285–294. https://doi.org/10.1109/CSMR.2013.37

[11] National Computer Security Center. 1985. *Department of Defense Trusted Computer System Evaluation Criteria*. Department of Defense. DOD 5200.28-STD (supersedes CSC-STD-001-83).

[12] Antanas Cenys, Andrius Normantas, and Lukas Radvilavicius. 2009. Designing role-based access control policies with UML. *Journal of Engineering Science and Technology Review* 2, 1 (July 2009), 48–50. https://doi.org/10.25103/jestr.021.09

[13] David F. Ferraiolo, D. Richard Kuhn, and Ramaswamy Chandramouli. 2007. *Role-Based Access Control* (2nd ed.). Artech House, Inc., Norwood, MA, USA.

[14] Stefanie Gerdes. 2007. *Rollenbasiertes Sicherheitskonzept für Krankenhäuser unter Berücksichtigung der aktuellen Entwicklungen in der Gesundheitstelematik*. Master's thesis. Universität Bremen.

[15] Martin Gogolla, Fabian Büttner, and Mark Richters. 2007. USE: A UML-based specification environment for validating UML and OCL. *Science of Computer Programming* 69, 1–3 (Dec. 2007), 27–34. https://doi.org/10.1016/j.scico.2007.01.013

[16] Martin Gogolla, Lars Hamann, Frank Hilken, Mirco Kuhlmann, and Robert France. 2014. From Application Models to Filmstrip Models: An Approach to Automatic Validation of Model Dynamics. In *Modellierung 2014 (LNI)*, Hans-Georg Fill, Dimitris Karagiannis, and Ulrich Reimer (Eds.), Vol. P225. Gesellschaft für Informatik e.V., Bonn, 273–288. https://dl.gi.de/20.500.12116/17056

[17] Martin Gogolla and Frank Hilken. 2016. Model Validation and Verification Options in a Contemporary UML and OCL Analysis Tool. In *Modellierung 2016 (LNI)*, Andreas Oberweis and Ralf Reussner (Eds.), Vol. P254. Gesellschaft für Informatik e.V., Bonn, 205–220. https://dl.gi.de/20.500.12116/825

[18] Junzhe Hu and Alfred C. Weaver. 2004. A Dynamic, Context-Aware Security Infrastructure for Distributed Healthcare Applications. In *First Workshop on Pervasive Security, Privacy and Trust (PSPT '04)*. Boston, MA, 8.

[19] Vincent C. Hu, D. Richard Kuhn, and David F. Ferraiolo. 2015. Attribute-Based Access Control. *Computer* 48, 2 (Feb. 2015), 85–88. https://doi.org/10.1109/MC.2015.33

[20] IPCSA. 2018. Retrieved January 4, 2019 from https://ipcsa.international

[21] Jan Jürjens. 2002. UMLsec: Extending UML for Secure Systems Development. In *UML 2002 — The Unified Modeling Language (LNCS)*, Jean-Marc Jézéquel, Heinrich Hussmann, and Stephen Cook (Eds.), Vol. 2460. Springer, Berlin, Heidelberg, 412–425. https://doi.org/10.1007/3-540-45800-X_32

[22] Rainer Koschke and Daniel Simon. 2003. Hierarchical Reflexion Models. In *10th Working Conference on Reverse Engineering, 2003. WCRE 2003. Proceedings.* IEEE, Victoria, British Columbia, Canada, 36–45. https://doi.org/10.1109/WCRE.2003.1287235

[23] Mirco Kuhlmann, Karsten Sohr, and Martin Gogolla. 2013. Employing UML and OCL for designing and analysing role-based access control. *Mathematical Structures in Computer Science* 23, 4 (Aug. 2013), 796–833. https://doi.org/10.1017/S0960129512000266

[24] D. Richard Kuhn, Edward J. Coyne, and Timothy R. Weil. 2010. Adding Attributes to Role-Based Access Control. *Computer* 43, 6 (June 2010), 79–81. https://doi.org/10.1109/MC.2010.155

[25] Arun Kumar, Neeran Karnik, and Girish Chafle. 2002. Context Sensitivity in Role-Based Access Control. *ACM SIGOPS Operating Systems Review* 36, 3 (July 2002), 53–66. https://doi.org/10.1145/567331.567336

[26] Indrakshi Ray, Na Li, Robert France, and Dae-Kyoo Kim. 2004. Using UML To Visualize Role-Based Access Control Constraints. In *Proceedings of the 9th ACM symposium on Access Control Models and Technologies (SACMAT '04)*. ACM, New York, NY, USA, 115–124. https://doi.org/10.1145/990036.990054

[27] Ravi Sandhu and Qamar Munawer. 1998. How to Do Discretionary Access Control Using Roles. In *Proceedings of the Third ACM Workshop on Role-based Access Control (RBAC '98)*. ACM, New York, NY, USA, 47–54. https://doi.org/10.1145/286884.286893

[28] Ravi S. Sandhu. 1993. Lattice-Based Access Control Models. *Computer* 26, 11 (Nov. 1993), 9–19. https://doi.org/10.1109/2.241422

[29] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. 1996. Role-Based Access Control Models. *Computer* 29, 2 (Feb. 1996), 38–47. https://doi.org/10.1109/2.485845

[30] Michael E. Shin and Gail-Joon Ahn. 2000. UML-Based Representation of Role-Based Access Control. In *Proceedings IEEE 9th International Workshops onEnabling Technologies: Infrastructure for Collaborative Enterprises (WET ICE 2000)*. IEEE, Gaithersburg, MD, USA, 195–200. https://doi.org/10.1109/ENABL.2000.883728

[31] Adam Shostack. 2014. *Threat Modeling: Designing for Security* (1st ed.). John Wiley & Sons.

[32] Guttorm Sindre and Andreas L. Opdahl. 2005. Eliciting security requirements with misuse cases. *Requirements Engineering* 10, 1 (Jan. 2005), 34–44. https://doi.org/10.1007/s00766-004-0194-4

[33] Karsten Sohr and Bernhard Berger. 2010. Idea: Towards Architecture-Centric Security Analysis of Software. In *Engineering Secure Software and Systems (LNCS)*, Fabio Massacci, Dan Wallach, and Nicola Zannone (Eds.), Vol. 5965. Springer, Berlin, Heidelberg, 70–78. https://doi.org/10.1007/978-3-642-11747-3_6

[34] Karsten Sohr, Mirco Kuhlmann, Martin Gogolla, Hongxin Hu, and Gail-Joon Ahn. 2012. Comprehensive two-level analysis of role-based delegation and revocation policies with UML and OCL. *Information and Software Technology* 54, 12 (2012), 1396–1417. https://doi.org/10.1016/j.infsof.2012.06.008

[35] Dave Steinberg, Frank Budinsky, Marcelo Paternostro, and Ed Merks. 2009. *EMF: Eclipse Modeling Framework* (2nd ed.). Addison-Wesley, Boston, MA.

[36] Daniel Tietjen. 2017. *Validierung eines RBAC-Ecore-OCL-Modells mittels des USE-Tools*. Master's thesis. Universität Bremen.

[37] Ulyana Tikhonova and Tim Willemse. 2015. Designing and Describing QVTo Model Transformations. In *Proceedings of the 10th International Conference on Software Engineering and Applications - Volume 1: ICSOFT-EA, (ICSOFT 2015)*. SciTePress, Colmar, Alsace, France, 401–406. https://doi.org/10.5220/0005556004010406

[38] Raja Vallée-Rai, Phong Co, Etienne Gagnon, Laurie Hendren, Patrick Lam, and Vijay Sundaresan. 2010. Soot: A Java Bytecode Optimization Framework. In *CASCON First Decade High Impact Papers (CASCON '10)*. IBM Corp., Riverton, NJ, USA, 214–224. https://doi.org/10.1145/1925805.1925818

[39] Longhua Zhang, Gail-Joon Ahn, and Bei-Tseng Chu. 2003. A Rule-Based Framework for Role-Based Delegation and Revocation. *ACM Trans. Inf. Syst. Secur.* 6, 3 (Aug. 2003), 404–441. https://doi.org/10.1145/937527.937530