Karlsruhe Institute of Technology

Software Design and Quality

Context-based confidentiality analysis in dynamic Industry 4.0 scenarios

Bachelor's Thesis of

Jonathan Schenkenberger

at the Department of Informatics
Institute for Program Structures and Data Organization (IPD)

Reviewer:	Prof. Dr. Ralf H. Reussner
Second reviewer:	Prof. Dr.-Ing. Anne Koziolek
Advisor:	M.Sc. Maximilian Walter
Second advisor:	M.Sc. Stephan Seifermann

04. June 2019 – 04. October 2019

I declare that I have developed and written the enclosed thesis completely by myself, and have not used sources or means without declaration in the text.

PLACE, DATE

. . . .

(Jonathan Schenkenberger)

Abstract

In Industry 4.0 environments highly dynamic and flexible access control strategies are needed. State of the art strategies are often not included in the modelling process but must be considered afterwards. This makes it very difficult to analyse the security properties of a system. In the framework of the Trust 4.0 project the confidentiality analysis tries to solve this problem using a context-based approach. Thus, there is a security model named “context metamodel”. Another important problem is that the transformation of an instance of a security model to a wide-spread access control standard is often not possible. This is also the case for the context metamodel. Moreover, another transformation which is very interesting to consider is one to an ensemble based component system which is also presented in the Trust 4.0 project. This thesis introduces an extension to the beforementioned context metamodel in order to add more extensibility to it. Furthermore, the thesis deals with the creation of a concept and an implementation of the transformations mentioned above. For that purpose, at first, the transformation to the attribute-based access control standard XACML is considered. Thereafter, the transformation from XACML to an ensemble based component system is covered. The evaluation indicated that the model can be used for use cases in Industry 4.0 scenarios. Moreover, it also indicated the transformations produce adequately accurate access policies. Furthermore, the scalability evaluation indicated linear runtime behaviour of the implementations of both transformations for respectively higher number of input contexts or XACML rules.

Zusammenfassung

In Industry 4.0 Umgebungen werden hochgradig dynamische und flexible Zugriffskontrollstrategien benötigt. Die Strategien des Standes der Technik sind häufig nicht im Modellierungsprozess enthalten und müssen später betrachtet werden. Dies macht es äußerst schwierig die Sicherheitseigenschaften des Systems zu analysieren. Im Rahmen des Trust 4.0 Projektes versucht die Vertraulichkeitsanalyse dieses Problem zu lösen indem sie einen kontextbasierten Ansatz verwendet. Deshalb existiert ein Sicherheitsmodell namens "Context Metamodel". Ein weiteres wichtiges Problem ist, dass die Transformation einer Instanz des Sicherheitsmodelles zu einem weit verbreiteten Zugriffskontrollstandard oftmals nicht möglich ist. Das ist auch der Grund warum es das Context Metamodel gibt. Des Weiteren betrachtet diese Arbeit eine andere Transformation, welche sehr interessant ist, nämlich die eines Ensemble basierten Komponentensystems das ebenfalls im Trust 4.0 Projekt präsentiert wird. Diese Arbeit stellt eine Erweiterung des zuvor genannten Context Metamodel vor, um mehr Erweiterbarkeit hinzuzufügen. Außerdem stellt diese Arbeit die Erstellung des Konzeptes und der Implementierung der oben genannten Transformationen dar. Deshalb wird zunächst die Transformation zu dem attributbasierten Zugriffskontrollstandard XACML betrachtet. Danach wird die Transformation von XACML zu einem Ensemble basierten Komponentensystem betrachtet. Die Evaluation deutete an, dass das Modell für Anwendungsfälle in Industry 4.0 Szenarien verwendet werden kann. Des Weiteren deutete sie ebenfalls an, dass die generierten Zugriffskontrollartefakte der Transformationen hinreichend akkurate Ergebnisse lieferte. Außerdem deutete die Skalierbarkeitsevaluation lineares Laufzeitverhalten für die Implementierungen beider Transformationen für eine respektive höhere Anzahl von Eingabekontexten oder XACML Regeln.

Contents

List of Figures

List of Listings

List of Tables

1. Introduction

Industry 4.0 is an optimization initiative which adds dynamic actions to industrial processes like supply chains. It uses IoT systems and self-organizing systems which interact in an ad-hoc manner with machines, persons and organizations involved in the process.

1.1. Motivation

In Industry 4.0 scenarios, access control mechanisms play an important role. However, the state of the art models for the subject of controlling access to data are not directly applicable to the use cases required in dynamic environments. An example for such dynamic environments are supply chains in which access to certain data can be varied for different organizations or persons in different contexts. Another important difficulty is the high amount of flowing data and flexibility needed. This is the reason for a model which ensures confidentiality in such fast-changing environments. Furthermore, model based security enables us to check security properties at the model level instead at the implementation level. This enables us to detect errors earlier in the process of setting up a secure system.

There is already the Trust 4.0 project which started to address the before-mentioned problems concerning access control in systems used in supply chains. Furthermore, there already exists a context metamodel [11, p. 11] which addresses the problem of fast changing contexts.

Another important motivation for this thesis is the advantage of automatic generation of access control artefacts from a model based security system. The automatisation ensures an accurate transformation of the model creator's intention to policies usable in access control decision. For example, all the modelled elements like *Context*s are transformed to respective checks in the generated policies. Especially, if many rules or rules with many checks are modelled, a manual creation of policies is very error-prone whereas an automatic generation ensures the correct and accurate transformation of all checks. Moreover, the developer can use her/his valuable time for more complicated tasks which cannot be automatised as easily.

A further advantage of model based security systems is the integration of the security definition in the software modelling process. This allows the model creator to consider security related issues at a higher level of abstraction. Thereafter, the creation of actual access control artefacts can be done automatically, so no one needs to explicitly care about the implementation of the modelled elements.

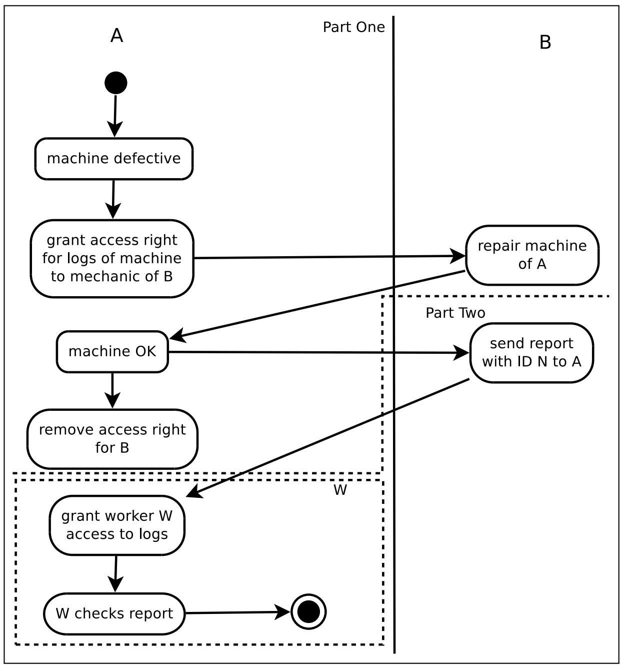

A
Part One
B
machine defective
grant access right for logs of machine to mechanic of B
repair machine of A
Part Two
machine OK
send report with ID N to A
remove access right for B
W
grant worker W access to logs
W checks report

Figure 1.1.: Running Example.

As a running example (see Figure 1.1) we consider the following: a machine of organization A has a malfunction and must therefore be checked by a mechanic of a different company B. For that purpose, the mechanic must obtain access to the logs of the machine. However, B should lose access rights to the machine as soon as the repair is finished. Part two of the example consists of a report with ID N which is sent by B after the repair was successfully finished. The arrival of this report is a prerequisite for an inspector W of A to inspect whether the machine was acceptably repaired. To this end, W gets access to the logs of the machine. The two parts of the example are separated by the dashed line.

1.2. Contribution

In my thesis, I am going to extend the context metamodel in order to facilitate the definition of more fine-grained access control constraints. This enables us to represent more complex usage scenarios in an Industry 4.0 environment. For example, the second part of the running example cannot be modelled using the original version of the context metamodel because the metamodel does not consider prerequisites. Additionally, an automatic generation of access control artifacts will be implemented. Thus, there will be an implementation of an automatic transformation from context model instances to XACML policy sets which are then automatically transformed to an ensemble system for security.

2. Foundation

This chapter describes the foundation of the thesis. As a foundation for the architectural component model the Palladio Component Model (see Section 2.1) and its data-flow extension Data-centric Palladio (see Section 2.2) will be used. Moreover, as a pattern for policy generation XACML and ensemble based component systems will be used (see Sections 2.4, 2.5). The model which will be extended and for which the policy generation will be implemented is described in Section 2.3.

2.1. Palladio Component Model

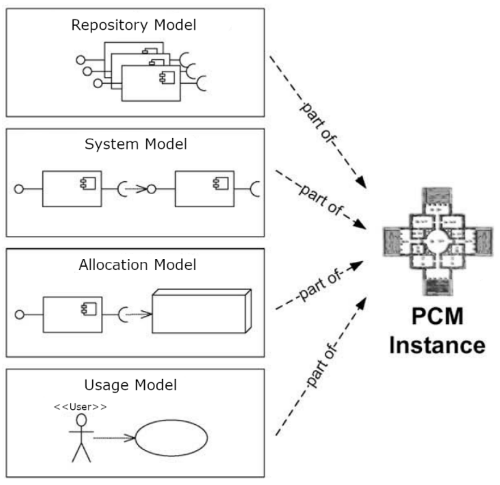


Figure 2.1.: Composition of a Palladio Component Model Instance (inspired by [7]).

Palladio is a tool-supported simulator for software architecture. It can be used to predict several properties concerning quality of software. These are for example reliability and performance. It has initially been developed by Karlsruhe Institute of Technology (KIT), FZI Research Center for Information Technology, and University of Paderborn.

The Palladio Component Model (PCM) is a detailed metamodel of component-based software architectures which is developed using the Eclipse Modelling Framework (EMF) [6]. The model is used to describe software by describing for example components, connectors, interfaces and other elements which can be used to determine the systems' performance or reliability [8].

An instance of the basic PCM consists of five models. Figure 2.1 shows all models except the runtime model which is used for extensions at the run-time level. The repository model describes the components and their interfaces. Service effect specifications (SEFF) thereby describe the inner behavior of the components. The SEFF is an abstraction from the control flow in order to render concentrating on relevant information for analysis possible. The system model represents the software architecture by combining the components of the repository model. The allocation model describes the deployment of components on different resources. The usage model describes the users' interaction with the system [8].

2.2. Data-centric Palladio

Data-centric Palladio (DCP) is an extension of the PCM which makes it possible to include data-flows into the model. It allows software developers to model data-flows by using data sources, sinks and processing operations. Sources represent creation points of data inside the data flow whereas sinks represent the end of a flow. Processing Operations are used for accessing and working with the data of the flow [24].

Furthermore, basic access control can also be modelled. To achieve this, the model makes use of so-called characteristics. In this model each characteristic has a type which specifies its purpose. These characteristics are used to describe metadata of the data. It is important to understand that this metadata is rather abstract so that different kinds of metadata can be used in different scenarios. One of the access control strategies which are made possible consists of role based access control for which roles and access rights are defined as types for characteristics. The access rights are added to the respective data as metadata and the roles are added to the processing operations. To achieve a proper confidentiality analysis, the access rights linked to the data are compared to the roles linked to the data processing operation [24].

The data is modelled on a metalevel which means data types are modelled rather than actual data. This has the advantage that the resulting models are more generally usable. Moreover, it is also possible to implement an analysis which considers the actual model including data-flows, additional information like access control policies and analysis goals. These goals can be defined beforehand in a logic program which can thereafter be tested using a logic programming environment [24].

2.3. Context Metamodel

The context metamodel is an extension of DCP and represents dynamic context information in order to describe dynamically changing properties of components in an Industry 4.0 environment. The model consists of three main parts. First, the subjects which represent the acting components in the system. Second, the contexts which represent important information concerning access control. Third, the helperattributes package that holds classes to describe the structure of the system. The helperattributes and the subject package are subpackages of a package called util.

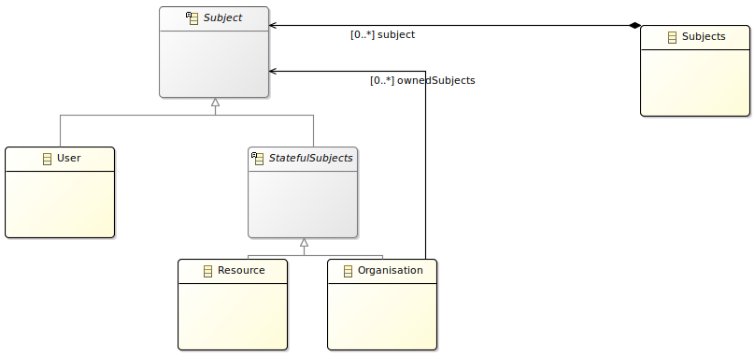


Figure 2.2.: Subject Package of the DynamicContextModel

As shown in Figure 2.2, the subjects are divided into resources, users and organizations. Resources represent non-human actors and users human actors. An organization is a composite subject, which means it can contain other subjects like sub-organizations, workers (users) and resources. Every subject has properties which are saved in a context. The difference between resource and user is that users can directly communicate with each other using physical data, whereas resources must use the underlying system in order to be able to interact with each other [11, pp. 11-12]. Resources and organizations are classified as stateful subjects whereas users do not have a state and are therefore directly classified as subjects.

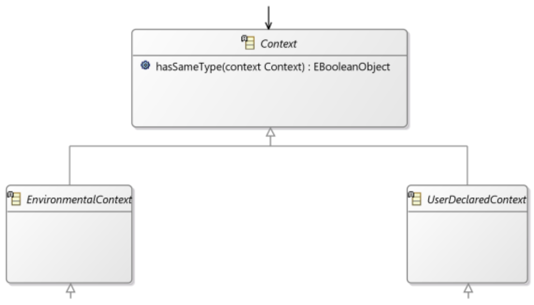


Figure 2.3.: Base of the Context Class Hierarchy [11, p. 12].

The base of the Context class hierarchy as shown in Figure 2.3 consists of an abstract class *Context* which has two abstract subclasses *EnvironmentalContext* and *UserDeclaredContext*. The first one is the super class for contexts concerning the subject's environment, the second one for contexts which are defined by users. All contexts inherit the method hasSameType which checks if two contexts are of the same type and can therefore be compared [11, pp. 12-13].

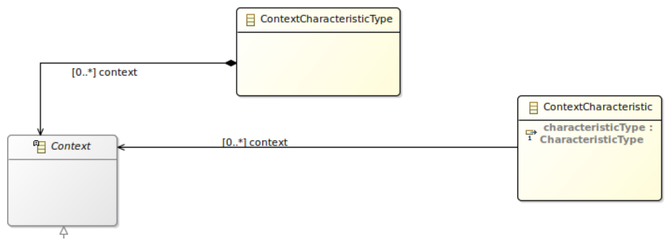

ContextCharacteristicType
[0..*] context
Context
[0..*] context
ContextCharacteristic
characteristicType : CharacteristicType

Figure 2.4.: The Connection to Data-centric Palladio.

As shown in Figure 2.4, the integration into data-centric Palladio is modeled using a new *ContextCharacteristic* and a respective type. The characteristic can hold several contexts.

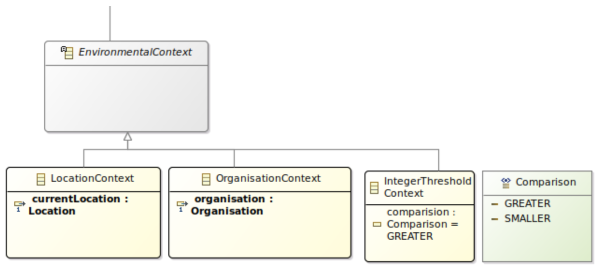


Figure 2.5.: The different environmental Contexts.

As depicted in Figure 2.5, the environmental contexts consist of a context representing membership in an organization and a context concerning location. Moreover, the *IntegerThreshold-Context* allows us to compare with an integer threshold which must be fulfilled.

Additionally, there are also user declared contexts which represent more complex user defined access control features [11, p. 14]. Figure 2.6 illustrates there are also several user declared contexts which consist of a *ShiftContext* and a *RoleContext* which describe respectively the shift and the role that a user fulfills. The *InternalStateContext* represents a resource which needs to have the same internal state as the context [11, p. 14]. An example for an internal state could be whether a machine is defective or normally working. The *PrivacyLevelContext* checks whether the declared privacy level is fulfilled. The *ShiftCheckContext* checks if the current shift matches the user's shift.

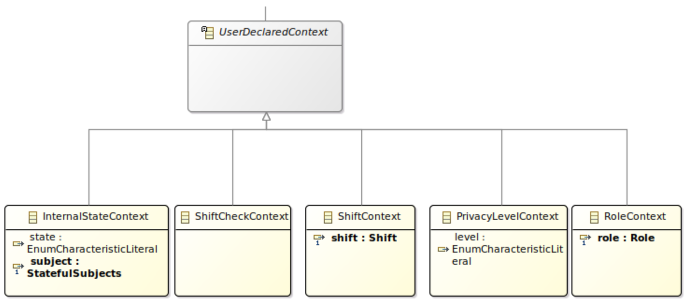


Figure 2.6.: The different User declared Contexts.

The Helperattributes package exists in order to model the environment of the system. It contains the model for locations, roles and shifts. The classes of this package are used by classes of the context package.

2.4. Ensemble-based Component System (EBCS)

In order to evaluate security in dynamic systems the usage of autonomic component ensembles can be helpful.

An ensemble is a group of several components which perform actions to fulfill a certain goal. The evaluation of a membership condition is done at runtime. It is also possible for a component to be a member of different ensembles. This is semantically equivalent to components performing different independent tasks. Furthermore, ensembles can also consist of sub-ensembles. This is semantically equivalent to sub-tasks performed by the different sub-ensembles. The combination of the sub-tasks then make up the main task performed by the top-level ensemble [1] [15].

In the scenario of security in component systems, ensembles are used to describe types of situations. Instead of defining joint actions, access control rules are defined. In static access control strategies these rules do mostly not consider dynamic context changes (see Chapter 3). The advantage of the approach using ensembles is that it is possible to adapt access control rules to dynamically changing contexts like for example different states of a component. This is the case because ensembles are context dependent and instantiated at runtime, so a changed state can be considered during evaluation of a rule [1].

In the process of this thesis, a newer implementation of the ensemble security system will be used (see section 5.1). The ensemble security system uses Scala as a programming language. Components are modeled as classes in order to make multiple instantiations possible. Ensembles can be modeled as singleton objects or as classes. In the example presented in the paper and in the generated ensemble system explained in this thesis the ensembles which represent the access control rules are modeled as singleton objects. An ensemble used as a security rule consists of different values which filter the components of the ensemble, an optional *situation* call, an optional *constraints* call, and one or more calls to allow or deny. The situation is used to describe environmental condition, i.e. a condition concerning time or location. The constraints are used to define further constraints for the ensemble to be instantiated. The call to allow or deny is then used to determine which effect the rule has. Furthermore, the different rules should be added in the root ensemble with a call to the *rules* function [3].

In listing 2.1 one can see an easy ensemble system for a test action which is implemented as an ensemble and requires the subject's location to be equal to a certain string literal. At the beginning of the model there is the definition of the used component classes and its attributes (l. 3-11). The actual definition of the ensemble representing the action can be found at l. 14-23. The *situation* and *constraints* calls are not shown in this example. They would be between the attribute definitions and the call to allow (l. 21). Moreover, all resources are allowed in this example (l. 15-16). They could also be checked in more detail with the filter method like it is done in order to define the *allowedSubjects* (l. 19). In this example, only subjects with location equal to "Production_Hall" are allowed (l.19). In the end, the map function called at the end of the definition of the attributes of the ensemble exists in order to enable the Scala compiler to be sure that the elements are always of the type *Component* (l. 20). Thus, the attributes *allowedSubjects* and *allowedResources* can be used as argument in the call to allow. At the end of the root ensemble definition further system wide calls can be done like adding rules, i.e. ensembles, to the root ensemble (l. 25). At the end of the model definition, further model wide calls can be done like extension methods or prerequisite handlers. Moreover, there must always be the definition of the root ensemble (l. 28). The listing omits the imports.

```
 1  class EasyExample(val now: LocalTime) extends Model {
 2
 3    class Subject(val subjectName: String, val location: String = null)
 4      extends Component {
 5      name(s"Subject $subjectName")
 6    }
 7
 8
 9    class Resource(val resourceName: String, val accessSubject: Subject) extends Component {
10      name(s"Resource $resourceName")
11    }
12
13    class System extends RootEnsemble {
14      object testAction extends Ensemble {
15        val allowedResources = components.select[Resource]
16            .map[Component](x => x.getClass().cast(x))
17        val allowedSubjects = components
18            .select[Subject]
19            .filter(x => x.location == "Production_Hall")
20            .map[Component](x => x.getClass().cast(x))
21
22        allow(allowedSubjects, "testAction", allowedResources)
23      }
24
25        val testActionRule = rules(testAction)
26    }
27
28    val rootEnsemble = root(new System)
29  }
```

Listing 2.1: An easy Ensembles System Example without Imports.

In listing 2.2 one can see a matching scenario definition for the example ensemble shown above. The scenario consists of a model definition (l. 3-4), subject and resource definitions (l. 5-7). Thereafter, the components (here: subjects and resources) are set as a list as the components of the model (l. 8). Then, the root ensemble which contains all ensembles (in this example only one ensemble) is initialized with a call to *init* (l. 9). After this, the root ensemble is solved with a call to *solve* (l. 10). In the end, the *exists* call on the specified rule's selected members can be used to determine the access control decision (l. 11-14). The result is an *allow* decision because *subjectA* is contained in the allowed subjects of the *testAction* ensemble. It is also possible to make other user-defined or more complex queries in the *exists* call (ore several calls to *exists*) in order to be able to model more complex scenarios. The figure omits the helper method *convertToCol* which converts a scala *Iterable* to a Java *Collection* in order to be able to test more complex scenarios.

```
1  object EasyExample {
2    def main(args: Array[String]) : Unit = {
3      val scenario = new RunningExample(LocalTime.parse("13:00:00Z",
4             DateTimeFormatter.ISO_OFFSET_TIME))
5      val subjectA = new scenario.Subject("A", "Production_Hall")
6      val subjectB = new scenario.Subject("B", "Location B")
7      val resourceA = new scenario.Resource("machine", subjectA)
8      scenario.components = List(subjectA, subjectB, resourceA)
9      scenario.rootEnsemble.init()
10     val solved = scenario.rootEnsemble.solve()
11     val testActionAllow = solved && scenario.rootEnsemble.instance
12       .testActionRule.selectedMembers.exists(
13         x =>  convertToCol(x.allowedSubjects).contains(subjectA) &&
14           !convertToCol(x.allowedSubjects).contains(subjectB))
15     if(testActionAllow) {
16       println("allow")
17     } else {
18       println("deny")
19     }
20   }
21 }
```

Listing 2.2: An easy Ensembles System Scenario definition Example without Helper Method.

2.5. ABAC - Attribute Based Access Control

In this section, I am going to describe the concepts of attribute based access control in general and XACML in particular. It is used as an intermediate model on the way of transforming a model instance of the context metamodel to an ensemble system.

2.5.1. General Information about Attribute Based Access Control

An often used access control strategy is Attribute Based Access Control (ABAC). It uses attributes which are name-value pairs. These attributes are given to different subjects and objects. Access control policies are then used in order to evaluate if the attributes match. The decision can also be dependent of further environmental conditions [16]. More information which are not essential for understanding the concept of this thesis but are rather descriptions of other state of the art access control strategies concerning ABAC can be found in section 3.3.2.

2.5.2. XACML - eXtensible Access Control Markup Language

Since attribute based access control does not work on the data flow, it is not directly usable for our case [5]. However, XACML is used as a intermediate model in order to create ensembles more easily. That is why, the necessary parts for our policy generation will be explained briefly in this section.

The basic structures of XACML 3.0 policies which are going to be used consist of policy sets, policies, rules, targets and matches. A policy set contains one or more policies which are combined with a policy-combining algorithm. Each policy contains one ore more rule(s) which are combined with a rule-combining algorithm. Each rule contains different matches which check the different attribute values with a match function against the respective request values. The matches are contained in a structure which is called *All-Of* which is contained in a structure called *Any-Of*. The *All-Of* structure requires that all the contained matches are correctly matched. The *Any-Of* structure can contain one or more *All-Of* structures from which one of them must be matched so that the *All-Of* structure is matched. The target of the rule contains zero or more *Any-Of* structures. The target determines if the respective rule is applicable, if it is applicable and the optional condition is met, then the effect (e.g. *permit*) of the rule is applied [26].

Another important concept of XACML are *Obligation*s which must be fulfilled by the policy enforcement point (PEP) [26]. However, the default implementation of the PEP by the used library does only show the obligations and does not automatically fulfil them. Fur further information concerning the used XACML library see section 5.1.

The different matches contained in the *All-Of* structure(s) determine which and how attributes should be checked. Listing 2.3 shows an example of a match structure. The *MatchId* refers to the used match function (l. 1). The *AttributeValue* structure determines the datatype and the value, which the request attribute must match according to the match function (l. 2-4). The

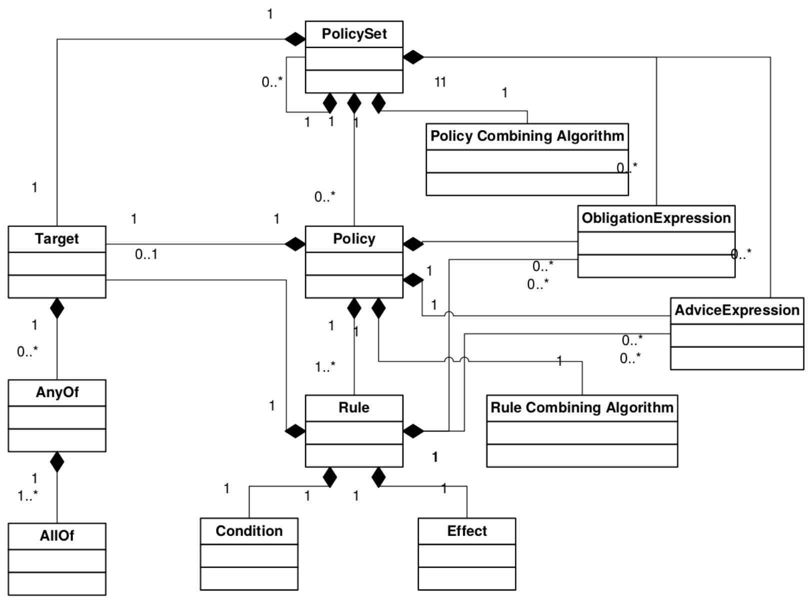


Figure 2.7.: XACML Policy Language Model [26].

AttributeDesignator structure determines which attribute (i.e. category and id must match) should be matched against the value defined in the *AttributeValue* structure (l. 5-8).

```
1  <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-regexp-match">
2    <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
3      John.*
4    </AttributeValue>
5    <AttributeDesignator
6      Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject"
7      AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
8     DataType="http://www.w3.org/2001/XMLSchema#string"/>
9  </Match>
```

Listing 2.3: Example of a Match Definition [26].

The request contains the different attributes of the different categrories which are *action*, *subject*, *resource* and *environment*. As Figure 2.8 shows, the policy enforcement point (PEP) decides whether the access can be granted or not. It uses the Policy Decision Point (PDP) which uses the policy repository controlled by the Policy Administration Point (PAP). Moreover, the PDP uses also the Policy Information Point (PIP) which loads the attributes of all categories including environmental attributes like for example the current date and time [26].

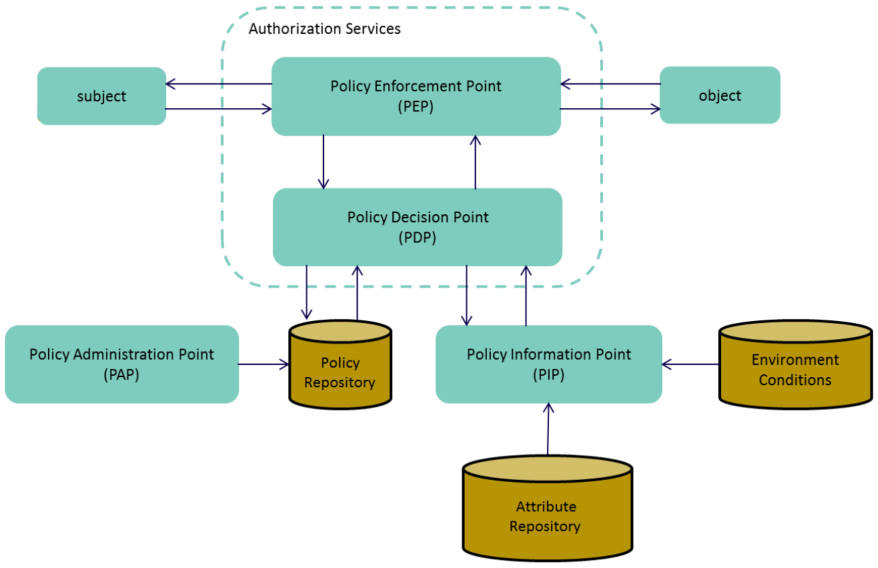

Authorization Services
subject
Policy Enforcement Point (PEP)
object
Policy Decision Point (PDP)
Policy Administration Point (PAP)
Policy Repository
Policy Information Point (PIP)
Environment Conditions
Attribute Repository

Figure 2.8.: XACML Access Control Decision Structure [16, p. 15].

3. State of the Art

This chapter describes state of the art strategies for preserving confidentiality and access control.

3.1. DRM - Digital Rights Management

Digital Rigths Management (DRM) systems consist of different servers which are communicating with each other in order to provide data to customers. In this scenario it is important that data is secured using a set of policies. However, this system is not very flexible because data is usually stored once at one specific location and thereafter not often changed [27]. It follows that this approach cannot be used in Industry 4.0 contexts because in this case data is often changed and distributed in the whole system.

An extension of DRM is E-DRM which allows data to be changed. It uses OrBAC [20] which is described briefly in subsection 3.3.3. Nevertheless, E-DRM is not applicable to Industry 4.0 scenarios because the data in E-DRM is still stored in files, whereas in Industry 4.0 it is often calculated during communication [11, p. 4]. However, it could still be used for data which leaves the company network.

3.2. R-PRIS - Runtime model-based PRIvacy CheckS

Another approach consists of Runtime model-based PRIvacy checkS (R-PRIS). It uses a runtime model and a checking algorithm. Though, it uses only geo-location for evaluating access rights [23]. This means it does not directly work on the data flow, which makes it not applicable in our case. Moreover, other important attributes used in Industry 4.0 access control surroundings are not considered in the R-PRIS approach. That is why, it can not directly be used for the use case of dynamic Industry 4.0 scenarios.

3.3. Access Control

The two often used access control methods are Discretionary Access Control (DAC) and Mandatory Access Control (MAC) which additionally requires a passphrase. These policies associate access rights directly to specific users [29] [13]. However, there are also other state of the art access control strategies which are described in the following subsections of this section.

3.3.1. RBAC - Role Based Access Control

Role based access control (RBAC) uses roles instead of users, which makes it possible to define user groups. This approach consists of users, which are members of roles. These roles can be used to define access rules to certain operations. It is also possible to define role hierarchies that enable us to create sub-groups. Furthermore, RBAC guarantees that users do not have more access than defined by their roles [14]. This access control strategy was used for example in SecureUML (see subsection 3.4.2).

However, dynamically changing contexts are not possible with this strategy because roles are rather static [30] and not able to be changed because of dynamic context changes. This can easily be seen regarding part one of the running example where B must only get access if the machine is defective. Since roles cannot change dynamically it is not possible to model this scenario in RBAC.

3.3.2. ABAC - Attribute Based Access Control

This section explains the usage of ABAC in an Industry 4.0 environment and a comparison with the more simple RBAC approach.

The ABAC system initially needs more effort to be set up than the RBAC system. However, the administration of the attribute system is much easier and more dynamic because new subjects must not be known to the system. Only the attributes must be known, which makes the access control model much more extensible. In RBAC, the roles for new subjects must be set explicitly. This leads to a higher administration effort. This advantage of ABAC is very important for Industry 4.0 environments where the different subjects are not necessarily known at the time of creation of the access control system [30].

Another important advantage of the ABAC approach is the fact that environment attributes can be checked, which makes it possible to create dynamic rules which also consider e.g. the actual location or time of the access [30].

3.3.3. OrBAC - Organization Based Access Control

Organisation based access control (OrBAC) makes it possible to model access control policies using contexts. It defines organisations and users as subjects. Furthermore, roles and contexts specify how access control rights can be given. When OrBAC was presented for the first time, contexts were not categorized more specifically [18].

However, a later paper introduced a classification of contexts. As shown in Figure 3.1, there are several different sub-classes of contexts. There are temporal contexts which represent a certain time and spatial contexts which represent a location determined for example by the user's ip address. Moreover, there is also a prerequisite context which can be used to grant access only in a certain pre-condition stored in the system database is fulfilled. Additionally, there is a provisional context which considers actions triggered by an action which happened beforehand. At last, there are user-declared contexts which can be used for further purposes defined by the user [12]. This model can also be seen as a prototype for the context metamodel.

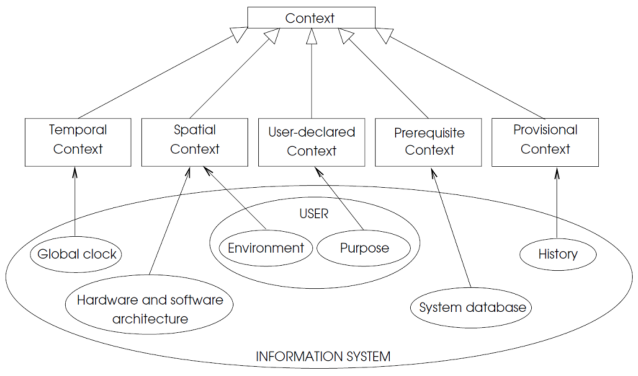


Figure 3.1.: Structure of Different Context Classes in OrBAC [12].

However, joining of entities is not possible. Thus, horizontal cooperation of organizations cannot be modeled using OrBAC. An extension of OrBAC is Coalition OrBAC which enables us to join different entities but only the ones which are of the same type [9].

3.4. Model-Driven Development of Secure Systems

In this section, the advantages of model-driven development for security are briefly explained. An important benefit of this approach is the fact that security questions can be already considered at modelling time. Moreover, the security can be checked in an earlier stage of development with help of model validations tools and easier formal checking than if security is added later in the development process. Furthermore, models are platform independent and less prone to errors than other possibilities for achieving security in software systems [21].

3.4.1. UMLSec

UMLSec can be used to model confidentiality on the control path. Thereby, it can ensure save communication and role based access control. There are four security requirements which are considered in the UMLSec extension. First, there is fair exchange which avoids cheating commited by either party of the communication. Second, confidentiality can also be modeled, which allows that only the legitimate receiver can access the confidential information. Third, secure information flow can also be used to guarantee that no information - not even partly - is leaked. Last, a secure communication link respective a given attacker model is contained in UMLSec as well [17]. However, dynamic formulation of constraints is difficult because UMLSec does not work on the data path but on the control path [4].

3.4.2. SecureUML

SecureUML is an approach which enables the modeller to define roles, permissions and users in an UML diagram. The significant information for access control is added using annotations which can be added to any UML element. Thereby, every element can be used as a secure resource. It is also possible to automatically generate access control systems in the form of policies. Moreover, the SecureUML approach is based on an extended RBAC model [19]. Thus, it cannot easily be used for context based or attribute based access control models like used in this thesis.

3.4.3. SECTET

Another approach which uses the model-driven development paradigm is SECTET. The paper explains the transformation of high level model instances to low level access control policies written in XACML. To achieve this, the SECTET UML meta-model is transformed to a XACML meta-model using the operational Query View Transformation (QVT) language. Thereafter, model instances can be created using EMF. These instances can then be transformed to XACML policy files using the XPAND language by OpenArchitectureWare [2]. The main differences between the SECTET approach and the approach shown in this thesis are that in SECTET another initial model is used and this thesis additionally considers the generation of an ensemble based system from the XACML intermediate model.

3.4.4. ACT

Another approach, which also considers creation of XACML policy files from a model defined security system is described in the paper "A Toolchain for Designing and Testing Access Control Policies". The described toolchain is called Access Control Testing toolchain (ACT) and consists of graphical model-driven design, access control policy generation and automatized testing of the generated policies. Moreover, there is also a tool which can be used to analyse certain properties concerning the consistency of model and policies like for example detecting errors in policy definitions in relation to the defined model. As a graphical access control model UWE (UML-based Web Engineering) is used. UWE is a UML extension similar to SecureUML and UMLSec. But in contrast to UMLSec, UWE is less detailed [10]. The main difference of this approach in comparison with the one presented in this thesis is that the initial model for the transformation is different. The model presented in the paper is a security UML extension based on RBAC whereas the initial model for this thesis uses an context based approach.

4. Concept

The thesis consists of two parts: extending the context metamodel (see Section 2.3) and improving the automatic generation of access control artifacts in the form of ensembles.

4.1. Extending the Context Metamodel

The context metamodel needs to be extended in order to enable us to model more use cases. Especially, already existing use cases should be modeled in a more general way. For instance, the part two of the running example cannot be modeled by the metamodel as it is at the moment. The reason for this is that it is not possible to model prerequisites.

4.1.1. General Information concerning the Extension of the Metamodel

Regarding the extension of the metamodel, the *IntegerThresholdContext* is renamed to a context named *IntegralTresholdContext* which is now a subclass of the abstract class named *Comparison-Context*. This context contains the respective comparison (greater or smaller). The threshold value is defined in the subclasses of *ComparisonContext*. Furthermore, another context named *FloatingComparisonContext* which contains a double threshold is added as another subclass of *ComparisonContext*. Generally, the comparison is defined as: the defined threshold value must be compared to the request value. For example if the threshold is 5 and the Comparison is *greater*, then the request value must be smaller or equal to 5.

Moreover, there is a new *PrerequisiteContext* which checks whether a certain prerequisite is fulfilled. This context is similar to the PrerequisiteContext in OrBAC (see Section 3.3.3). For that purpose there is an addition of a *Prerequisite* class in the helperattributes package. Instances of this class contain a reference to an *OperationSignature* which defines the prerequisite. Due to this modeling, it is possible to add further prerequisites during later design time. An example for a prerequisite consists of a report with an ID that should be arrived before access can be given. This can be modeled with a respective *OperationSignature* which describes an arriving report. The prerequisites are stored in a *PrerequisiteContainer*.

Another important addition to the existing metamodel is the creation of an extension mechanism which can be used to specify checking code inside a generated ensemble after generation. This adds lots of extensibility to the access control possibilities represented by the metamodel. For instance, the code added later could dynamically check a state of an external system or a combination of attributes and external states. The extension mechanism is realized with an *ExtensionContext* which defines a method name which should be created and inserted into the ensemble system (see also sections 4.2, 4.3). Additionally, the context also defines whether the method should work on the component (i.e. `isAtEnd`) or not (i.e. `!isAtEnd`).

A last minor change is that the *ShiftCheckContext* is removed because the matching of the shift against the current time should be always be done. Therefore, the *ShiftCheckContext* is now obsolete.

Figure 4.1 summarizes the beforementioned changes by showing them highlighted in green.

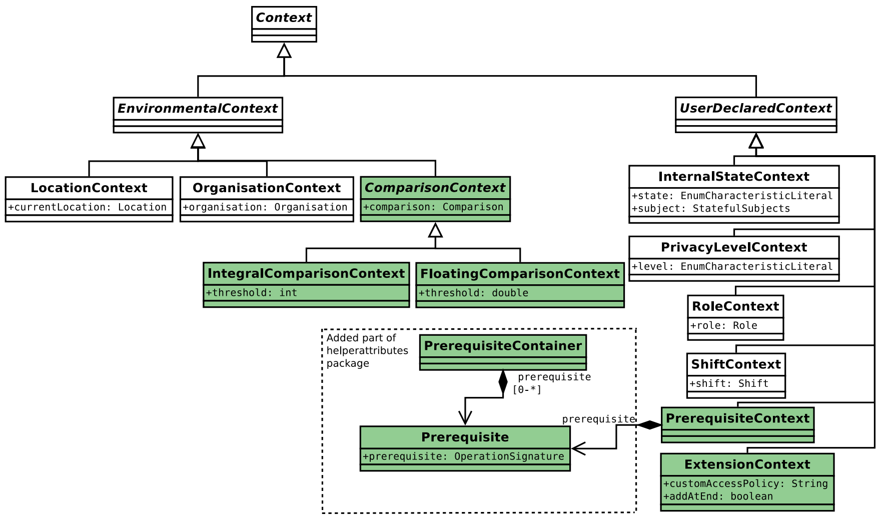


Figure 4.1.: Summary of Extensions to the Metamodel.

4.1.2. Running Example

The running example introduced in the motivation (see Section 1.1) is modelled using two characteristics which represent two rules. The first rule is for the situation when the machine is in the *OK* state. That is why this rule contains an *InternalStateContext* containing the *OK* state. Additionally, this rule contains role and organisation contexts to check if the access subject is a worker of organisation A. Moreover, the rule contains the newly created *PrerequisiteContext* to check whether the report was sent. In the later implementation, the report must be sent when the change from the *DEFECTIVE* to the *DEFECTIVE* state happens. This must be considered when implementing the system because the generation software does not know about the underlying system. The second rule contains an *InternalStateContext* containing the *DEFECTIVE* state and a role and organisation context to check if the access subject is a repairperson of organisation B. In order to keep the example simple, the organisation A releases all access rights to the logs of the machine. In reality, there would be an additional rule to ensure that organisation A can always somehow access its own machine.

4.2. Generating the XACML Intermediate Model

This section describes the transformation from PCM context model instances to XACML policy set files.

4.2.1. Advantages of the Intermediate Model Approach

The advantages of first creating the XACML intermediate model instead of directly generating an ensemble system consist of the use of a wide-spread standard which can also be used in systems which do not employ ensemble based security systems. Moreover, the generation of ensembles from an XACML policy set file is easier and more understandable than the direct creation from context model instances. Another important advantage is the easier optimization and an easier integration into other systems.

4.2.2. General Information concerning the Generation

The concept of the generation of this intermediate model is descibed in this section. An entity of the related characteristics in the context model instance which represents an action is transformed to a XACML policy. All the policies are combined in one policy set which represents the whole model instance. As a policy and rule combining algorithm *Permit-Overrides* is used which is equivalent to a logical *OR*.

The different context containers are mapped to different rules which are combined with a logical *OR*. The different contexts inside one related context container are inside one rule combined with a logical *AND*. This is achieved using the XACML *All-Of* structure. Each policy has a last rule which always denies. That is why, there is always a determinate result, i.e. if no rule applies of no policy applies *Deny* is returned instead of *NotApplicable*. In order to make further usages of the intermediate model more easy, only the *Target* and not the *Condition* structure of the rule is used. Since all constraints can be expressed in the *Target*, the *Condition* is not used.

4.2.3. Mappings of Contexts to XACML Structures

Most of the contexts are mapped to matches. The table 4.1 show the diferent mappings from contexts to matches which are used to create XACML matches that represent the information stored in the respective contexts.

The internal state is mapped to a simple string comparison because the internal state must be equal to the one defined in the request. Roles, locations and organisations are mapped to regular expression matches due to their hierarchical structure which needs to be transformed to a logical *OR* inside the regular expression. The privacy level is also matched to a regex matching because lower privacy levels are also accepted. This means for example that if the privacy level is SECRET, the levels PRIVATE, RESTRICTED, PUBLIC and UNDEFINED are also allowed. The comparisons and shifts are transformed to respective match wrappers which create the needed XACML matches in order to successfully check the access control information.

Context	Match
InternalStateContext	StringComparisonMatch
RoleContext	respective subclass of RegexComparisonMatch
LocationContext	respective subclass of RegexComparisonMatch
OrganisationContext	respective subclass of RegexComparisonMatch
PrivacyLevelContext	respective subclass of RegexComparisonMatch
ComparisonContext	ComparisonMatch
ShiftContext	ShiftMatch

Table 4.1.: Mapping from Contexts to Matches.

The listing 4.1 shows an example of the organisation context which is mapped to the a match which uses regex matching (l. 1). The context contains an organisation *A* which has a sub organisation *ASub* (l. 3), that is why the test which needs to be done is a comparison with the two organisations combined with a logical *OR*.

```
 1  <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-regexp-match">
 2    <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
 3      (\QA\E)|(\QASub\E)
 4    </AttributeValue>
 5    <AttributeDesignator
 6      Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject"
 7      AttributeId="context:organisation"
 8      DataType="http://www.w3.org/2001/XMLSchema#string"
 9     MustBePresent="false"/>
10  </Match>
```

Listing 4.1: Example of an Organisation Match.

Furthermore, there are two contexts which are mapped to obligations. These are the *PrerequisiteContext* and the *ExtensionContext*. The generated obligations contain the respective information needed for an extension method call or a prerequisite operation signature which can be called by an extension of a PEP (see chapter 7.1). During the process of this thesis, these information are later used together with the attribute information saved in the rule matches for generating ensembles.

Listing 4.2 shows an example of an extension obligation for an extension context which defines the extension method with name "testExtensionMethod" (l. 5). The method should be called at the end of the checks. If there are more obligations, they are added in the *ObligationExpressions* structure (l. 1-14) as a further *ObligationExpression* structure (l. 2-13).

```
 1  <ObligationExpressions>
 2    <ObligationExpression ObligationId="obligation:extension" FulfillOn="Permit">
 3      <AttributeAssignmentExpression AttributeId="context:extension:testExtension">
 4        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
 5          testExtensionMethod
 6        </AttributeValue>
 7      </AttributeAssignmentExpression>
 8      <AttributeAssignmentExpression AttributeId="context:extension:isend">
 9        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#boolean">
10          true
11        </AttributeValue>
12      </AttributeAssignmentExpression>
13    </ObligationExpression>
14  </ObligationExpressions>
```

Listing 4.2: Example of an Extension Obligation.

4.2.4. Running Example

The two characteristics of the running example model are transformed to two XACML rules. The contexts are transformed to matches like explained above. However, the *PrerequisiteContext* is transformed to an obligation which shall be fulfilled by a PEP. In our case the obligation is considered during ensemble generation. The whole running example XACML policy set file can be found in the appendix of this thesis (see Section A.1).

4.3. Generating Ensembles from the Intermediate Model

The third contribution of this thesis is the generation of access control artefacts in the form of ensembles.

4.3.1. General Information concerning the Generation

In order to achieve this, the software which was implemented in the process of this thesis loads the intermediate model saved in an XACML policy set file. This file was previously created by the generation software which transforms instances of the dynamic context model to XACML policy set files. Due to the structure of the intermediate model, the generated ensembles are also described in a certain way which is similar to the structure defined in the intermediate model. The composition of the ensemble system is described in detail in this section.

The different policies in the XACML policy set are mapped to respective ensembles, which is logical because a policy represents an action in our case. Another important issue is the generation of the different kinds of components which can be used as parts of an ensemble. In order to maintain the structure of the intermediate model there are two different kinds of components *Subject*s and *Resource*s. They represent the respective XACML attribute categories. The contained attributes are added as attributes for the respective subclass of *Component*. It is also important to notice that only the contained attributes are added, other ones which do not exist in the whole policy set are not added. See also Section 2.4 for the general structure of an ensemble system.

The subject and resource attributes are checked with match functions which are equivalent to the ones defined in the respective *Match*. The environmental attributes, i.e. the attributes used for checking the shift time against the *now* time, are considered together with the shift match in the subject checks. Furthermore, each resource has a reference to an access subject which is checked against the subject attributes in the rule which is being transformed. This is done in order to keep the logical combination of *AND* inside one rule. For the same reason, the subject checks test if all resources match the resource attributes of the respective rule. It follows that one should only use one access subject in one scenario at a time. However, subjects with matching attributes are also found in the process of determining the access control decision. Since it is possible to define several scenarios and solving the system with different accessing subjects at the same time is rather a seldom used special case, the ensemble system can still be used for several use cases.

All policies in the policy set are added as sub ensembles of one root-ensemble which represents the whole policy set. The access control decision can then be inspected in the scenario definition. This definition is also added during the transformation but only as example code which needs to be adapted in order to match the scenario one wants to consider.

4.3.2. Running Example

Listing 4.3 shows the generated ensemble for the read log action of the running example. The subject checks inside the resources checks start with a null check and continue with the checks defined in the respective rule (l. 6-10). Thereafter, the internal state is checked against “OK” (l. 11). This (l. 6-11) and the analogue code in the subject checks (l. 23-28) is the transformation of the first rule which defines the situation when the machine is not defective. The situation of the machine being defective is represented by the second rule (l. 12-17, l. 29-33). It is also important to notice the call to the `prereq_sendReport` method (l. 24). This method can later be used in order to actually fulfil the prerequisite of sending a report. The content of this method must be added by hand because the generator does not know about the underlying system and can thus not fulfil the prerequisite. The whole running example ensemble system including one scenario definition can be found in the appendix of this thesis (see Section A.1).

```
object readLog extends Ensemble {
  val allowedResources =
    components
    .select[Resource]
    .filter(x => (
        (x.accessSubject != null
          && ((x.accessSubject.organisation != null
            && x.accessSubject.organisation.matches("(\\QA\\E)"))
          && (x.accessSubject.role != null
            && x.accessSubject.role.matches("(\\QWorker\\E)")))
        && (x.internalstate == "OK"))
      ||(x.accessSubject != null
          && ((x.accessSubject.organisation != null
            && x.accessSubject.organisation.matches("(\\QB\\E)"))
          && (x.accessSubject.role != null
            && x.accessSubject.role.matches("(\\QRepairperson\\E)")))
        && (x.internalstate == "DEFECTIVE"))))
    .map[Resource](x => x.getClass().cast(x))
  val allowedSubjects =
    components
    .select[Subject]
    .filter(x => (
        (allowedResources.forall(y => ((y.internalstate == "OK")))
          && prereq_sendReport("sendReport")
          && (x.organisation != null
            && x.organisation.matches("(\\QA\\E)"))
          && (x.role != null
            && x.role.matches("(\\QWorker\\E)")))
      ||(allowedResources.forall(y => ((y.internalstate == "DEFECTIVE")))
        && (x.organisation != null
          && x.organisation.matches("(\\QB\\E)"))
        && (x.role != null
          && x.role.matches("(\\QRepairperson\\E)")))))
    .map[Subject](x => x.getClass().cast(x))

  allow(allowedSubjects, "readLog", allowedResources)
}
```

Listing 4.3: The Generated Ensemble representing the Read Log Action of the Running Example.

5. Implementation

In this chapter the implementation of the two parts of the policy generation is described. The transformation implementations are implemented as eclipse plugins.

5.1. Used Libraries

The programming language used for implementing the transformations is Java 11. The generated ensembles are written in Scala 2.12.

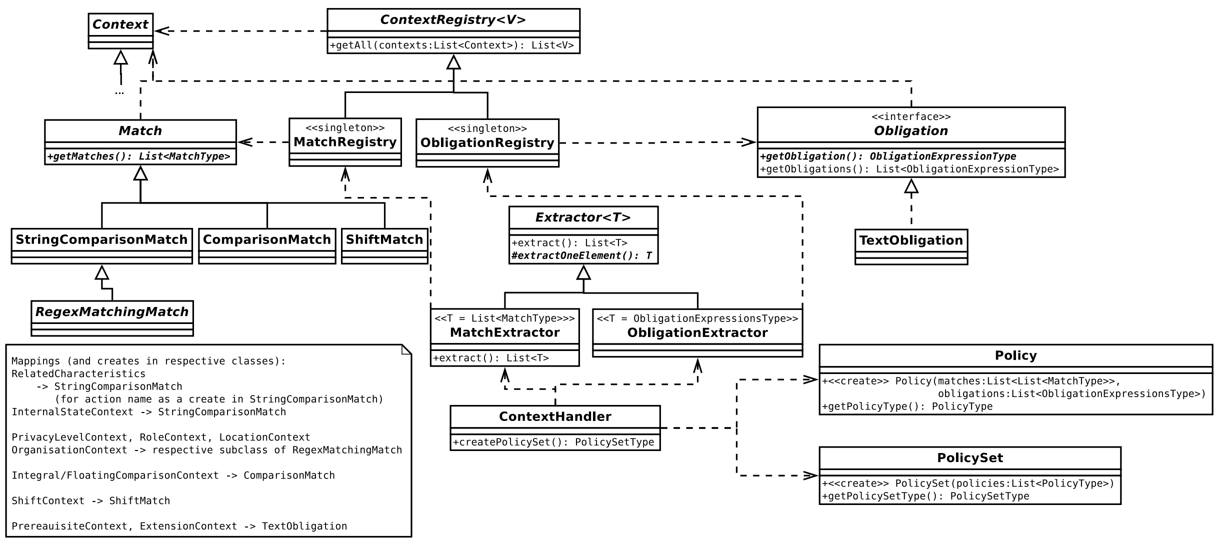
In the process of implementation there was a library used in order to ease the writing and loading of XACML policy set files and testing their correct function. This library used is the *AT&T XACML* framework (`https://github.com/att/XACML`) licensed under the *MIT* license. Another used library is the *Guava* library but it is already a prerequisite of the *AT&T XACML* framework.

The framework with which the generated ensembles can be used is defined in the paper [3]. In order to resolve a compatibility issue, a minor change in a print function in the library is done which does not alter the function of the solver.

5.2. XACML Policy Generation

In Figure 5.1 one can see an overview over the most important parts of the context model to XACML policy transformation software which was written in the process of this thesis. It shows the classes of the *generation* package and its sub-packages *matches* and *obligations*. The *execute* method of the *SampleHandler* which is the starting point of the plugin instantiates the *ContextHandler* and calls the *createPolicySet* method. It uses the *Match-* and *ObligationExtractor* in order to create XACML policies which are then combined to XACML policy sets via the respective wrapper classes *Policy* and *PolicySet*.

The respective extractors for matches and obligations extract the information contained in the context model instance with the help of two registries which define the mappings of contexts to either a sub-class of *Match* or *Obligation*. This infrastructure makes it possible to easily adapt the generation of the XACML intermediate model to future changes to the context metamodel. The mappings which are set at the moment can also be seen in the figure. One can also see that the *MatchExtractor* also overrides the *extract* method. This is done in order to be able to also handle actions which do not have any contexts related to them. Another important noteworthiness is that the *MatchExtractor* always adds the name of the action to the list of matches at index 0. This is done in order to be able to combine all policies in one policy set which represents the whole model instance.


Context
...
ContextRegistry<V>
+getAll(contexts:List<Context>): List<V>
Match
+getMatches(): List<MatchType>
<<singleton>>
MatchRegistry
<<singleton>>
ObligationRegistry
<<interface>>
Obligation
+getObligation(): ObligationExpressionType
+getObligations(): List<ObligationExpressionType>
StringComparisonMatch
ComparisonMatch
ShiftMatch
Extractor<T>
+extract(): List<T>
#extractOneElement(): T
TextObligation
RegexMatchingMatch
<<T = List<MatchType>>>
MatchExtractor
+extract(): List<T>
<<T = ObligationExpressionsType>>
ObligationExtractor
Policy
+<<create>> Policy(matches:List<List<MatchType>>,
obligations:List<ObligationExpressionsType>)
+getPolicyType(): PolicyType
ContextHandler
+createPolicySet(): PolicySetType
PolicySet
+<<create>> PolicySet(policies:List<PolicyType>)
+getPolicySetType(): PolicySetType

```
Mappings (and creates in respective classes):
RelatedCharacteristics
    -> StringComparisonMatch
       (for action name as a create in StringComparisonMatch)
InternalStateContext -> StringComparisonMatch

PrivacyLevelContext, RoleContext, LocationContext
OrganisationContext -> respective subclass of RegexMatchingMatch

Integral/FloatingComparisonContext -> ComparisonMatch

ShiftContext -> ShiftMatch

PrereauisiteContext, ExtensionContext -> TextObligation
```

Figure 5.1.: XACML Generation Implementation Overview.

In order to get more information about the concrete implementation of each class and its methods one can have a look at the JavaDoc documentation.

5.3. Ensembles Generation

The ensembles generation relies mainly on compounding of different parts of Scala code. That is why, there is a package with name "scala" which contains classes which represent small Scala code parts. These code parts can be easily assembled in the process of transformation of the XACML input policy set into the ensemble system to be created. Another important package is the "xacml" package which contains larger parts of the transformation. The *generation* package contains the other two packages and combines the policies to different ensembles in a complete ensemble system. In Figure 5.2, one can see the structure of the packages as described in this paragraph. Moreover, one can see the *CodePart* interface which represents a larger Scala code block which can be obtained by the "getCode" method. There are two implementations of this interface, the *PolicySetHandler* which creates the whole ensemble system with the help of creating different ensembles using the *PolicyCodePart.* It is also an implementation of the *CodePart* interface.

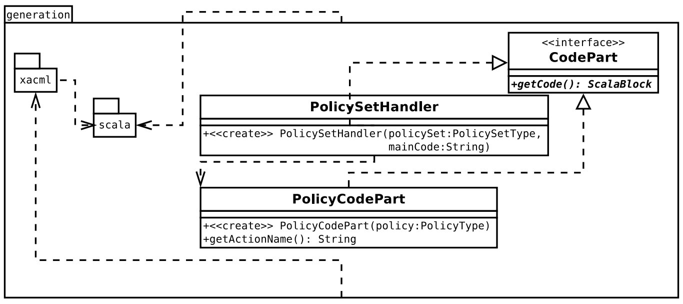


Figure 5.2.: Ensembles Generation Packages Overview.

In Figure 5.3, one can see the different Scala code parts. The *ValueDeclaration* and *ValueInitialisation* classes represent the respective Scala constructs, the *isOptional* flag in the constructor of the declaration is used to set whether the standard value (mostly `null`) should be set. The *ScalaClass* class represents the declaration of a class or singleton in Scala. The *MethodSignature* and *Call* classes are for the respective functions of method signatures and method calls in Scala. Moreover, the *ScalaBlock* represent the larger structure of a code block. Additionally, several blocks can be appended to an iterable structure. This makes it possible to save a whole ensemble system including the scenario definition in one object. The structure allows the iteration over string builders of the different appended blocks. This enables block-wise writing of the structure.

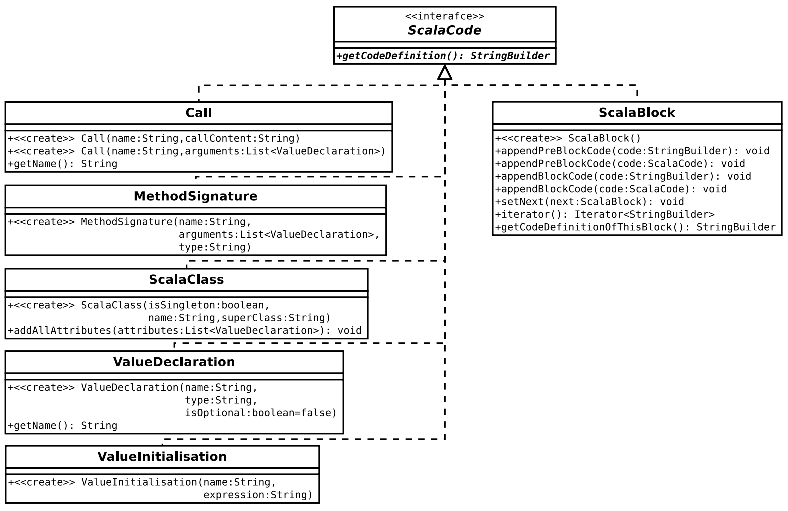

<<interafce>>
ScalaCode
+getCodeDefinition(): StringBuilder
Call
+<<create>> Call(name:String,callContent:String)
+<<create>> Call(name:String,arguments:List<ValueDeclaration>)
+getName(): String
ScalaBlock
+<<create>> ScalaBlock()
+appendPreBlockCode(code:StringBuilder): void
+appendPreBlockCode(code:ScalaCode): void
+appendBlockCode(code:StringBuilder): void
+appendBlockCode(code:ScalaCode): void
+setNext(next:ScalaBlock): void
+iterator(): Iterator<StringBuilder>
+getCodeDefinitionOfThisBlock(): StringBuilder
MethodSignature
+<<create>> MethodSignature(name:String, arguments:List<ValueDeclaration>, type:String)
ScalaClass
+<<create>> ScalaClass(isSingleton:boolean, name:String,superClass:String)
+addAllAttributes(attributes:List<ValueDeclaration>): void
ValueDeclaration
+<<create>> ValueDeclaration(name:String, type:String, isOptional:boolean=false)
+getName(): String
ValueInitialisation
+<<create>> ValueInitialisation(name:String, expression:String)

Figure 5.3.: Ensembles Generation “Scala” Package.

In Figure 5.4 one can see the structure of the “xacml” package of the ensemble generation software. It contains classes for the extraction of attributes and obligations from the XACML policy set. The extraction results consist of code for checking the resources’ and subjects’ attributes against the attributes defined in the given XACML matches. In order to render the extraction possible there are three enumerations. The *Category* enum represents the three

XACML attribute categories resource, subject and environment. The *Function* enum represents the different used XACML match functions like for example "string-regepx-match". This enum also has a getter for the equivalent Scala code to the match function. In the end, the *Attribute* enum contains the different used XACML attributes. In order to be able to transform the XACML obligations into respective extension methods there are interfaces for obligation lists and the particular obligations. Furthermore, there are implementations for the text obligations. The obligations are extracted together with the normal attributes inside the "extract" method of the *RuleAttributeExtractor* for each XACML rule and category. This extractor is used by the *AttributeExtractor* which combines the extraction results of the different rules by connecting them with a logical *OR*. The sets which are extracted can be used to allow the *ComponentCode* to only contain attributes for the existing XACML attributes and obligations.

The extractions done in this package by the *AttributeExtractor* are then used by the *PolicyCodePart* which creates the different ensembles which represent the different policies. In the end, the *PolicySetHandler* uses the *ComponentCode* and the obligations' method block definitions which are written into the ensemble system at model level.

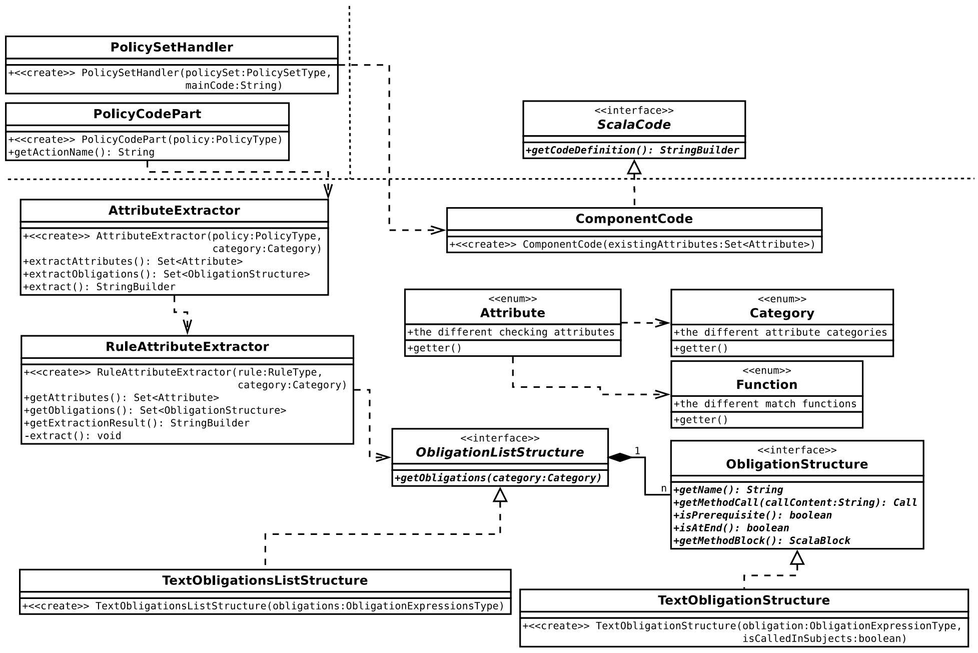

PolicySetHandler
+<<create>> PolicySetHandler(policySet:PolicySetType, mainCode:String)
PolicyCodePart
+<<create>> PolicyCodePart(policy:PolicyType)
+getActionName(): String
<<interface>>
ScalaCode
+getCodeDefinition(): StringBuilder
AttributeExtractor
+<<create>> AttributeExtractor(policy:PolicyType, category:Category)
+extractAttributes(): Set<Attribute>
+extractObligations(): Set<ObligationStructure>
+extract(): StringBuilder
ComponentCode
+<<create>> ComponentCode(existingAttributes:Set<Attribute>)
<<enum>>
Attribute
+the different checking attributes
+getter()
<<enum>>
Category
+the different attribute categories
+getter()
RuleAttributeExtractor
+<<create>> RuleAttributeExtractor(rule:RuleType, category:Category)
+getAttributes(): Set<Attribute>
+getObligations(): Set<ObligationStructure>
+getExtractionResult(): StringBuilder
-extract(): void
<<enum>>
Function
+the different match functions
+getter()
<<interface>>
ObligationListStructure
+getObligations(category:Category)
1
n
<<interface>>
ObligationStructure
+getName(): String
+getMethodCall(callContent:String): Call
+isPrerequisite(): boolean
+isAtEnd(): boolean
+getMethodBlock(): ScalaBlock
TextObligationsListStructure
+<<create>> TextObligationsListStructure(obligations:ObligationExpressionsType)
TextObligationStructure
+<<create>> TextObligationStructure(obligation:ObligationExpressionType, isCalledInSubjects:boolean)

Figure 5.4.: Ensembles Generation "xacml" Package Overview.

5.4. Software Engineering - Tests and Version Control

In this section issues concerning software engineering are briefly explained. This section describes the used software engineering concepts which were considered during implementation.

There are unit and scenario tests which can be used to check whether the generation works as expected. The unit tests check parts of the implementation which can be easily tested as units. The scenario tests test the more complex interaction of transforming given input models to respective access control policies. This is a XACML policy set file in the case of the PCM-2-XACML transformation and a Scala file representing an ensemble system for security in the case of the XACML-2-Ensembles transformation. The scenario tests can be seen as integration tests. Nevertheless, in case of PCM-2-XACML they are realised as JUnit DynamicTests in order to make automatic testing possible. The XACML-2-Ensembles scenario tests are not automatised because the generated code would have to be compiled. There is a semi-automatised test structure for this transformation.

In the process of implementation, the version control system `git` was used. The table 5.1 shows all the used `GitHub` repositories. The transformation repositories always contain the related tests and evaluation scenario tests.

Git Address	Description
`https://github.com/Lolalol97/PCM-Dynamic-Context-MetaModel.git`	PCM context metamodel
`https://github.com/Lolalol97/UseCasesTechnicalReport.git`	PCM context model instances
`https://github.com/Lolalol97/PCM-2-XACML.git`	PCM-2-XACML transformation
`https://github.com/Lolalol97/XACML-2-Ensembles.git`	XACML-2-Ensembles transformation
`https://github.com/Lolalol97/ensembleTester_tcoof-security-ecsa.git`	Tester for generated ensembles

Table 5.1.: The used GitHub repositories.

6. Evaluation

In order to evaluate the thesis a GQM-Plan [22] will be used. In this chapter, the evaluation goals, questions and metrics for this thesis will be explained. Thereafter, the results of the evaluation are presented and discussed.

6.1. Evaluation GQM Plan

The Evaluation considers the correct functioning of the automatic policy generation described in the chapters 4 and 5. In order to evaluate this, three evaluation goals are considered:

- **EG-1:** Evaluate the feasibility of the metamodel and policy generation for its planned purpose.
- **EG-2:** Evaluate the accuracy of the generated access control policies, i.e. whether the access control decisions are accurate.
- **EG-3:** Evaluate the scalability of the implemented policy generation.

6.1.1. Design of Evaluation of Feasibility

In order to evaluate feasibility (**EG-1**), there will be a description of the planned use by describing possible use cases. Afterwards, it will be evaluated whether the context metamodel and policy generation is usable in the given cases. This design results in the evaluation questions:

- **(Q-1.1)** Is the metamodel usable for modelling the given use case?
- **(Q-1.2)** Is the XACML policy generation usable for transforming the given use case to a XACML policy set?
- **(Q-1.3)** Is the ensemble policy generation usable for transforming the XACML policy set representing the use case to an ensemble system.

(M-1.1) As a metric for the evaluation of feasibility one can use the number of successful represented use cases divided by the total number of use cases.

6.1.2. Design of Evaluation of Accuracy

In order to evaluate accuracy (**EG-2**), there will be a testing of access control decisions for several use cases. The use cases which are checked for accurate access control decision are the same as the ones used for evaluating feasibility. The following evaluation questions are raised:

- **(Q-2.1)** Is the generated XACML policy set accurate, i.e. it results in a correct access control decision when requested properly?
- **(Q-2.2)** Is the generated ensemble system accurate, i.e. it results in a correct access control decision when requested properly?

(M-2.1) As a metric precision (PR) an recall (RC) will be used. It consists of the formulas

$$PR = \frac{TP}{TP+FP}$$
$$RC = \frac{TP}{RPA}$$

where TP is the number of true positives, FP is the number of false positives and RPA the total number of expected positives [28]. In our case, a true positive is a correct access granting and a false positive consists of a incorrect given access.

6.1.3. Design of Evaluation of Scalabilty

Scalability is an important factor because policy generation should be fast so that a fast roll-out of new access rules via XACML policy sets or ensemble systems can be done. This is necessary in a fast changing dynamic Industry 4.0 environment.

In order to evaluate scalability (**EG-3**), the following questions are considered:

- **(Q-3.1)** How does the generation of XACML policy sets scale with an increasing number of context characteristics container?
- **(Q-3.2)** How does the generation of ensemble systems scale with an increasing number of XACML rules?
- **(Q-3.3)** How does the generation of XACML policy sets scale with an increasing number of contexts in a rule?
- **(Q-3.4)** How does the generation of ensemble systems scale with an increasing number of XACML matches in a rule?

(M-3.1) As a metric the time needed for policy generation in comparison to the number of input contexts is used.

6.2. Evaluation Setups

In this section the setups to achieve the beforementioned evaluation goals are described.

6.2.1. Description of Use Cases

In this subsection, the different use cases used for the evaluation are briefly described. In order to render the evaluation more easy, only the significant parts for the confidentiality analysis of the use case are modelled. The use case models can be found in the "PCM context model instances" git (see Section 5.4).

The use case **UC0** consists of a telemaintenance scenario. A service technician of organisation A should do telemaintenance of a machine of organisation B. The access should only be given when the machine is in state SERVICE_MODE, the time is between 15 and 17 o'clock and the service technician has done an adequate security check beforehand [30]. Especially, the role, internal state and time are significant for this use case.

The use case **UC1** consists of a worker in the production hall of organisation A which informs the foreman about a problem. Thus, the worker sends a report. Thereafter, the foreman receives the report. Then, the foreman sees the schedule of the workers in order to address the problem in production. Especially, the role, organisation, time and location attributes are important for this use case [5]. However, the organisation and location attributes are rather considered in **UC4**. The regarded shift is the early production shift which takes place from 6 to 14 o'clock.

The use case **UC2** consists of a worker in the factory of organisation B which informs the foreman about a faulty item. Thereafter, the foreman informs the external supplier A. Especially, the role, time and location attributes are important for this use case [5].

The use case **UC3** consists of a scenario concerning a board which reacts on a low total first time yield. This is an often analysed important quantity. The part of the use case considered here consists of the board informing two companies with the lowest score [5]. It is assumed that the two companies are A and B.

The use case **UC4** consists of a worker entering the production hall 30 minutes before the shift starts [5]. The regarded shift is the early production shift which takes place from 6 to 14 o'clock. In order to model that the entry should be allowed 30 minutes before the actual shift starts, the shift is modelled form 05:30:00Z to 14:00:00Z.

The use case **UC5** is a superordinate use case of **UC4** and consists of a whole workflow of a shift from the arrival of the foreman via the arrival of workers, the leaving of workers and finally the leaving of the foreman 45 minutes after the end of the shift [5]. Furthermore, the evaluation of this use case considers an alternative model for checking the time using an *ExtensionContext*. Since this extension is only implemented in the generated ensemble system and not in the XACML, this use case is not considered in the XACML accuracy evaluation. The reason for this is that the default implementation of the PEP does not consider obligations (see Section 7.1).

The use case **UC6** (UC-Combined model) consists of a combined use case of the use cases **UC1**, **UC2** and an extension which considers quality assurance (QA). In this scenario, the additional actions are in fact communications between QA A and QA B using reports. For these reports there are certain restrictions concerning confidentiality. For example, QA of A should only be allowed to inform QA of B when the privacy level of the report has a certain privacy level [25]. The model considers the PROTECTED privacy level and in the process of accuracy evaluation this or a less restrictive privacy level (PUBLIC) is checked.

The use case **UC7** (UC-Running model) consists of the running example which is only considered additionally. Moreover, there are two further models: `UC-Test` which tests all contexts and `UC-Scale` which contains exactly one *ShiftContext*. These two models are used for scalability evaluation (see Subsection 6.2.4).

6.2.2. Evaluation Setup for Feasibility

This subsection describes the setup for the evaluation goal of feasibility (**EG-1**). The feasibility evaluation should show the usability of the meta model and the transformations for the use cases which describe possible future usages.

The feasibility is evaluated using the metric (**M-1.1**) described in the respective design section (see Section 6.1.1). Due to pressure of time, the feasibility is only evaluated by myself and not by a known security expert. In order to answer the question of model feasibility (**Q-1.1**), the possibility of modelling the given use case is considered. The evaluation of feasibility also considers the other two questions (**Q-1.2**, **Q-1.3**) by evaluating whether it is possible to automatically transform the given use case model to an XACML policy set. Thereafter, it is determined whether it is possible to transform the generated XACML policy set to an ensemble system. These considerations are done for all described use case. The evaluation of an accurate access control decision is not considered in this evaluation goal but rather in the accuracy evaluation goal. The models for all the use cases can be found in the model instances Git (see Section 5.4).

6.2.3. Evaluation Setup for Accuracy

This subsection describes the setup for the evaluation goal of accuracy (**EG-2**). The accuracy evaluation should show the accuracy of the access control decision of the generated access control artefacts when asked with a adequate request.

The accuracy is evaluated using the metric (**M-2.1**) described in the respective design section (see Section 6.1.2). In order to answer the question of accurate XACML policy set generation (**Q-2.1**), the generated XACML policy set file from the use case model instance is tested in a dynamic test with adequate requests which are also defined in the respective use case description. However, larger use cases are only considered partially in XACML accuracy evaluation but so as to indicate the feasibility and accuracy of certain requirements. The generated XACML policy set files are then transformed to an ensemble system and tested with an adequate scenario definition (**Q-2.2**). Moreover, for models which use prerequisite or extension contexts there is also manually added extension or prerequisite code in the generated ensemble system. The setup for the testing of the accuracy can be found in the respective implementation Gits (see Section 5.4).

6.2.4. Evaluation Setup for Scalability

This subsection describes the setup for the evaluation goal of scalability (**EG-3**). The time measurements for all the evaluation goals are done in an automatic test system. Since it is not part of the scope of this thesis, the actual printing library call is not considered in the measurement.

The system used for the evaluation of the scalability of the transformations is a Aspire E15 E5-576-53AG with the following specifications:

- Intel Core i5-7200U with 2.5GHz
- 12 GB DDR4 RAM
- Linux Ubuntu 18.04.3 LTS
- Java 11

In order to generate the input model for an increasing number of rules (**Q-3.1**), a characteristic container from a base model is copied several times. This approach simulates the existence of a model with many rules. The considered copying numbers are 1, 10, 100, 1000, 10000, 100000 and 500000. As base models the `UC-Scale` and the `UC-Test` are used. The first one contains only one *ShiftContext* whereas the second one contains all the contexts once. So one can be sure that the scalability is given for all possible contexts.

The other test which is done in the automatic test system is the consideration of an increasing number of contexts inside one rule (**Q-3.3**). This test is only done with the `UC-Scale` base model because the *ShiftContext* is the context which produces the most XACML matches. Therefore, it is ensured that the transformation of all other contexts scale in the same way because the transformation of all contexts is built similarly.

The tests for the respective questions concerning the generation of ensemble systems (**Q-3.2**, **Q-3.4**) use the resulting policy sets from the beforementioned scalability tests as input. Thereafter, the transformation into an ensemble system is measured.

The actual results are evaluated by doing a warm up first and then doing 10 iterations of the beforementioned tests. The average of the results are then depicted in a diagram to determine whether linear runtime behaviour can be seen.

The automatic test systems can be found in the implementation Gits of the respective transformations inside the packages [1].

[1] `.ensemblesgeneration.tests.evaluation` and `.xacmlpolicygeneration.tests.evaluation`

6.3. Evaluation Results

In this section the results of the evaluation are presented and discussed.

6.3.1. Results of Evaluation of Feasibility

All the beforementioned use cases were modelled in the process of the feasibility evaluation. There were no larger problems implementing the use cases using the context meta model.

Table 6.1 shows the result of the feasibility evaluation. The first row summarizes whether the modelling was possible for the different models representing the use cases (**Q-1.1**). The second row summarizes whether the transformation to XACML was possible (**Q-1.2**) and the third row sums up if the transformation to an ensemble system was possible (**Q-1.3**).

Question	Number of use cases	Number of successfully represented use cases	Metric Result (**M-1.1**)
Q-1.1	7	7	1
Q-1.2	7	7	1
Q-1.3	7	7	1

Table 6.1.: The feasability evaluation result.

A limitation of the evaluation of feasibility consist of the fact that no well-known security expert was asked due to time reasons. Another limitation is that usability and integration in a real Industry 4.0 environment was not considered in the process of evaluation and can hence be considered future work (see also Section 7.3). Moreover, the easiness of creation of models was also not evaluated and could also be considered future work.

To summarize, the feasibility evaluation indicated that the evaluation questions could be answered to an adequate degree. Nevertheless, more evaluation and extensions to the implementations should be done in the future.

6.3.2. Results of Evaluation of Accuracy

This subsection describes the results of the accuracy evaluation. Due to time and redundancy issues, the XACML policy sets are only tested partially (see setup section 6.2.3). There are overall 29 relevant requests for the different use cases. Table 6.2 shows the result for the different use cases and the overall result (**M-2.1**).

Use case / Question	TP	RPA	FP	Result PR	Result RC
UC0	1	1	0	1	1
UC1	3	3	0	1	1
UC2	2	2	0	1	1
UC3	3	3	0	1	1
UC4	1	1	0	1	1
UC6	2	2	0	1	1
Q-2.1	12	12	0	1	1

Table 6.2.: The accuracy evaluation result for the XACML generation.

There is also an accuracy evaluation of the XACML-2-Ensembles transformation. Each use case model was tested with a scenario representing a *permit* decision and and one for a *deny* decision. This means there are overall 14 Scala scenario definitions. Table 6.3 shows the result for the different use cases and the overall result (**M-2.1**).

Use case / Question	TP	RPA	FP	Result PR	Result RC
UC0	1	1	0	1	1
UC1	1	1	0	1	1
UC2	1	1	0	1	1
UC3	1	1	0	1	1
UC4	1	1	0	1	1
UC5	1	1	0	1	1
UC6	1	1	0	1	1
Q-2.2	7	7	0	1	1

Table 6.3.: The accuracy evaluation result for the Ensembles generation.

A limitation of the accuracy evaluation is that modelling of requests and scenarios are not completely unambiguous but it was always tried to model requests and scenarios the best way possible. Another important limitation is that only a partial set of all possible combinations of requests can be done so it is not strictly proven that the concept or the implementation is correct. However, the accuracy evaluation indicated an adequately accurate representation of the use cases. Like already considered in the discussion of the feasibility evaluation there is also the problem of setting up the system in a real Industry 4.0 environment. Hence, this can also be considered future work.

To sum up, the accuracy evaluation indicated that the implemented software generates adequately accurate access control policies for the considered use cases.

6.3.3. Results of Evaluation of Scalability

In this subsection the results of the scalability evaluation are presented and discussed. Figure 6.1 illustrates that the execution time of XACML generation indicates linear behaviour in relation to the count of rules (**Q-3.1**). The blue points depict the averaged results of the evaluation test structure for the `UC-Scale` base model whereas the red points are the averaged results for the `UC-Test` base model (see also subsection 6.2.4). The green and the black lines are respective linear references for the given data points to indicate that the execution time behaviour is linear for input numbers of 1000 or more.

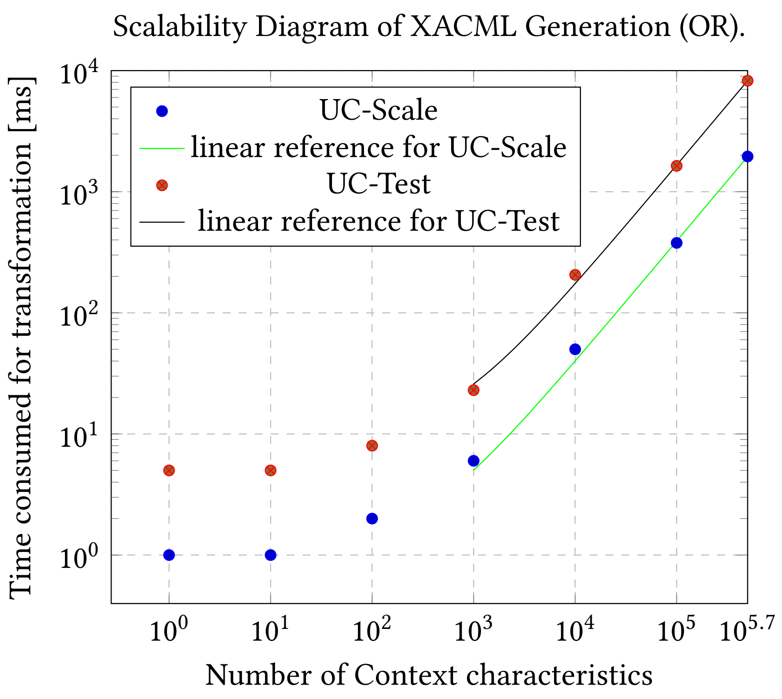


Figure 6.1.: Scalability Diagram of the PCM-2-XACML Transformation (OR) (Q-3.1).

Due to limitations of the used hardware for evaluation, the 500000 copies test is not done in the scalability tests for the generating of ensemble systems. Figure 6.2 illustrates that the ensembles generation indicates linear runtime behaviour in relation to the count of rules (**Q-3.2**). The blue points depict the averaged results of the evaluation test structure for the respective generated `UC-Scale` XACML policy set file whereas the red points are the averaged results for the `UC-Test` XACML policy set file. The green and the black lines are respective linear references for the given data points to indicate that the execution time behaviour is linear for input numbers of 1000 or more.

Figure 6.3 illustrates that the execution time of XACML generation indicates linear behaviour in relation to the count of contexts in one rule (**Q-3.3**). The blue points depict the averaged results of the evaluation test structure for the `UC-Scale` base model. The green line is a linear reference for the given data points to indicate that the execution time behaviour is linear for input numbers of 10000 or more.

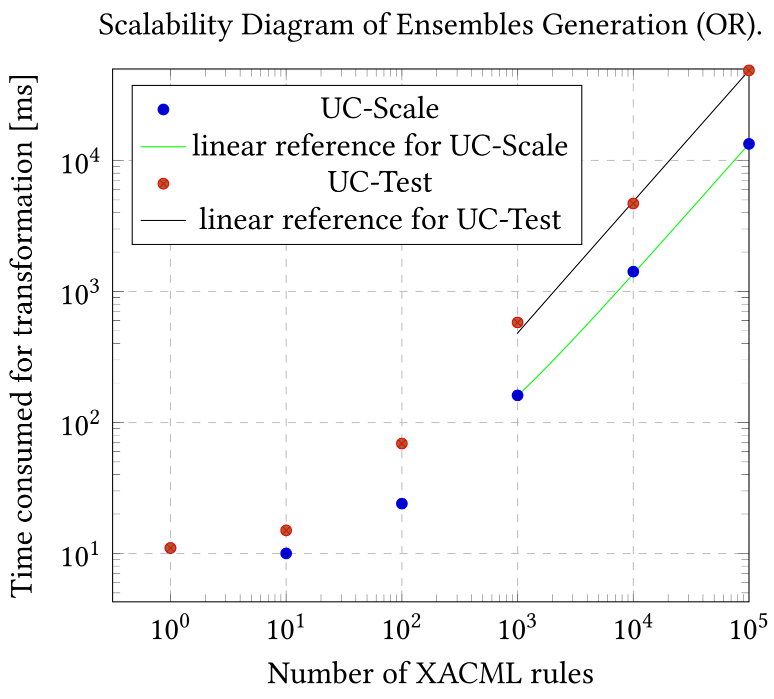


Figure 6.2.: Scalability Diagram of the XACML-2-Ensembles Transformation (OR) (Q-3.2).

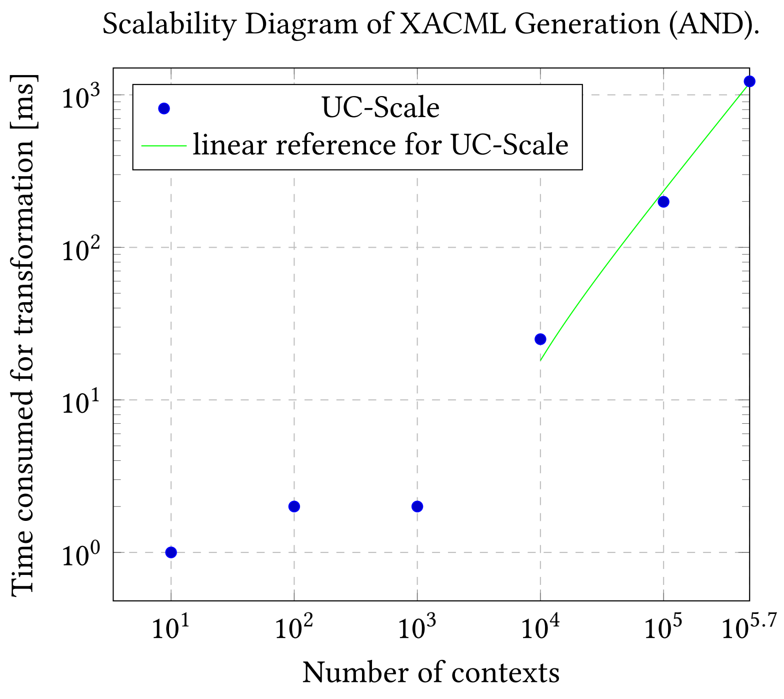


Figure 6.3.: Scalability Diagram of the PCM-2-XACML Transformation (AND) (Q-3.3).

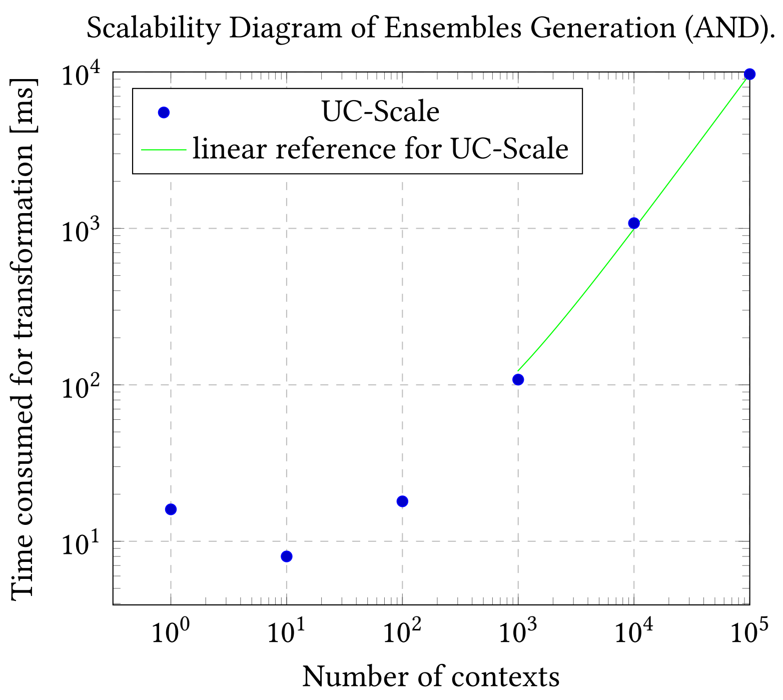


Figure 6.4.: Scalability Diagram of the XACML-2-Ensembles Transformation (AND) (Q-3.4).

Figure 6.4 illustrates that the execution time of Ensembles generation indicates linear behaviour in relation to the count of contexts in one rule (**Q-3.4**). The blue points depict the averaged results of the evaluation test structure for the `UC-Scale` base model. The green line is a linear reference for the given data points to indicate that the execution time behaviour is linear for input numbers of 1000 or more.

In this paragraph the limitations of the scalability evaluation are discussed. One limitation is that memory scalability was not considered because it is difficult to measure this exactly. Moreover, deep hierarchies of roles, organisations or locations were also not considered but as long as the hierarchy is not too deep to make the implementation run out of stack memory, which is very unlikely.

The scalability evaluation indicated that the execution time of the implemented transformations is linear for relevant numbers of contexts. Furthermore, it can be seen that the absolute runtime (maximum ca. 50 seconds for 100000 XACML rules) is also acceptable considering that the generation is done at design-time. To conclude, the scalability of both implemented transformations is ensured for its planned purpose.

7. Conclusion

In this chapter, the findings of this thesis are concluded. Moreover, the limitations of the approach and possible future work is briefly described.

7.1. Limitations

During the process of creating this thesis, some limitations were encountered which are briefly explained in this section. First, the PEP does not evaluate obligations (see Section 4.2.3) which means that evaluating prerequisite and extension contexts is not yet possible in XACML. However, it can be considered in the ensemble system. Second, the considered access subject must be explicitly defined in the resource (see Section 4.3.1). Third, the ensembles generation only considers the transformation of XACML policy sets generated by the XACML generation. Fourth, the XACML policy sets and the ensembles are not optimized.

7.2. Summary

To sum up, the thesis presented an extension to the context meta model (see Section 4.1) as a first contribution, a transformation of model instances to the attribute based XACML policy sets (see Section 4.2) and a transformation of these policy sets to an ensemble based component system (see Section 4.3). The model extension mainly considered extensibility at a later design time. The transformations from the model to XACML makes it possible to generate access control artefacts of a wide-spread security standard. Furthermore, the transformation to an ensemble system considers the generation of an EBCS Scala file from the XACML policy set file which was created beforehand by the PCM-2-XACML program.

7.3. Future Work

This section describes some ideas for future work concerning the topic of this thesis. First, there could be an implementation of a PEP extension which considers obligations. Second, the ensemble generation could be extended to a more general transformation which also considers XACML policy sets which are not generated bei PCM-2-XACML. Third, further and larger usability tests for the meta model and the transformations and improvements can be done in the future. Fourth, a better integration into eclipse could also be considered like for example an improved settings management which pays more attention to more detailed settings.

Bibliography

[1] Rima Al Ali et al. “Dynamic Security Specification Through Autonomic Component Ensembles”. In: *Leveraging Applications of Formal Methods, Verification and Validation. Distributed Systems.* Ed. by Tiziana Margaria and Bernhard Steffen. Cham: Springer International Publishing, 2018, pp. 172–185. ISBN: 978-3-030-03424-5.

[2] Muhammad Alam, Ruth Breu, and Michael Hafner. “Model-Driven Security Engineering for Trust Management in SECTET”. In: *Journal of Software* 2.1 (Jan. 2007). DOI: `10.4304/jsw.2.1.47-59`.

[3] Rima Al-Ali et al. “Dynamic Security Rules for Legacy Systems”. In: *First Workshop on Systems, Architectures, and Solutions for Industry 4.0, Paris, France.* 2019.

[4] Rima Al-Ali et al. “Modeling of dynamic trust contracts for industry 4.0 systems”. In: Sept. 2018, pp. 1–4. DOI: `10.1145/3241403.3241450`.

[5] Rima Al-Ali et al. *Use Cases in Dataflow-Based Privacy and Trust Modeling and Analysis in Industry 4.0 Systems.* Faculty of Informatics, KIT, DE, Report, 2018.

[6] Steffen Becker, Heiko Koziolek, and Ralf Reussner. “Model-Based Performance Prediction with the Palladio Component Model”. In: *Proceedings of the 6th International Workshop on Software and Performance.* WOSP ’07. Buenes Aires, Argentina: ACM, 2007, pp. 54–65. ISBN: 1-59593-297-6. DOI: `10.1145/1216993.1217006`. URL: `http://doi.acm.org/10.1145/1216993.1217006`.

[7] Steffen Becker, Heiko Koziolek, and Ralf Reussner. “The Palladio Component Model for Model-driven Performance Prediction”. In: *Journal of Systems and Software* 82 (2009), pp. 3–22. DOI: `10.1016/j.jss.2008.03.066`. URL: `http://dx.doi.org/10.1016/j.jss.2008.03.066`.

[8] Steffen Becker et al. *Modeling and Simulating Software Architectures - The Palladio Approach.* Oct. 2016. ISBN: 9780262034760.

[9] Iman Ben Abdelkrim et al. “Coalition-OrBAC: An Agent-Based Access Control Model for Dynamic Coalitions”. In: *Trends and Advances in Information Systems and Technologies.* Ed. by Álvaro Rocha et al. Cham: Springer International Publishing, 2018, pp. 1060–1070. ISBN: 978-3-319-77703-0.

[10] Antonia Bertolino et al. “A Toolchain for Designing and Testing Access Control Policies”. In: *Engineering Secure Future Internet Services and Systems: Current Research.* Ed. by Maritta Heisel et al. Cham: Springer International Publishing, 2014, pp. 266–286. ISBN: 978-3-319-07452-8. DOI: `10.1007/978-3-319-07452-8_11`. URL: `https://doi.org/10.1007/978-3-319-07452-8_11`.

[11] Nicolas Boltz. "Representing Dynamic Context Information in Dataflow Based Privacy Analysis". Karlsruher Institut für Technologie, Fakultät für Informatik, Postfach 6980, 76128 Karlsruhe: KIT, Nov. 2018.

[12] F. Cuppens and A. Miege. "Modelling contexts in the Or-BAC model". In: *19th Annual Computer Security Applications Conference, 2003. Proceedings.* Dec. 2003, pp. 416–425. DOI: 10.1109/CSAC.2003.1254346.

[13] Sabrina De Capitani di Vimercati and Pierangela Samarati. "Mandatory Access Control Policy (MAC)". In: *Encyclopedia of Cryptography and Security.* Ed. by Henk C. A. van Tilborg and Sushil Jajodia. Boston, MA: Springer US, 2011, pp. 758–758. ISBN: 978-1-4419-5906-5. DOI: 10.1007/978-1-4419-5906-5_822. URL: https://doi.org/10.1007/978-1-4419-5906-5_822.

[14] David F. Ferraiolo, Janet A. Cugini, and D. Richard Kuhn. "Role-Based Access Control (RBAC) : Features and Motivations". In: *Proceedings of 11th annual computer security application conference.* 1995, pp. 241–248.

[15] Rolf Hennicker and Annabelle Klarl. "Foundations for Ensemble Modeling – The Helena Approach". In: *Specification, Algebra, and Software: Essays Dedicated to Kokichi Futatsugi.* Ed. by Shusaku Iida, José Meseguer, and Kazuhiro Ogata. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 359–381. ISBN: 978-3-642-54624-2. DOI: 10.1007/978-3-642-54624-2_18. URL: https://doi.org/10.1007/978-3-642-54624-2_18.

[16] Chung Tong Hu, David F. Ferraiolo, and David R. Kuhn. *Guide to Attribute Based Access Control (ABAC) Definition and Considerations.* NIST, US, NIST SP, 2019. URL: https://doi.org/10.6028/NIST.SP.800-162.

[17] Jan Jürjens. "UMLsec: Extending UML for Secure Systems Development". In: *UML 2002 — The Unified Modeling Language.* Ed. by Jean-Marc Jézéquel, Heinrich Hussmann, and Stephen Cook. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 412–425. ISBN: 978-3-540-45800-5.

[18] A. A. E. Kalam et al. "Organization based access control". In: *Proceedings POLICY 2003. IEEE 4th International Workshop on Policies for Distributed Systems and Networks.* June 2003, pp. 120–131. DOI: 10.1109/POLICY.2003.1206966.

[19] Torsten Lodderstedt, David Basin, and Jürgen Doser. "SecureUML: A UML-Based Modeling Language for Model-Driven Security". In: *UML 2002 — The Unified Modeling Language.* Ed. by Jean-Marc Jézéquel, Heinrich Hussmann, and Stephen Cook. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 426–441. ISBN: 978-3-540-45800-5.

[20] M. Munier, V. Lalanne, and M. Ricarde. "Self-Protecting Documents for Cloud Storage Security". In: *2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications.* June 2012, pp. 1231–1238. DOI: 10.1109/TrustCom.2012.261.

[21] Phu H. Nguyen et al. "An extensive systematic review on the Model-Driven Development of secure systems". In: *Information and Software Technology* 68 (2015), pp. 62–81. DOI: 10.1016/j.infsof.2015.08.006.

[22] Victor R. Basili, Gianluigi Caldiera, and Dieter Rombach. "The goal question metric approach". In: *Encyclopedia of Software Engineering* 1 (Jan. 1994).

[23] E. Schmieders, A. Metzger, and K. Pohl. "Architectural Runtime Models for Privacy Checks of Cloud Applications". In: *2015 IEEE/ACM 7th International Workshop on Principles of Engineering Service-Oriented and Cloud Systems.* May 2015, pp. 17–23. DOI: `10.1109/PESOS.2015.11`.

[24] Stephan Seifermann, Robert Heinrich, and Ralf Reussner. "Data-Driven Software Architecture for Analyzing Confidentiality". In: *2019 IEEE International Conference on Software Architecture (ICSA).* IEEE, 2019, pp. 1–10. DOI: `10.1109/ICSA.2019.00009`.

[25] Stephan Seifermann and Maximilian Walter. "Evolving a Use Case for Industry 4.0 Environments Towards Integration of Physical Access Control". In: *EMLS 2019: 6th Collaborative Workshop on Evolution and Maintenance of Long-Living Systems.* 2019.

[26] OASIS Standard. *eXtensible Access Control Markup Language (XACML) Version 3.0.* Jan. 2013. URL: `http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html`.

[27] S. R. Subramanya and B. K. Yi. "Digital rights management". In: *IEEE Potentials* 25.2 (Mar. 2006), pp. 31–34. ISSN: 0278-6648. DOI: `10.1109/MP.2006.1649008`.

[28] K. M. Ting. "Precision and recall". In: *Encyclopedia of machine learning.* Springer, 2011, pp. 781–781. ISBN: 978-0-387-30768-8.

[29] Sabrina De Capitani di Vimercati. "Discretionary Access Control Policies (DAC)". In: *Encyclopedia of Cryptography and Security.* Ed. by Henk C. A. van Tilborg and Sushil Jajodia. Boston, MA: Springer US, 2011, pp. 356–358. ISBN: 978-1-4419-5906-5. DOI: `10.1007/978-1-4419-5906-5_817`. URL: `https://doi.org/10.1007/978-1-4419-5906-5_817`.

[30] Bundesministerium für Wirtschaft und Energie (BMWi). *Zugriffssteuerung für Industrie 4.0-Komponenten zur Anwendung von Herstellern, Betreibern und Integratoren.* Diskussionspapier. Nov. 2018. URL: `https://www.plattform-i40.de/PI40/Redaktion/DE/Downloads/Publikation/zugriffssteuerung-industrie40-komponenten.html`.

A. Appendix

A.1. Generated Running Example

The Figure A.1 shows the generated XACML policy set file for the running example. The Figure A.2 shows the whole generated ensemble system and a scenario definition which checks the situation when the machine is OK.

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<PolicySet xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17" PolicySetId="completePolicySet" Version="1.0"
           PolicyCombiningAlgId="urn:oasis:names:tc:xacml:3.0:policy-combining-algorithm:permit-overrides">
   <Description>all policies combined</Description>
   <Target/>
   <Policy PolicyId="policy:readLog" Version="1.0" RuleCombiningAlgId="urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:permit-overrides">
      <Description>Policy for readLog</Description>
      <Target/>
      <Rule RuleId="rule:readLog:0" Effect="Permit">
         <Description>Context check rule for entity readLog’s characteristic 0</Description>
         <Target>
            <AnyOf>
               <AllOf>
                  <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
                     <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">readLog</AttributeValue>
                     <AttributeDesignator Category="urn:oasis:names:tc:xacml:3.0:attribute-category:action" AttributeId="entity:name"
                                          DataType="http://www.w3.org/2001/XMLSchema#string" MustBePresent="false"/>
                  </Match>
                  <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
                     <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">OK</AttributeValue>
                     <AttributeDesignator Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource" AttributeId="context:internalstate"
                                          DataType="http://www.w3.org/2001/XMLSchema#string" MustBePresent="false"/>
                  </Match>
                  <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-regexp-match">
                     <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">(\QA\E)</AttributeValue>
                     <AttributeDesignator Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject" AttributeId="context:organisation"
                                          DataType="http://www.w3.org/2001/XMLSchema#string" MustBePresent="false"/>
                  </Match>
                  <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-regexp-match">
                     <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">(\QWorker\E)</AttributeValue>
                     <AttributeDesignator Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject" AttributeId="context:role"
                                          DataType="http://www.w3.org/2001/XMLSchema#string" MustBePresent="false"/>
                  </Match>
               </AllOf>
            </AnyOf>
         </Target>
         <ObligationExpressions>
            <ObligationExpression ObligationId="obligation:prerequisite" FulfillOn="Permit">
               <AttributeAssignmentExpression AttributeId="context:prerequisite:sendReport">
                  <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">sendReport</AttributeValue>
               </AttributeAssignmentExpression>
            </ObligationExpression>
         </ObligationExpressions>
      </Rule>
      <Rule RuleId="rule:readLog:1" Effect="Permit">
         <Description>Context check rule for entity readLog’s characteristic 1</Description>
         <Target>
            <AnyOf>
               <AllOf>
                  <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
                     <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">readLog</AttributeValue>
                     <AttributeDesignator Category="urn:oasis:names:tc:xacml:3.0:attribute-category:action" AttributeId="entity:name"
                                          DataType="http://www.w3.org/2001/XMLSchema#string" MustBePresent="false"/>
                  </Match>
                  <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
                     <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">DEFECTIVE</AttributeValue>
                     <AttributeDesignator Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource" AttributeId="context:internalstate"
                                          DataType="http://www.w3.org/2001/XMLSchema#string" MustBePresent="false"/>
                  </Match>
                  <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-regexp-match">
                     <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">(\QB\E)</AttributeValue>
                     <AttributeDesignator Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject" AttributeId="context:organisation"
                                          DataType="http://www.w3.org/2001/XMLSchema#string" MustBePresent="false"/>
                  </Match>
                  <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-regexp-match">
                     <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">(\QRepairperson\E)</AttributeValue>
                     <AttributeDesignator Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject" AttributeId="context:role"
                                          DataType="http://www.w3.org/2001/XMLSchema#string" MustBePresent="false"/>
                  </Match>
               </AllOf>
            </AnyOf>
         </Target>
      </Rule>
      <Rule RuleId="denyIfNotApplicable" Effect="Deny">
         <Description>this rule denies if this case is not applicable</Description>
         <Target/>
      </Rule>
   </Policy>
</PolicySet>
```

Listing A.1: The XACML policy set file for the running example.

```scala
package scenarios
import tcof.{Component, _}
import java.time._
import java.time.format._
import java.util.Collection
import java.util.ArrayList
class RunningExample(val now: LocalTime) extends Model {
 class Subject(val subjectName: String, val organisation: String = null, val role: String = null) extends Component {
  name(s"Subject $subjectName")
 }
 class Resource(val resourceName: String, val accessSubject: Subject, val internalstate: String = null) extends Component {
  name(s"Resource $resourceName")
 }

 class System extends RootEnsemble {
  object readLog extends Ensemble {
   val allowedResources =
     components
     .select[Resource]
     .filter(x => (
          (x.accessSubject != null
            && ((x.accessSubject.organisation != null && x.accessSubject.organisation.matches("(\\QA\\E)"))
            && (x.accessSubject.role != null && x.accessSubject.role.matches("(\\QWorker\\E)")))
          && (x.internalstate == "OK"))
        ||(x.accessSubject != null
            && ((x.accessSubject.organisation != null && x.accessSubject.organisation.matches("(\\QB\\E)"))
            && (x.accessSubject.role != null && x.accessSubject.role.matches("(\\QRepairperson\\E)")))
          && (x.internalstate == "DEFECTIVE"))))
     .map[Resource](x => x.getClass().cast(x))
   val allowedSubjects =
     components
     .select[Subject]
     .filter(x => (
          (allowedResources.forall(y => ((y.internalstate == "OK")))
            && prereq_sendReport("sendReport")
            && (x.organisation != null && x.organisation.matches("(\\QA\\E)"))
            && (x.role != null && x.role.matches("(\\QWorker\\E)")))
        ||(allowedResources.forall(y => ((y.internalstate == "DEFECTIVE")))
          && (x.organisation != null && x.organisation.matches("(\\QB\\E)"))
          && (x.role != null && x.role.matches("(\\QRepairperson\\E)")))))
     .map[Subject](x => x.getClass().cast(x))

   allow(allowedSubjects, "readLog", allowedResources)
  }
  val readLogRule = rules(readLog)
 }
 val rootEnsemble = root(new System)
 def prereq_sendReport(operationSignatureName: String) : Boolean = {
   //TODO implement call to operation signature for operationSignature "sendReport"
   return true
 }
}

object RunningExample {
 def convertToCol(iterable: Iterable[Component]) : Collection[Component] = {
   val collection = new ArrayList[Component]
   val iter = iterable.iterator
   while (iter.hasNext) {
     collection.add(iter.next)
   }
   return collection
 }

 def main(args: Array[String]) : Unit = {
//TODO: adapt to your usecase scenario
   val scenario = new RunningExample(LocalTime.parse("13:00:00Z", DateTimeFormatter.ISO_OFFSET_TIME))
   val subjectA = new scenario.Subject("W", "A", "Worker")
   val subjectA2 = new scenario.Subject("W2", "A", "Worker")
   val subjectB = new scenario.Subject("R", "B", "Repairperson")
   val accessSubject = subjectA
   val resourceA = new scenario.Resource("machineOk", accessSubject, "OK")
   val resourceB = new scenario.Resource("machineDef", accessSubject, "DEFECTIVE")
   scenario.components = List(subjectA, subjectB, subjectA2, resourceA, resourceB)
   scenario.rootEnsemble.init()
   val solved = scenario.rootEnsemble.solve()
   scenario.rootEnsemble.instance.readLogRule.selectedMembers.foreach(x => println(convertToCol(x.allowedSubjects)));
   scenario.rootEnsemble.instance.readLogRule.selectedMembers.foreach(x => println(convertToCol(x.allowedResources)));
   val testActionAllow = solved && scenario.rootEnsemble.instance.readLogRule.selectedMembers.exists(x => convertToCol(x.allowedSubjects).contains(subjectA)
                                                                                                         && convertToCol(x.allowedResources).contains(resourceA)
                                                                                                         && !convertToCol(x.allowedSubjects).contains(subjectB))
   if (testActionAllow) {
     println("allow")
   } else {
     println("deny")
   }
 }
}
```

Listing A.2: The ensemble system for the running example.