Modeling and Validating Role-Based Authorization Policies for a Port Communication System with UML and OCL

Christian Maeder[*], Karsten Sohr[*], Rodrigue Wete Nguempnang[*], Nils Meyer-Larsen[†], and Rainer Müller[†]
[†]Institute of Shipping Economics and Logistics (ISL), Germany
[*]University of Bremen, Germany

ABSTRACT Modern sea or inland ports rely on digital communication and systems to boost rapid turnover of trade. Stakeholders like shippers, shipping lines, container terminals and port authorities collaborate and compete using their own legacy applications. Many sea ports operate Port Community Systems (PCS) to orchestrate processes between the players. These software systems are potential targets of security threats that may lead to payment fraud, espionage of competitors, smuggling, theft, export control violations, up to disasters involving dangerous goods possibly effecting public mains.
In our approach we apply modeling to the field of information security. We combine and focus on Role-Based Access Control (RBAC) with constraints and Attribute-Based Access Control (ABAC) for finer grained authorization constraints. In a concrete case study we model authorization policies within port communities that partly utilize dedicated PCS. The purpose is to increase the integrity of exchanged data and thus reduce the risks of attacks or failures. We employ the UML-based Specification Environment (USE) and its OCL support to validate specified security properties for a typical container shipping scenario.

KEYWORDS Access Control Model, Permissions, RBAC, ABAC, OCL, Port Community Systems

1. Introduction

In this paper, we apply advanced *Role-Based Access Control* (RBAC) concepts (Sandhu et al. 1996) and combine it with *Attribute-Based Access Control* (ABAC) for fine-grained authorizations (Hu et al. 2015) in the context of a whole *port community* rather than a single company. We model basic parts of the port domain, RBAC, and ABAC constraints (a.k.a. "policies") using the *UML-based Specification Environment* (USE) (Gogolla et al. 2007) and its support for the *Object Constraint Language* (OCL) (OMG 2014) to test and validate the specified constraints. The USE tool has been developed in the research group of Martin Gogolla over the last twenty years. The testing and validation steps allow a policy designer to systematically evaluate defined policies and finally improve the integrity of data shared within a port community.

Since around 90 percent of the world's trade is done via container shipping, the relevance of maritime ports communities is evident. Many ports of the global maritime network play vital roles for transshipments along major trading routes and for the regional economy. Quite a few companies related to these port activities must be considered as *critical infrastructures* that need to be observed and secured against failure. Many sea ports run so-called *Port Community Systems* (PCS) in order to support frictionless operations and increase turnover via IT systems. PCS are in operation for the North-Sea ports Rotterdam, Amsterdam, Antwerp, Zeebrugge, Wilhelmshaven, Bremerhaven, Hamburg, and others. These independent PCS also share and exchange common global shipping data among each other as well as with governmental authorities, i.e. local port authorities. Information regarding dangerous goods or waste potentially harmful to the environment is also systematically collected and reported to authorities.

JOT reference format:
Christian Maeder, Karsten Sohr, Rodrigue Wete Nguempnang, Nils Meyer-Larsen, and Rainer Müller. *Modeling and Validating: Role-Based Authorization Policies for a : Port Communication System with UML and OCL*. Journal of Object Technology. Vol. 19, No. 3, 2020. Licensed under Attribution - No Derivatives 4.0 International (CC BY-ND 4.0)
http://dx.doi.org/10.5381/jot.2020.19.3.a8

The *International Port Community System Association* (IPCSA[1]) 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. Inland ports usually do not have a PCS, which means that corresponding individual stakeholders communicate directly with each other.

Due to the ever increasing importance and complexity of such PCS the objective of two German research projects—as acknowledged at the end—was to deeper examine the potential threats for such software and IT systems. Proper risk assessment for the business processes as well as an overall security architecture for the many stakeholders of a port community are deemed to be essential (Meyer-Larsen & Müller 2018).

Port processes, such as import or export, are often business critical. If they are disrupted, this may even jeopardize the supply for the population. Hence, adequate measures must be implemented to secure digital port processes (Meyer-Larsen et al. 2019). The *European Union Agency for Cybersecurity* (ENISA) (Drougkas et al. 2019) mention the cyberattack in Antwerp, the NotPetya incident impacting Maersk, and other ransomware attacks in the ports of Barcelona and San Diego. Among many security aspects for port processes, i.e. network security to increase availability and reliable authentication possibly supported by a *Public Key Infrastructure* (PKI), access control mechanisms are important for confidentiality and integrity.

Access control has already been adopted by many application areas, such as banking and healthcare (Anderson 2008). Role-based policies as standardized in ANSI (2012) are only well-suited for role hierarchies and simple static or dynamic *Separation of Duty* (SoD) constraints (Simon & Zurko 1997; Ahn & Sandhu 2000). Fine-grained access control that considers i.e. a history (Bertino et al. 2001) or a context (Georgiadis et al. 2001) requires substantial extensions that should be better perceived as being attribute-based. For specifying both, RBAC and ABAC constraints, we employ OCL as unifying formal language. While appreciating the support of OCL via the USE tool, a more adequate unified specification language for RBAC and ABAC seems still to be missing.

The remainder of this paper is organized as follows. After a discussion of related work in Section 2, we present the background of this paper in Section 3. Thereafter, we introduce our approach to modeling and validating exemplary role-based and attribute-based policies in the port domain in Section 4. Section 5 concludes with a summary and an outlook.

2. Related Work

A plethora of works exist that integrate security policies into software and system models based on UML. These works include Kuhlmann et al. (2011, 2013); Sohr et al. (2012, 2008); Jürjens (2002); Ray et al. (2004); Ahn & Shin (2001); Fernández-Medina & Piattini (2004); Basin et al. (2006); Alam et al. (2008); Strembeck & Mendling (2011); Basin et al. (2011). Some of the approaches do not particularly address RBAC like UMLsec (Jürjens 2002). Basin et al. (2006) 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 discuss SoD constraints. In Simon & Zurko (1997); Gligor et al. (1998); Sohr et al. (2008), we explicitly model role-based SoD constraints with UML and OCL. A similar model-driven approach for role-based policies is introduced by Alam et al. (2008) within the frameworks for applications based on web services.

Several works on the validation of role-based policies based on UML and OCL exist (Basin et al. 2009; Yu et al. 2008; Sohr et al. 2008). Based upon SecureUML, Basin et al. (2009) propose an approach to analyze role-based 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?". Our approach allows similar kinds of queries through the query facility of the USE tool (Gogolla et al. 2007). In Yu et al. (2008), 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 cooperation with Martin Gogolla, we have published several works which are similar to the aforementioned papers dealing with UML/OCL modeling in the context of RBAC and its extensions (Kuhlmann et al. 2011, 2013; Sohr et al. 2012, 2008). Not only do these publications focus on the modeling part but also on the analysis of role-based policies by means of a validation approach that employs the USE tool. Furthermore, they support dynamic authorization constraints, such as object-based dynamic SoD and history-based SoD (Simon & Zurko 1997; Gligor et al. 1998)—an aspect that is not in the focus of the other modeling approaches.

This work now applies the concepts of our earlier papers in the context of a port-based scenario. This scenario has been developed with stakeholders of seaports, such as the vessel operator Hapag-Lloyd, a logistician BLG LOGISTICS as well as a PCS vendor/operator.

3. Background

We first describe a PCS, the core of the IT infrastructure of many seaports, and then we discuss RBAC and its extensions. We conclude this Section with explaining relevant concepts of UML, OCL, and the USE tool.

3.1. Port Community Systems

A PCS supports the processes to export and import goods. In Figure 1 we describe the export as done in the German sea ports of Hamburg, Wilhelmshaven, or Bremerhaven. Only essential electronic messages are shown and accompanying physical activities are omitted. The following roles are typical for port processes.

[1] https://ipcsa.international

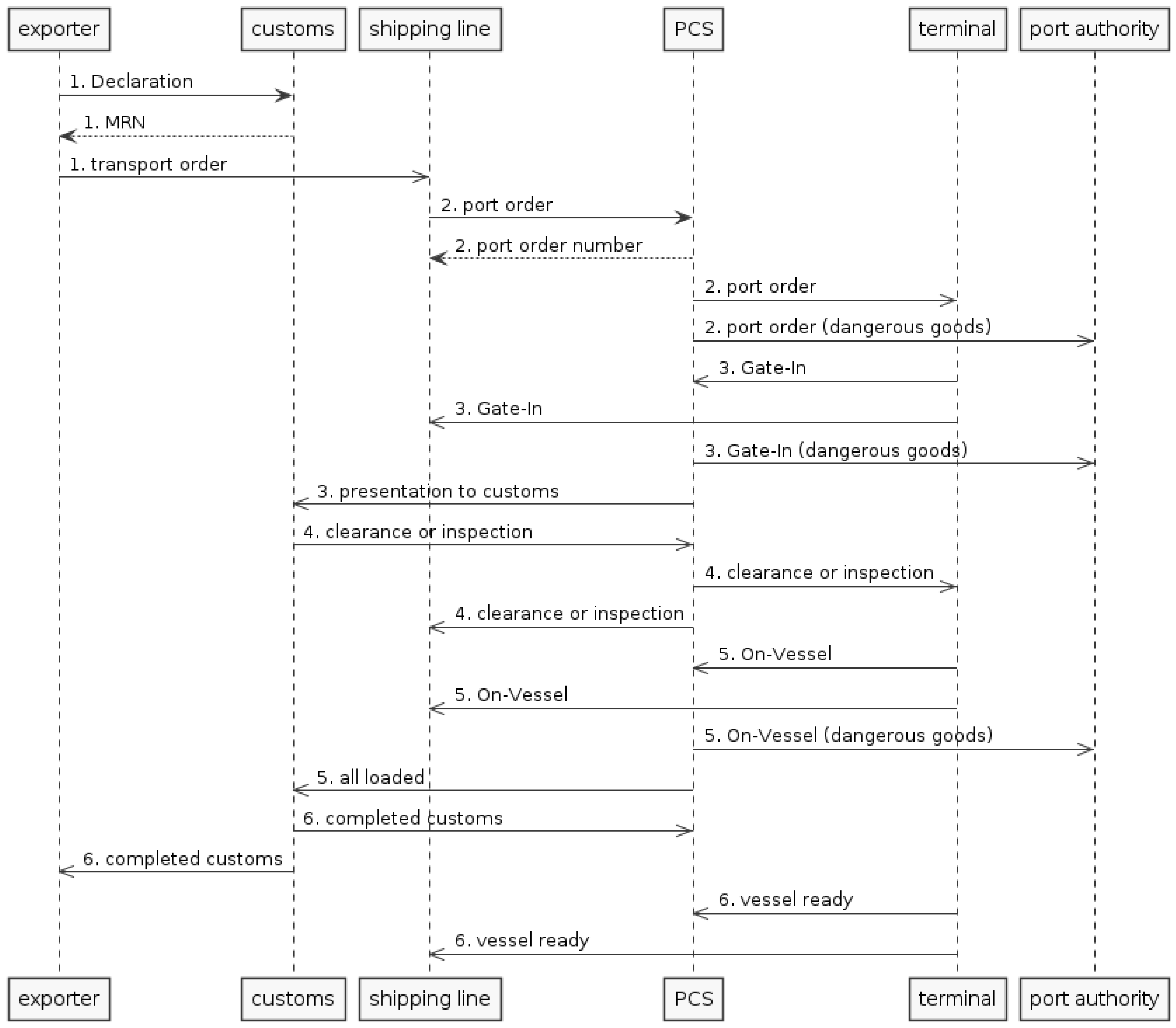

Figure 1 Export process (numbers refer to descriptions below)

– Exporters, traders, shippers, or importers actually want to export, import, or simply transfer containerized goods via the port. There are also inland carriers (or forwarders) that transport goods to or from the ports but we comprise all these players using a single exporter role in the sequence diagram.
– Customs has ultimate control over the goods being imported or exported that exporters have to declare in advance. Customs duties may be charged. Goods are held in customs areas—usually terminals at ports—until being cleared possibly after some containers have been inspected.
– Shipping lines offer the transport capabilities. Exporters ask shipping lines or agents to transport (their) goods by ship. The shipping line organizes the subsequent container handling, i.e. by sending a port order to a PCS in charge. For ports without a PCS shipping lines communicate with terminals directly.
– The PCS of Figure 1 is basically responsible to coordinate the presentation to customs for containers arriving at terminals. Decisions of customs as well as other information, i.e. about dangerous goods, is forwarded to other involved players.
– Terminal operators administer storage locations at the port and control required clearances before goods are actually moved. Terminals closely communicate with shipping lines for loading and unloading of containers.
– A port authority is informed about all dangerous goods arriving, leaving, or located at the port to ensure timely and appropriate responses in case of hazards.

The major steps made by individual players of the aforementioned roles are numbered as in the sequence diagram of Figure 1 and described as follows.

1. The first step for an exporter is to declare the goods to be exported at customs. Customs returns shipping documents and a so-called *Movement Reference Number* (MRN). The exporter hands over this information to a shipping line.

2. The shipping line (or an agent) organizes the transport of the exporter's goods via forwarders to a terminal and sends an official *port order* to the PCS. Apart from the MRN, the port order contains the actual packing of containers and the concrete voyage comprising the terminal for loading and the destination port. The PCS returns a port order number to the shipping line and forwards the port order to the chosen terminal. A further copy of the port order is sent to the responsible port authority in case *dangerous goods* are involved.

3. The terminal associates arriving containers with port orders and reports this back to the PCS and to the shipping line as *Gate-In* messages. The PCS forwards *Gate-In* messages for containers with dangerous goods to the port authority. The PCS gets declaration data from customs and requests customs clearance as soon as all goods associated with one MRN are located at the terminal. This is called *qualified presentation to customs*.

4. Customs can now grant clearance to containers or order various kinds of inspections. Their decision is sent to the PCS that in turn informs the terminal and shipping line. Only after customs clearance, containers may be loaded. Furthermore, all containers need a *Verified Gross Mass* (VGM). The shipping line is responsible for obtaining a VGM from the exporter but the exporter can also supply a VGM via (a service of) the PCS to the shipping line. The terminal and the shipping line together agree on a bay- or stowage plan for the vessel.

5. The terminal reports the loading of a container to the PCS and the corresponding shipping line. In case of dangerous goods, the PCS forwards this message to the port authority. When all containers associated with one MRN are loaded, the PCS informs customs.

6. Customs acknowledges completion of the process to the exporter and the PCS. Finally the terminal informs the PCS and the shipping line when vessel loading has completed.

3.2. RBAC and Authorization Constraints

RBAC is a well-established access control model, by which users obtain permissions not directly but via roles that users play in their organization. RBAC has been formalized by Sandhu et al. (1996), which in turn was later used as the basis for the ANSI (2012) RBAC standard. The following concepts are defined in Sandhu et al. (1996):

– the sets U, R, P, S (users, roles, permissions, and sessions, respectively)
– $\mathsf{UA} \subseteq \mathsf{U} \times \mathsf{R}$ (user assignment)
– $\mathsf{PA} \subseteq \mathsf{R} \times \mathsf{P}$ (permission assignment)
– $\mathsf{RH} \subseteq \mathsf{R} \times \mathsf{R}$ is a partial order called the role hierarchy or role dominance relation written as $\preceq$.

Users may *activate* a subset of the roles they are *authorized* for in a *session*. P is a set of ordered pairs of operations and objects. In the context of access control all *resources* accessible in an IT-system (e.g. files, database tables) are referred to by the notion *object*. An *operation* is an action on objects (e.g. read, update, delete). The relation PA assigns roles from R to permissions from P. So PA determines for a given role the operation(s) it may execute and the object(s) to which the operation in question is applicable. Thus a user with an activated role of a session can apply an operation to an object if the corresponding ordered pair is an element of the permission assignment PA. Role hierarchies can be formed by the RH relation. Users of senior roles inherit permissions from junior roles through the RH relation (e.g., the role chief_physician inherits all permissions from the physician role, written as: physician $\preceq$ chief_physician). A user from U is *authorized* for a role if the user is directly *assigned* to the role *or to a senior role* by the user assignment relation UA.

Figure 2 shows a basic class diagram for the above sets and relations, where the role hierarchy that may be optional is modularly given by a subclass HR of R for *hierarchical roles* with inverse senior and junior relations. The structure of permissions

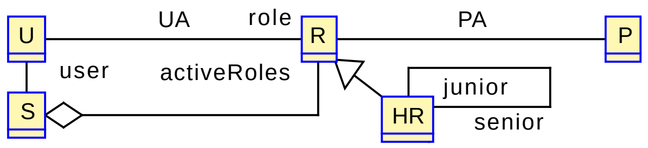

Figure 2 RBAC class diagram

P with operations and objects[2] is omitted. A session from S has a unique associated user from U and aggregate roles from R as *activated roles*.

Figure 3 shows an example of an object diagram based on the RBAC class diagram (Figure 2) that reflects the above sequence diagram (Figure 1). The top row shows all role objects as in the sequence diagram, all other nodes are permission objects. The partly overlapping connections—only between roles and permissions and not between permissions—show the role permission assignment PA. A PCS shares some permissions with other players, basically forwarding information. In order to keep the picture readable, receiving information is not modeled as permission. Therefore no permission is assigned to port authorities, although in a realistic setting, reading sensitive data like dangerous goods should be modeled as a permission as well.

Authorization constraints are an important advanced concept of RBAC that are basically given by restrictions of the above RBAC functions and relations. For example, an SoD constraint may state that a user must not be authorized for both a cashier and a (non-senior) cashier_supervisor role, i.e. the UA relation is *statically* restricted. Less strict would be a *dynamic* SoD constraint, where the two roles merely must not be *active* in simultaneous *sessions* of a user.

Sandhu et al. (1996) introduce a family of RBAC models for the various dimensions. $RBAC_0$ is the core model and $RBAC_1$ adds role hierarchies. $RBAC_2$ adds constraints to $RBAC_0$ and is incomparable to $RBAC_1$. The consolidated model is $RBAC_3$ as union of $RBAC_1$ and $RBAC_2$. Ahn & Sandhu (2000) propose the Role-based Constraint Language (RCL) to specify constraints for $RBAC_2$ and $RBAC_3$. RCL is a domain specific language, based on a restricted first-order logic, that allows to express constraints in a declarative way without explicit quantifiers as we explain in Section 4.4.1.

It has been argued elsewhere that authorization constraints are the principal motivation behind the introduction of RBAC (Sandhu et al. 1996). They allow a policy designer to express higher-level organizational rules. In the literature, several kinds of authorization constraints have been identified, such as various types of static and dynamic SoD constraints (Gligor et al. 1998; Simon & Zurko 1997; Ahn & Sandhu 2000), prerequisite roles and cardinality constraints (Sandhu et al. 1996), constraints on delegation (Zhang et al. 2003), context constraints (Joshi et al. 2005; Georgiadis et al. 2001), workflow constraints (Bertino et al. 1999), binding of duty (Brucker et al. 2012). The RBAC standard (ANSI 2012) recommends specific SoD relations based on conflicting roles and cardinalities. Only sessions are dynamic in RBAC. Dynamic constraints based on i.e. attributes, types, contexts like the time of day, history, etc., go clearly beyond the standard and $RBAC_3$.

For advanced constraints, attribute-based access control (ABAC) has been introduced (Hu et al. 2015). This access control model unifies access control concepts, such as roles, contexts, or access history, into a common model and represents these access control entities as attributes, which are used within access control decisions. ABAC is the recommended access control model for promoting information sharing between diverse and disparate organizations (Rubio-Medrano et al. 2013; Hu et al. 2014).

As RBAC, however, is still one of the most prevalent access control models in practice, we focus on it as long as possible. We switch to ABAC for context constraints (Georgiadis et al. 2001), which are needed within the frameworks of port processes, e.g. import and export. A typical context constraint is so-called *multitenancy*, meaning that users (i.e. customers) despite having the same basic role are only allowed to access resources they *own* (or created) as we elaborate in Section 4.4.2.

3.3. UML, OCL, and USE

In the following, we briefly introduce UML, OCL, and the USE tool. For this purpose, we partly resort to our earlier publications (Hamann et al. 2015; Sohr et al. 2012).

The Unified Modeling Language (UML) represents a general-purpose visual modeling language, in which one can specify, visualize, and document the components of software systems. It captures decisions and understanding about systems that are to be constructed. UML has become a standard modeling language in the field of software engineering.

Through different views and corresponding diagrams, UML permits the description of static, functional, and dynamic models. In this paper, we concentrate on UML class and object diagrams as well as on sequence diagrams for documentation purposes of business processes. A class diagram provides a structural view of a system. Classes are defined in terms of their attributes and relationships. Relationships are binary or n-ary associations between classes but they can also be association classes with additional attributes. Object diagrams visualize instances of the modeled system, i.e. class instances (objects), attribute instances (values), and instances of associations (links). Figures 2 and 3 show an example class and object diagram. A sequence diagram is depicted in Figure 1 visualizing an export process via a PCS.

The Object Constraint Language (OCL) is a declarative textual language that describes constraints on object-oriented models (Warmer & Kleppe 2003; OMG 2014). It is an industrial standard for object-oriented analysis and design.

OCL expressions consist of standard operations or user-defined query operations. The built-in standard operations support calculations on the basic types Boolean (e.g. *and*, *or*, *implies*), Integer (e.g. +, $*$, *mod*), Real (e.g. /, *round*), String (e.g. +) as well as on collection types, i.e. sets, bags (multisets),

[2] In order to avoid confusion with the term *object*, i.e. in the context of object diagrams, we henceforth prefer the term *resource* in conjunction with RBAC permissions. In the literature on access control, however, still the term *object* is common.

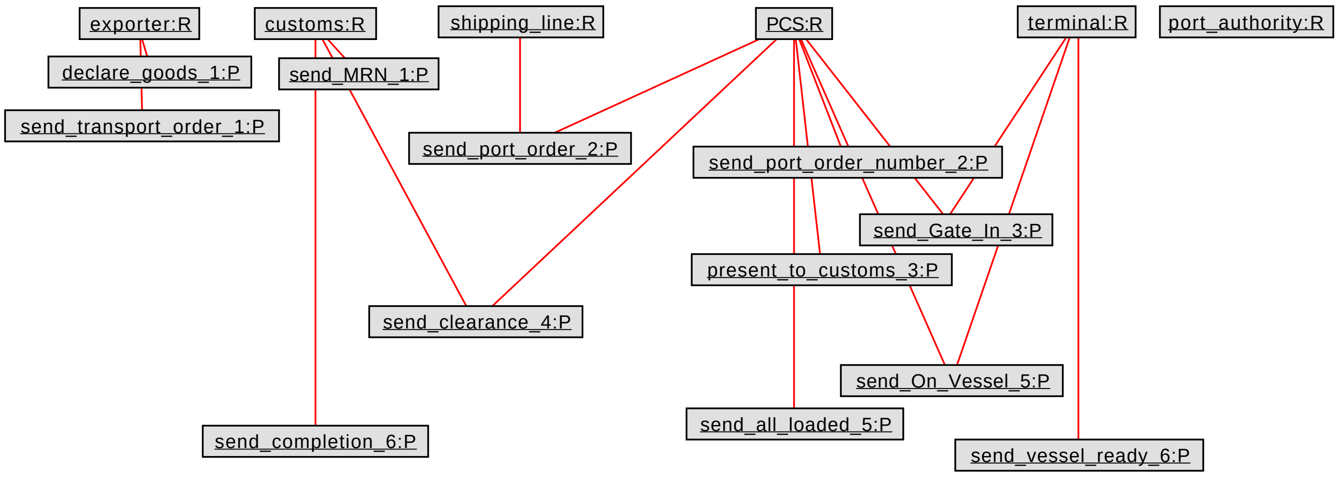

Figure 3 Object diagram for PCS roles and permissions

```
context S inv activ:
  user.role.getRoles()→includesAll(activeRoles)
```

Listing 1 Check active roles of a session

ordered sets, and sequences. Beside the usual collection type operations (e.g. *union*, *size*, and *includes*) several operations enable iteration over the members of a collection, such as *forAll*, *exists*, *collect*, *select*, and *reject*. These latter operations can all be expressed by the very generic *iterate*[3] operation that takes an additional accumulator argument. The most important features of OCL are navigation and attribute access, which connect an OCL expression with the values in a concrete model instance. By definition, OCL constraints can restrict aspects of a UML model through invariants.

The OCL invariant in Listing 1 checks if all *activated* roles of a session are included in the *authorized* roles of the session's *user*. The operation getRoles (cf. Listing 2) computes all potential junior roles of a role including itself. The term user.role selects the directly *assigned* roles from the UA relation.

The UML-based Specification Environment (USE) supports modeling features and their analysis through validation and verification (Gogolla et al. 2007). Within USE, UML class, object, statechart, sequence, and communication diagrams extended with OCL are available. USE assists the developer in validating and verifying model characteristics. USE features a model validator based on relational logic and SMT solvers. Model properties to be inspected include consistency, redundancy freeness, checking consequences from stated constraints, and reachability. These properties are handled on the conceptual modeling level, not on an implementation level. Employing these instruments, central and crucial model characteristics can be efficiently verified.

[3] From functional languages the iterate operation is known as a function reduce or foldl (HASKELL) for folding from left to right. Reduction or folding also allows one to express map and filter functions. OCL's collect operation corresponds to mapping, whereas select and reject is filtering.

```
class R operations
  getRoles():Set(R) = Set{self} end
class HR < R operations
  getRoles():Set(R) = juniors()→including(self)
  juniors():Set(R) = junior→closure(junior)
constraints inv acyclic:
  juniors()→excludes(self) end
association RH between
  HR[*] role senior
  HR[*] role junior end
```

Listing 2 Specification of hierarchical roles

USE expects a model specification with constraints as text. The classes R and HR as well as the RH association from the class diagram in Figure 2 for roles, hierarchical roles, and the role hierarchy are shown in Listing 2.

The subclass HR of R overwrites the operation getRoles from the superclass to include all junior roles. The auxiliary operation juniors collects all proper junior roles using OCL's *closure* operation. The class HR also contains a constraint to ensure that role hierarchies are acyclic. It is a matter of taste to specify constraints as part of a class, i.e. HR, or within a separate *context* as done for the invariant in Listing 1 that is specified in the *context* of the session class S. Having separate classes HR and R at all was a design decision made to keep the basic super class R as simple as possible. Yet, the class R has the operation getRoles that yields a somewhat artificial singleton set of it *self* for roles without a hierarchy but such an operation is indispensable for an extension like a role hierarchy.

Given a class diagram, i.e. Figure 2, created from a textual specification in a text file (with extension .use) as partly shown in Listing 2, USE supports several ways to construct corresponding object diagrams. The primary way is to construct an object diagram with objects, attributes, and links of associations, interactively. The objects, attributes, and links created can then be saved to a script file. Additionally, the layout of object dia-

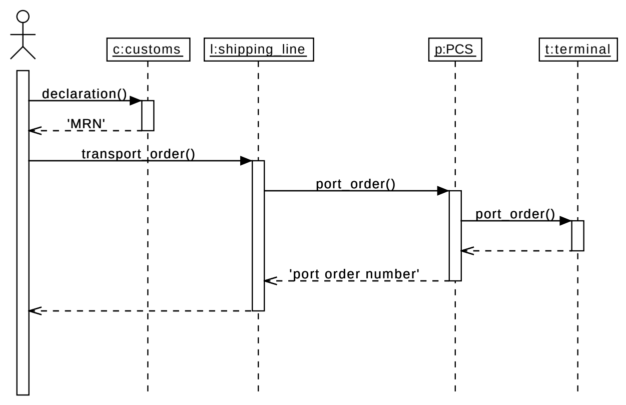

Figure 4 Sequence diagram based on SOIL

grams created manually or via layout algorithms can be saved separately similar to those of class diagrams.

Another way to create object diagrams works through the integrated model validator. The search space of the validator can be configured via a `.properties` file. For example, minimal and maximal numbers of objects and links for classes and associations can be set. The default configuration assumes exactly one object for every class. However, this initial configuration is unsatisfiable for the acyclic invariant of hierarchical roles HR. At least two different roles are required for a simple role hierarchy RH. The configuration allows to set minimal values to zero but then only partial solutions are generated because invariants of classes without objects will never be checked. Invariants can also be directly omitted from the search space to easy the analysis of inconsistencies caused i.e. by contradicting invariants. The model validator can also extend a partial, yet consistent object diagram that have been created earlier—possibly manually. Eventually one wants to obtain object diagrams—manually or through suitable configurations—that best reflect the reality.

USE also allows one to imperatively program class operations using the *Simple OCL-based Imperative Language* (SOIL) described in Büttner & Gogolla (2011). The sequence diagrams created from such SOIL programs show the participating objects and the operations invoked on such objects as incoming and returning arrows. In Figure 4 the actor on the left declares goods to customs and makes a transport order to a shipping line. Method calls with return arrows are shown. Such a concrete sequence diagram is different from the sequence diagram[4] depicted in Figure 1 for *modeling* purposes. Modeling is more abstract than programming, i.e. roles instead of concrete players are addressed.

4. Approach

The starting point of our modeling approach are sequence diagrams like the one of Figure 1 that was the result of several discussions with actual stakeholders of the port community. Helpful are also business process models or other informal or

[4] created using PlantUML https://plantuml.com/

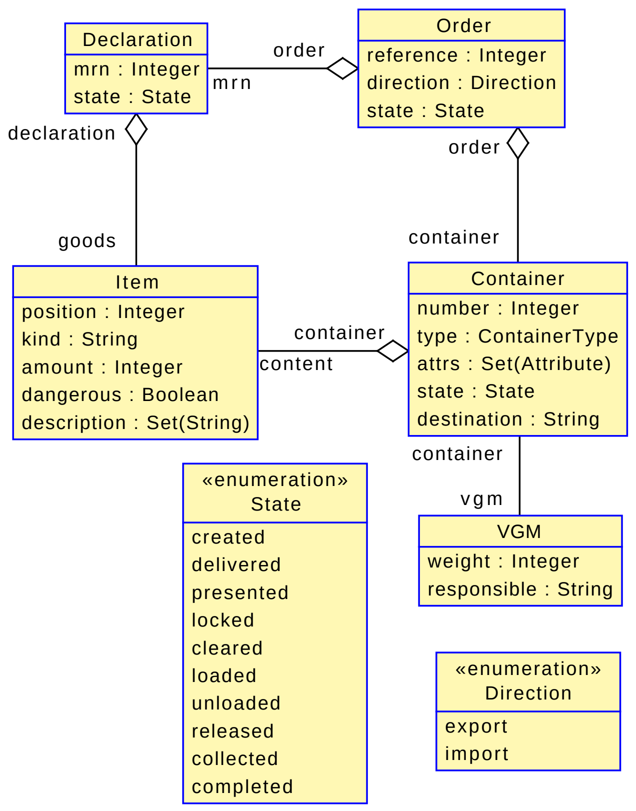

Figure 5 Class diagram for container and port order goods

semi-formal process descriptions. Sequence diagrams depict the flow of information between participants.

The first task is to find a useful set of roles that participants may adopt. Considering the numerous companies and employees of these companies within a port community it is non-trivial to come up with a manageable number of roles. Typically, companies are subdivided and have their own role models. Furthermore, many activities are delegated to specialized agents, counselors, secretaries, charterers, etc. that act on behalf of a client with complicated legal consequences regarding responsibilities. Our naturally subjective choice was to restrict the roles to those described in Section 3.3.1.

The second task is to identify the important processing steps. Eventually we aim for a useful initial sequence diagram like Figure 1. Only after consolidation we create straightforward object diagrams showing the associations between roles and processing steps as permissions, like in Figure 3. As we look at IT systems and not physical actions, all processing steps create, send, receive, read, aggregate, extract, update, or delete data. Therefore we focus on the *data* being processed. The data model for our container export (or import) scenario is presented as a class diagram in Figure 5.

Goods to be exported have to be declared item by item. Each item has an internationally standardized name and number according to the *Harmonized System* (HS) of tariff nomenclature maintained by the *World Customs Organization* (WCO). A single declaration usually comprises many items stating *what* goods should be exported.

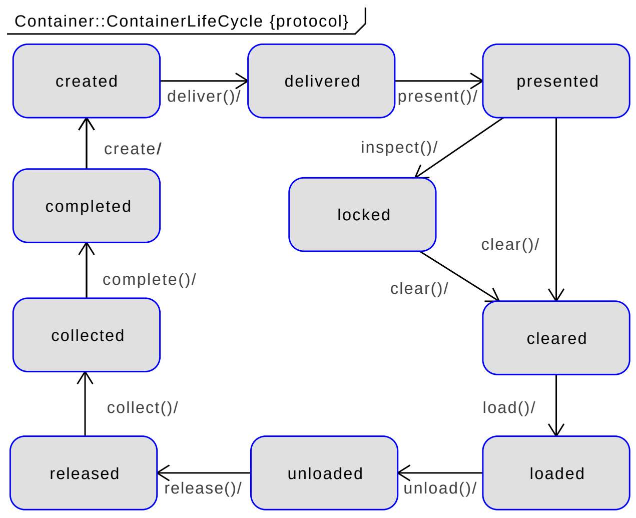

Figure 6 Container life cycle statechart

The same individual items need to be put into containers for transport. We assume that every item fits into a single container, otherwise an item needs to be split into several items, each with a lower amount. Containers determine *how* goods should be transported.

Port orders join two views: customs want to know *what* and shipping lines *how* to export. Items (for the same destination) from several declarations may be inside a single container and all items of a single declaration may have been packaged into several containers.

The attributes of the classes Item, Declaration, Container, and (port) Order are only rough indications, i.e. in reality dangerous goods would be far more elaborated. It is also a matter of style or taste if attributes are kept inside a class or if associations for non-primitive attributes are used.

For instance, the verified gross mass (VGM) of a container is not a primitive attribute but an association to an extra class. This weight of a container became mandatory in 2016 to increase safety on sea, because wrong weights have sometimes caused naval accidents. A VGM also comprises a person that is responsible for the accuracy of the given weight. There are only two legal ways to determine a VGM. One can either weigh a container or sum up the weights of all items inside plus the weight of the empty container. Containers without a VGM must not be put on board of a sea ship.

Customs declarations, port orders, and containers all have a *state* attribute, accidentally sharing an enumeration class. This—mostly implicit—attribute indicates the current processing state during a life cycle, where a life cycle of a container as shown in Figure 6 begins when it is ordered for transport from an empty container depot (#created[5]), and ends when the container is emptied again after the voyage (#completed).

The states of declarations, port orders, and containers are interdependent. Listing 3 shows an invariant for a declaration d that has been fully presented to customs. This *qualified* pre-

```
context d:Declaration inv qualified:
  d.state = #presented implies
  d.order→collect(container)
    →select(c | c.state = #delivered)
      →collect(content→includesAll(d.goods))
```

Listing 3 Qualified presentation to customs

sentation is done by the PCS as soon as all declared goods (d.goods) are *inside*—i.e. content of—a container c at the terminal. Containers at the terminal are in the state #delivered and the PCS is informed about this fact via *Gate-In* messages. Any container c must also be known via a port order that references the declaration (where d.order denotes the reverse reference).

The data described above are exactly those *resources* that are the second components of permissions P from the RBAC model as described in Section 3.3.2. The first components of permissions P are *operations* and as generic operations one initially only considers create, read, update, and possibly delete. The combinations of operations and resources are permissions. The name of a permission is simply the operation name connected to a resource name via an underscore.

As third task we are able to draw a first picture of access control. One simple way is to use an access control matrix as laid out in Table 1.

The first column lists the roles, whereas the first row lists the resources being accessed. As access within the matrix only create and read operations are distinguished. We assume that data is never (re-)written or deleted after creation. All presented data is created by the exporter. Customs may know/read everything about the goods to be exported but do not care about container details. The shipping line and the terminal only need to roughly know the content of containers but care about dangerous goods, a VGM, and container attributes, i.e. for power supply of cooling containers. The PCS has the additional permission to create a VGM by summing up container content. The port authority is only interested in information about dangerous goods at least as long as no accidents occur. Access rights of authorities will surely increase in cases of emergency but we restrict ourselves to model *normal operation*.

Other interesting roles may be those of attackers or insiders. Any attacker can read the information that is publicly available. For instance, anyone could observe physical containers being transported on road or rail near terminals. So an attacker could collect some container numbers and dangerous good indicators and enter container numbers into some container tracking services. However, since this attack is manual and somewhat tedious, we do not consider it further. Insiders would play the roles of their companies.

Generally, the data presented above and traditionally exchanged as EDIFACT[6] is not supposed to be available to others outside the port community.

An access control matrix can be presented differently, i.e. column-wise as access control list (ACL) or row-wise as capabilities (Anderson 2008). The readability of an equivalent

[5] Values of enumerations in USE are preceded by a # sign. The standard OCL notation State::created is legal, too.

[6] *Electronic Data Interchange For Administration, Commerce, and Transport*

data role	value/description	destination/kind/amount	dangerous goods	container attributes	VGM
exporter	create	create	create	create	create
customs	read	read	read		
shipping line		read	read	read	read
PCS		read	read	read	create
terminal		read	read	read	read
port authority			read		

Table 1 Access control matrix

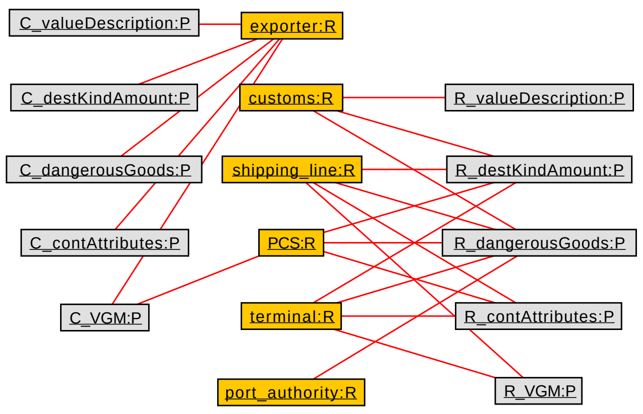

Figure 7 Object diagram for access control matrix

presentation as object diagram like in Figure 7 with marked roles is questionable but large tables are also problematic. For our container export scenario, the given access control matrix is still too simplistic and merely serves as overview.

As fourth task we refine our access control model using *authorization constraints*. We first enumerate the authorization constraints for a port community informally. Our analysis of the port processes revealed the following constraints.

- The customs' job is to control goods. Thus to avoid a conflict of interest we demand a separation of duty constraint. Since only customs is allowed to clear containers, customs must not also play any other role. We formalize this separation of duty in Section 4.4.1.
- The terminal area as well as a vessel allow only physically restricted access. The exact positions of containers at the terminal or on board is supposed to be as secret as possible not only for the public but in particular also for the initiating exporters in order to impede physical manipulations before or after custom clearance. Terminals only report container positions to the PCS and also customs can query container positions for inspection purposes.
- Exporters only create their own data and cannot read or see any data created by other exporters. While shipping lines and terminals closely communicate for stowage planning of concrete vessels, they are supposed to only read the data of their clients and no data of competing shipping lines or terminals. From the perspective of the port community or a PCS we have multitenancy for exporters, shipping lines, and terminals that we discuss in Section 4.4.2.
- Information about dangerous goods is very important for piling containers at terminals or on board of a vessel. Despite hiding exact container positions, containers have dangerous goods indicators that may be publicly spotted.

The last constraint is somehow contradictory and hard to formalize. Containers with dangerous goods need to be recognizable for cases of (traffic) accidents but i.e. a summary of all dangerous goods at a terminal or of a vessel should definitely not be available to terrorists.

Container positions could be made a simple attribute of the Container class in Figure 5 but a separate association would maybe better facilitate the extra protection required. Since we do not further discuss these positions they are also omitted from our access control matrix in Table 1.

4.1. Separation of Duty

Simple separation of duty constraints are easily modeled by defining *conflicting roles*. From any set of conflicting roles at most one role may be authorized for any user. In Listing 4 the class CR represents a set of conflicts via an association ExclusiveRoles to at least two distinct roles. The terms *exclusive* and *conflicting* roles are used interchangeably. The invariant noConflicts ensures that authorized roles of every user (u.role .getRoles()) contain at most one role from the set of conflicts . Using the model validator we can check our specified constraints at any stage, in particular before trying to construct an object diagram manually. If the (properly configured) validator does not find a solution, the specification might be (or definitely is) inconsistent, i.e. no object diagrams, with all constraints verified, exist.

For customs we need to define the following sets of conflicting roles. From all elements of CR in Listing 5 at most one from the two roles may be taken by any player. A representation of these sets as object diagram is shown in Figure 8. The mere existence of this object diagram proves the consistency of our

```
class CR constraints inv noConflicts:
  U.allInstances→forAll(u | u.role.getRoles()
    →intersection(conflicts)→size ≤ 1) end
association ExclusiveRoles between
  CR[*]
  R[2..*] role conflicts end
```

Listing 4 Constraint for exclusive roles

```
CR = {{customs, exporter},
      {customs, shipping_line},
      {customs, PCS},
      {customs, terminal},
      {customs, port_authority}}
```

Listing 5 Set of sets of conflicting roles

model if all constraints are true. It remains to discuss if the model is appropriate.

With the set of sets CR of Listing 5 an alternative formalization can be given via Ahn's role-based constraint language RCL 2000 (Ahn & Sandhu 2000).

```
|roles*(OE(U)) ∩ OE(CR)| ≤ 1
```

The authorized roles of any user *OE*(U) may at most contain a single role from any conflict set *OE*(CR). *OE* denotes *one element* of the argument set and since this may be any chosen element, the statement must be valid for all elements.

Combining the role customs with all others roles bears the problem that additions or deletions of roles also causes additions and deletions of conflicting role sets. This can be overcome by different or additional class definitions. An alternative class without the need to create combinations but only a proper subclass instance, i.e. for customs, is displayed in Listing 6. This constraint has the benefit that it is easy to add further unique roles, for instance for port authorities, since they also play a supervisory role, i.e. with respect to dangerous goods.

An alternative formalization using the following RCL statement would ensure that one element of the user set can only have a single role, if this user has a role from the set UR where UR would contain the role customs as an element.

```
OE(UR) ∈ roles*(OE(U)) ⇒ |roles*(OE(U))| = 1
```

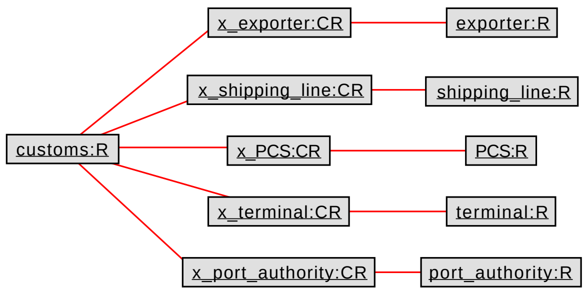

Figure 8 Diagram of exclusive roles

```
class UR < R constraints inv unique:
  U.allInstances→forAll(u | u.role.getRoles()
    →size ≤ 1) end
```

Listing 6 Unique exclusive role

```
inv i: UR()→forAll(r | U()
  →forAll(u | uroles(Set{u})→includes(r)
    implies uroles(Set{u})→size = 1))
```

Listing 7 RCL statement as OCL invariant

The function *roles** computes all authorized roles of a user that includes junior roles introduced by a role hierarchy. For such RCL statements we have developed a translator to USE that has not been published yet. The corresponding OCL invariant for the above RCL statement is given in Listing 7 where the term uroles(*Set*{u}) matches u.role.getRoles() from Listings 6 and 4. In our extension of RCL the duplicated term *roles**(*OE*(U)) could be avoided via a *let-construct* as could be done in the OCL translation for the duplicated term uroles(*Set*{u}). The operations uroles (for *user roles*), UR(), and U() are extra operations (not shown here) that return sets where U() is identical to U. *allInstances*. The premise before *implies* replaces the subclass specification of UR in Listing 6. The order of the quantifications using *forAll* is irrelevant and chosen in the order of appearance of *OE* terms within the RCL statement. In the end both OCL formalizations in Listings 6 and 7 are equivalent.

The given constraints for unique roles must not be mixed up with cardinality constraints as described in Berger et al. (2019). A cardinality constraint restricts the *number of users* who may be authorized for a role. It could be argued that there should be only a single player[7] for the customs role. Yet, this user should also not take any other role, which is only ensured by the above role exclusions and independent from cardinality. (The RCL statement for a single user having the role customs would be: |*user*(customs)| = 1). Without the cardinality constraint we allow several employees working as customs, exclusively.

4.2. Multitenancy

The RBAC model as described so far is too static to model fine-grained access for users having the same roles. Subdividing roles, i.e. separate exporter roles for different terminals or ports, may lead to *role explosion* (Elliott & Knight 2010). If we look at our data, we omitted the fact that customs declarations, for instance, are made by exporters and port orders by shipping lines. This connection is not even easily modeled using UML, because we only have a class of users U and no separate classes for exporters and shipping lines. By joining the class diagrams from Figure 2 and 5 we could only add associations with U (or attributes with values from U).

Via the declaration an exporter is also indirectly associated to all items of this declaration. A port order is associated to a concrete voyage of a vessel of a shipping line. Associated to the

[7] A single user of a customs role may be the German ATLAS system.

voyage is a terminal at a port where goods shall be loaded and unloaded. These associations carry over to all containers of a port order. Container items are matched against declared items, thus one or several exporters are associated to containers. The visibility or accessibility should be as follows:

- An exporter may see only her own items in a container, and maybe the fact that these are not all items of this container. She may also know the container number, the state, and some more attributes of such a container.
- Via the container number the voyage, vessel, shipping line, and terminal may be looked up.
- A shipping line only sees containers destined for its vessels, contact information of exporters (or importers) with goods in these containers, and contact information of individuals responsible for VGMs. Vessels know the terminals they call at.
- A terminal basically sees information about vessels and containers that are going to be loaded or unloaded. These containers are known in advance for planning purposes.

The current state of the art for such fine-grained or dynamic permissions is *attribute-based access control* (ABAC) (Hu et al. 2015). As the name suggests, attributes of users, resources, or even global data determine whether the access of a user to a resource via an operation is granted or denied. A formalism and key standard that implements ABAC is the XML-based *eXtensible Access Control Markup Language* (XACML) (Rissanen 2017). However, the major implementations[8] are extensive. Unfortunately, a profile of XACML called ALFA (Giambiagi et al. 2015) for *Abbreviated Language for Authorization* with a text-based lightweight syntax seems to be non-free[9] and the state of standardization via the *Organization for the Advancement of Structured Information Standards* (OASIS) since 2015 is unknown (Giambiagi et al. 2015) whereas a recent JSON[10] profile of XACML 3.0 (Brossard & Legg 2019) exists but replacing XML by JSON does not provide the readability promised by ALFA.

ABAC alone is expressive enough to discard RBAC entirely. Any role can be viewed as an attribute of a user or operation and some recommend to transition away from RBAC to ABAC (Fatima et al. 2016). However, since RBAC is so widespread and comprehensible, the current practice should be to combine RBAC and ABAC. OASIS acknowledged RBAC support for XACML back in 2014 (Rissanen 2014) and the access management system Apache Fortress^{TM} based on RBAC advertised ABAC support (McKinney 2019) in a release note 2019[11]. Since we merely concentrate on authorization and not on web-based authentication, we prefer to continue to use OCL to express additional ABAC (or context) constraints as described by Berger et al. (2019). The idea is to have additional associations between users (from U) and resources (as parts of permissions from P) or to have subclasses of users and resources with additional operations that allow to retrieve those attributes—if not

[8] AuthZForce https://authzforce.ow2.org/

[9] Axiomatics https://www.axiomatics.com

[10] *JavaScript Object Notation*

[11] Fortress Feature Content 269 https://issues.apache.org/jira/browse/FC-269

```
abstract class Constraint operations
  isAllowed(u:U, r:Resource):Boolean end
aggregation Owner between
  U [0..1] role owner
  Resource [*] end
class OwnerConstraint < Constraint operations
  isAllowed(u:U, r:Resource):Boolean =
    u.resource→includes(r)
    or r.oclIsKindOf(Item) and (
      let i = r.oclAsType(Item) in
      u.resource→includes(i.declaration) or
      u.resource→includes(i.container.order))
    or r.oclIsKindOf(Container) and (
      let c = r.oclAsType(Container) in
      u.resource→includes(c.order) or
      c.content→exists( i | u.resource
        →includes(i.declaration))) end
```

Listing 8 Generic and owner constraint

directly given as UML attributes—that should further determine access decisions. The common part of such constraints is a simple Boolean predicate as shown in Listing 8. Given an owner association between a user and resources (as aggregation) we can express the constraint that the user u can only access any of these resources r by u.resource→*includes*(r). The repeated[12] subexpression u.resource denotes all resources the user u owns. Assuming further that the classes Item, Container, Declaration, and Order from Figure 5 are all subclasses of Resource, we can additionally express (using *or*) that:

1. all *items* i
   (a) of a declaration can be accessed by the owner u of the declaration,
   (b) of a container of a port order can be accessed by the owner u of the order;
2. all *containers* c
   (a) of a port order can be accessed by the owner u of this order,
   (b) can be accessed that contain at least one item i of any declaration owned by the accessing user u.

Note that items or containers may also be directly owned or may not be associated to declarations or port orders, yet. The owner multiplicity [0..1] only ensures that at most one owner for any resource can exist.

The operation isAllowed of any constraint's instance is evaluated shortly before accessing a resource but after the static RBAC constraints have been evaluated. Whereas invariants can be validated, the actual access to resources can only be tested. The above disjunction (using *or*) within the owner constraint is still quite restrictive as it does not allow access to any other resources *without owners* that are not items or containers. It

[12] The repetition could be avoided by a let-expression but a telling name would not be shorter.

is also cumbersome to express conditions within the body of isAllowed that address the RBAC roles and permissions of the user and resource arguments.

4.3. Discussion

Modeling the data (cf. Figure 5), i.e. the resources to be accessed, turned out to be a challenge.

- We have classes for items or containers with different access rights for certain sensitive attributes, i.e. the value and detailed description should be only visible for customs.
- We have collections, i.e. of containers with dangerous goods, that should be collectively protected while every container exposes a classification label.
- Most messages exchanged between players merely contain *reference numbers*, i.e. for port orders, declarations / *movement* (MRN), or container numbers. It is quite unclear how sensitive these numbers are, if they are known to malicious actors.
- Finally, the data is associated to various users with separate roles and we have access restrictions like multitenancy.

Within our access control matrix in Table 1, we may notice that the permissions for the roles shipping line, PCS, and terminal are similar.

1. The given permissions for shipping lines and terminals are the same.

2. Compared to shipping lines and terminals, the PCS has the create permission rather than only the read permission for a VGM.

From 1 we might conclude that there should only be a single role for the two roles. From 2 we might conclude that the PCS should be a senior role as creating a date surely subsumes reading it. Because we ignored further resources and permissions of shipping lines, PCS, and terminals in wider or other scenarios, we refrained from joining roles or from introducing a role hierarchy. For some separation of duty (SoD) constraints (RBAC$_2$), a role hierarchy (RBAC$_1$) may interfere (Sandhu et al. 1996; ANSI 2012). If senior and junior roles are defined as exclusive roles, the use of the senior role would establish a conflict. Generally, conflicting roles should have no common senior role.

We could easily model and validate the simple SoD between customs and other roles (cf. Section 4.4.1). The alternative OCL invariants are fairly readable. Yet, a domain specific language like RCL (Ahn & Sandhu 2000) may be a better choice if there is some tool support like a translation from RCL to OCL as indicated by Sohr et al. (2008).

Multitenancy (cf. Section 4.4.2) goes beyond RBAC and a suitable specification language seems to be missing if i.e. ALFA (Giambiagi et al. 2015; Brossard et al. 2017) is not revived. The OCL owner constraint given as definition of an operation is more complex but having the USE tool allows to experiment with object diagrams, evaluate OCL expressions and to run tests.

5. Conclusions

We presented the classical RBAC model from Sandhu et al. (1996) using UML and OCL as done with minor variations in earlier works that often also employed the USE tool. In a larger case study we modeled typical authorization policies of port communities with a PCS. Such systems are used by many sea ports.

Authorizations constraints for RBAC, like separation of duty (SoD) by exclusive roles, could be easily modeled and validated using USE. Although RBAC is a frequently applied standard, it is limited for fine-grained authorization. Adhering to RBAC may lead either to a violation of the *principle of least privilege* by coarse roles or to the acknowledged problem of *role explosion* (Elliott & Knight 2010). A proposed solution was the transition to next-generation ABAC with its own policy language. We suggest to combine RBAC and ABAC; as common language we prefer OCL over XACML also for fine-grained ABAC policies like multitenancy. OCL allows to reuse the well-established RBAC policies and concentrate on fewer complex policies. A more adequate specification language is conceivable but missing. RCL (Ahn & Sandhu 2000) and ALFA (Giambiagi et al. 2015) are biased with respect to RBAC or ABAC and tool support for these languages is unclear.

The formal and consistent specification of policies for a community is only the starting point for the companies to implement enforcements. Such implementations are beyond USE but code can be generated from UML and OCL using the *Eclipse Modeling Framework* (EMF) (Steinberg et al. 2008) that is more heavy-weight. An alternative implementation may target XACML (Rissanen 2017). Summarizing, the light-weight USE tool proved to be effective to obtain formal models for both classical RBAC as well as flexible ABAC authorization constraints. The validation, testing and last but not least visualization functions of USE helped to promote access control as an important part of the overall security of port communities.

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 of Transport and Digital Infrastructure (BMVI) under the grant 19H18012E (SecProPort project). Finally, we thank the anonymous reviewers for their valuable feedback.

References

Ahn, G.-J., & Sandhu, R. (2000). Role-based authorization constraints specification. *ACM Transactions on Information and System Security (TISSEC)*, *3*(4), 207–226. doi: doi:10.1145/382912.382913

Ahn, G.-J., & Shin, M. E. (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* (pp. 157–162). IEEE. doi: doi:10.1109/ENABL.2001.953406

Alam, M., Hafner, M., & Breu, R. (2008). Constraint based role based access control in the SECTET-framework: A model-driven approach. *Journal of Computer Security*, *16*(2), 223–260. Retrieved from https://dl.acm.org/doi/10.5555/1370687.1370692

Anderson, R. (2008). *Security engineering*. John Wiley & Sons. Retrieved from https://www.cl.cam.ac.uk/~rja14/book.html

ANSI. (2012). *American national standard for information technology—role based access control.* New York, NY, USA: American National Standards Institute, Inc. (INCITS 359-2012 (R2017) Revision of INCITS 359-2004 https://profsandhu.com/journals/tissec/ANSI+INCITS+359-2004.pdf)

Basin, D., Clavel, M., Doser, J., & Egea, M. (2009). Automated analysis of security-design models. *Information and Software Technology*, *51*(5), 815–831. doi: doi:10.1016/j.infsof.2008.05.011

Basin, D., Clavel, M., & Egea, M. (2011). A decade of model-driven security. In *Proceedings of the 16th acm symposium on access control models and technologies* (pp. 1–10). New York, NY, USA: ACM. doi: doi:10.1145/1998441.1998443

Basin, D., Doser, J., & Lodderstedt, T. (2006). Model driven security: From UML models to access control infrastructures. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, *15*(1), 39–91. doi: doi:10.1145/1125808.1125810

Berger, B. J., Maeder, C., Wete Nguempnang, R., Sohr, K., & Rubio-Medrano, C. (2019). Towards effective verification of multi-model access control properties. In *Proceedings of the 24th acm symposium on access control models and technologies* (pp. 149–160). New York, NY, USA: ACM. doi: doi:10.1145/3322431.3325105

Bertino, E., Bonatti, P. A., & Ferrari, E. (2001). TRBAC: A temporal role-based access control model. *ACM Transactions on Information and System Security*, *4*(3), 191–233. doi: doi:10.1145/501978.501979

Bertino, E., Ferrari, E., & Atluri, V. (1999). The specification and enforcement of authorization constraints in workflow management systems. *ACM Transactions on Information and System Security*, *2*(1), 65–104. doi: doi:10.1145/300830.300837

Brossard, D., Gebel, G., & Berg, M. (2017). A systematic approach to implementing ABAC. In *Proceedings of the 2nd acm workshop on attribute-based access control* (pp. 53–59). New York, NY, USA: ACM. doi: doi:10.1145/3041048.3041051

Brossard, D., & Legg, S. (2019). *JSON profile of XACML 3.0 version 1.1.* (OASIS Standard https://docs.oasis-open.org/xacml/xacml-json-http/v1.1/xacml-json-http-v1.1.html)

Brucker, A. D., Hang, I., Lückemeyer, G., & Ruparel, R. (2012). SecureBPMN: Modeling and enforcing access control requirements in business processes. In *Proceedings of the 17th acm symposium on access control models and technologies* (pp. 123–126). New York, NY, USA: ACM. doi: doi:10.1145/2295136.2295160

Büttner, F., & Gogolla, M. (2011). Modular embedding of the object constraint language into a programming language. In A. Simao & C. Morgan (Eds.), *Sbmf 2011: Formal methods, foundations and applications* (Vol. 7021, pp. 124–139). Berlin, Heidelberg: Springer. doi: doi:10.1007/978-3-642-25032-3_9

Drougkas, A., Sarri, A., Kyranoudi, P., & Zisi, A. (2019). *Port cybersecurity: Good practices for cybersecurity in the maritime sector.* ENSISA. doi: doi:10.2824/328515

Elliott, A., & Knight, S. (2010). Role explosion: Acknowledging the problem. In H. R. Arabnia, H. Reza, L. Deligiannidis, J. J. Cuadrado-Gallego, V. Schmidt, & A. M. G. Solo (Eds.), *Software engineering research & practice* (pp. 349–355). Las Vegas, Nevada, USA: CSREA Press. Retrieved from http://knight.segfaults.net/papers/20100502%20-%20Aaron%20Elliott%20-%20WOLRDCOMP%202010%20Paper.pdf

Fatima, A., Ghazi, Y., Shibli, M. A., & Abassi, A. G. (2016). Towards attribute-centric access control: an ABAC versus RBAC argument. *Security and Communication Networks*, *9*(16), 3152–3166. doi: doi:10.1002/sec.1520

Fernández-Medina, E., & Piattini, M. (2004). Extending OCL for secure database development. In T. Baar, A. Strohmeier, A. Moreira, & S. J. Mellor (Eds.), *Uml 2004 — the unified modeling language. modeling languages and applications* (Vol. 3273, pp. 380–394). Berlin, Heidelberg: Springer. doi: doi:10.1007/978-3-540-30187-5_27

Georgiadis, C. K., Mavridis, I., Pangalos, G., & Thomas, R. K. (2001). Flexible team-based access control using contexts. In *Proceedings of the sixth acm symposium on access control models and technologies* (pp. 21–27). New York, NY, USA: ACM. doi: doi:10.1145/373256.373259

Giambiagi, P., Nair, S. K., & Brossard, D. (2015). *Abbreviated language for authorization version 1.0.* OASIS eXtensible Access Control Markup Language (XACML) TC. (https://www.oasis-open.org/committees/download.php/55228/alfa-for-xacml-v1.0-wd01.doc)

Gligor, V. D., Gavrila, S. I., & Ferraiolo, D. (1998). On the formal definition of separation-of-duty policies and their composition. In *Proceedings. 1998 ieee symposium on security and privacy (cat. no. 98cb36186)* (pp. 172–183). IEEE. doi: doi:10.1109/SECPRI.1998.674833

Gogolla, M., Büttner, F., & Richters, M. (2007). USE: A UML-based specification environment for validating UML and OCL. *Science of Computer Programming*, *69*(1–3), 27–34. doi: doi:10.1016/j.scico.2007.01.013

Hamann, L., Sohr, K., & Gogolla, M. (2015). Monitoring database access constraints with an RBAC metamodel: A feasibility study. In F. Piessens, J. Caballero, & N. Bielova (Eds.), *Engineering secure software and systems* (Vol. 8978, pp. 211–226). Cham: Springer. doi: doi:10.1007/978-3-319-15618-7_16

Hu, V. C., Ferraiolo, D., Kuhn, R., Schnitzer, A., Sandlin, K., Miller, R., & Scarfone, K. (2014). Guide to attribute based access control (ABAC) definition and considerations. *NIST Special Publication 800-162*. doi: doi:10.6028/NIST.SP.800-162

Hu, V. C., Kuhn, D. R., & Ferraiolo, D. F. (2015). Attribute-based access control. *Computer*, *48*(2), 85–88. doi: doi:10.1109/MC.2015.33

Joshi, J. B. D., Bertino, E., Latif, U., & Ghafoor, A. (2005). A generalized temporal role-based access control model. *IEEE Transactions on Knowledge and Data Engineering*, *17*(1), 4–23. doi: doi:10.1109/TKDE.2005.1

Jürjens, J. (2002). UMLsec: Extending UML for secure systems development. In J.-M. Jézéquel, H. Hussmann, & S. Cook (Eds.), *Uml 2002 — the unified modeling language* (Vol. 2460, pp. 412–425). Berlin, Heidelberg: Springer. doi: doi:10.1007/3-540-45800-X_32

Kuhlmann, M., Sohr, K., & Gogolla, M. (2011). Comprehensive two-level analysis of static and dynamic RBAC constraints with UML and OCL. In *2011 fifth international conference on secure software integration and reliability improvement* (pp. 108–117). IEEE. doi: doi:10.1109/SSIRI.2011.18

Kuhlmann, M., Sohr, K., & Gogolla, M. (2013). Employing UML and OCL for designing and analysing role-based access control. *Mathematical Structures in Computer Science*, *23*(4), 796–833. doi: doi:10.1017/S0960129512000266

McKinney, S. (2019). *Towards an attribute-based role-based access control system.* (https://s.apache.org/rbac-abac Accessed: July 31, 2020)

Meyer-Larsen, N., & Müller, R. (2018). Enhancing the cybersecurity of port community systems. In M. Freitag, H. Kotzab, & J. Pannek (Eds.), *Dynamics in logistics* (pp. 318–323). Cham: Springer. doi: doi:10.1007/978-3-319-74225-0_43

Meyer-Larsen, N., Müller, R., & Zedel, K. (2019). New concepts for cybersecurity in port communication networks. In *Hamburg international conference of logistics (hicl)* (pp. 543–558). doi: doi:10.15480/882.2483

OMG. (2014). *Object constraint language, OCL version 2.4.* Object Management Group. (https://www.omg.org/spec/OCL/2.4)

Ray, I., Li, N., France, R., & Kim, D.-K. (2004). Using UML to visualize role-based access control constraints. In *Proceedings of the 9th acm symposium on access control models and technologies* (pp. 115–124). New York, NY, USA: ACM. doi: doi:10.1145/990036.990054

Rissanen, E. (2014). *XACML v3.0 core and hierarchical role based access control (RBAC) profile version 1.0.* (OASIS Committee Specification 02 http://docs.oasis-open.org/xacml/3.0/rbac/v1.0/xacml-3.0-rbac-v1.0.html)

Rissanen, E. (2017). *eXtensible access control markup language (XACML) version 3.0 plus errata 01.* OASIS eXtensible Access Control Markup Language (XACML) TC. (OASIS Standard incorporating Approved Errata http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-en.html)

Rubio-Medrano, C. E., D'Souza, C., & Ahn, G.-J. (2013). Supporting secure collaborations with attribute-based access control. In *9th ieee international conference on collaborative computing: Networking, applications and worksharing* (pp. 525–530). IEEE. doi: doi:10.4108/icst.collaboratecom.2013.254168

Sandhu, R. S., Coyne, E. J., Feinstein, H. L., & Youman, C. E. (1996). Role-based access control models. *Computer*, *29*(2), 38–47. doi: doi:10.1109/2.485845

Simon, R., & Zurko, M. E. (1997). Separation of duty in role-based environments. In *Proceedings 10th computer security foundations* (pp. 183–194). IEEE. doi: doi:10.1109/CSFW.1997.596811

Sohr, K., Drouineaud, M., Ahn, G.-J., & Gogolla, M. (2008). Analyzing and managing role-based access control policies. *IEEE Transactions on Knowledge and Data Engineering*, *20*(7), 924–939. doi: doi:10.1109/TKDE.2008.28

Sohr, K., Kuhlmann, M., Gogolla, M., Hu, H., & Ahn, G.-J. (2012). Comprehensive two-level analysis of role-based delegation and revocation policies with UML and OCL. *Information and Software Technology*, *54*(12), 1396–1417. doi: doi:10.1016/j.infsof.2012.06.008

Steinberg, D., Budinsky, F., Paternostro, M., & Merks, E. (2008). *Emf: Eclipse modeling framework* (2nd ed.). Addison-Wesley.

Strembeck, M., & Mendling, J. (2011). Modeling process-related RBAC models with extended UML activity models. *Information and Software Technology*, *53*(5), 456–483. doi: doi:10.1016/j.infsof.2010.11.015

Warmer, J., & Kleppe, A. (2003). *The object constraint language: Getting your models ready for mda* (2nd ed.). Addison-Wesley.

Yu, L., France, R. B., & Ray, I. (2008). Scenario-based static analysis of UML class models. In K. Czarnecki, I. Ober, J.-M. Bruel, A. Uhl, & M. Völter (Eds.), *Model driven engineering languages and systems* (Vol. 5301, pp. 234–248). Berlin, Heidelberg: Springer. doi: doi:10.1007/978-3-540-87875-9_17

Zhang, L., Ahn, G.-J., & Chu, B.-T. (2003). A rule-based framework for role-based delegation and revocation. *ACM Transactions on Information and System Security*, *6*(3), 404–441. doi: doi:10.1145/937527.937530

About the authors

Christian Maeder is a member of the Software Engineering Group headed by Rainer Koschke https://www.informatik.uni-bremen.de/st. You can contact the author at c.maeder@uni-bremen.de.

Karsten Sohr is a senior researcher at the Center for Computing Technologies (TZI) at the University of Bremen and the coordinator for the development of the topic "Information Security". You can contact the author at sohr@uni-bremen.de.

Rodrigue Wete Nguempnang is a member of the Software Engineering Group headed by Rainer Koschke https://www.informatik.uni-bremen.de/st. You can contact the author at wete@uni-bremen.de.

Nils Meyer-Larsen is a project manager at ISL in Bremerhaven and head of ISL's competence area "Maritime Security". You can contact the author at meyer-larsen@isl.org.

Rainer Müller is a senior researcher at ISL in Bremen. Main topics of his research are Supply Chain Risk Management, Supply Chain Security and resilience in maritime and intermodal transport. You can contact the author at rmueller@isl.org.