Developer's Guide

# HERAS$^{AF}$: PEP

Department of Computer Science
University of Applied Sciences Rapperswil (HSR)
Spring Semester 2008

**Artifact of HERAS$^{AF}$: PEP and PDP Web Service Integration [RW08PEP]**

Students: Daniel Regli, Yannick Winiger
Examiner: Wolfgang Giersche
Coaches: René Eggenschwiler, Florian Huonder

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

HERAS^AF
PEP

# Table of Contents

**PART I: INTRODUCTION ........ 6**

**1 INTENDED AUDIENCE ........ 7**
**2 OVERVIEW ........ 8**
2.1 DEFINITION ........ 8
2.2 HERAS^AF ........ 9
2.3 REQUIREMENTS OF THE PEP ........ 11
2.4 MODULE DESIGN ........ 12

**PART II: CONFIGURATION ........ 15**

**1 OVERVIEW ........ 16**
**2 SPRING CONFIGURATION – MODULE AND COMPONENTS WIRING ........ 17**
2.1 CONFIGURATION FILES ........ 17
2.2 SPRING SECURITY (SECURITY.XML) ........ 18
2.3 CORE (CORE.XML) ........ 19
2.4 HERAS^AF XACML 2.0 IMPLEMENTATION (HERASFXACML.XML) ........ 21
2.5 PDP (PDP.XML) ........ 21
2.6 SPRING AOP (SPRINGAOP.XML) ........ 22
2.6.1 DataTypeAttributes (DataTypeAttributes.xml) ........ 25
2.7 CONTEXTANDPOLICYCONFIGURATION (CONTEXTANDPOLICYCONFIGURATION.XML) ........ 26
**3 HINTS ........ 27**
3.1 SPRING IDE ........ 27

**PART III: ARCHITECTURE & DESIGN ........ 28**

**1 OVERVIEW ........ 29**
1.1 ARCHITECTURE ........ 29
1.1.1 Modules ........ 29
1.1.2 Interaction scenario ........ 30
**2 CORE ........ 32**
2.1 HERAS^AF PEP IMPLEMENTATION ........ 32
2.2 PACKAGE ANNOTATION ........ 33
2.3 PACKAGE CORE ........ 34
2.4 PACKAGE DATA ........ 35
2.5 PACKAGE EXCEPTION ........ 36
2.6 PACKAGE INTEGRATION ........ 37
2.7 PACKAGE OBLIGATION ........ 38
2.8 PACKAGE RESOLVER ........ 39
2.9 PACKAGE UTIL ........ 40
2.10 INTERFACES / ABSTRACT CLASSES ........ 40
2.10.1 AbstractDecisionHandler ........ 41
2.10.2 AbstractHerasDecisionException ........ 41
2.10.3 DecisionContext ........ 41
2.10.4 DecisionHandler ........ 42

2.10.5 DecisionManager ........ 42
2.10.6 Pdp ........ 43
2.10.7 PdpLocator ........ 43
2.10.8 RequestContext ........ 43
2.10.9 RequestContextFactory ........ 44
2.10.10 ResponseContext ........ 44
2.10.11 ObligationHandler ........ 44
2.10.12 ObligationProcessor ........ 45
2.11 SCENARIOS ........ 45
2.11.1 Decision Handling ........ 45
2.11.2 Obligation Handling ........ 47
2.12 DESIGN DECISIONS ........ 49
2.12.1 Attribute data types ........ 49
2.12.2 Finding/locating a PDP ........ 50
2.12.3 XACML response ........ 51
2.12.4 Decision handling ........ 51
2.12.5 Obligation handling ........ 53
**3 SPRING AOP ........ 55**
3.1 HERAS^AF PEP IMPLEMENTATION ........ 55
3.2 PACKAGE SPRINGAOP ........ 55
3.3 INTERFACES / ABSTRACT CLASSES ........ 57
3.3.1 AbstractSpringAopXXXResolver ........ 57
3.4 SCENARIO ........ 57
3.4.1 Intercepting ........ 58
3.5 DESIGN DECISIONS ........ 58
3.5.1 Interception of a method call ........ 58
3.5.2 Resolving request information ........ 60
**4 HERASAFXACML ........ 63**
4.1 HERAS^AF PEP IMPLEMENTATION ........ 63
4.2 PACKAGE HERASAFXACML ........ 63
4.3 INTERFACES / ABSTRACT CLASSES ........ 64
4.3.1 AbstractTransformer ........ 65
4.4 DESIGN DECISIONS ........ 65
4.4.1 XACML request context creation ........ 65
**5 UTILITIES ........ 67**
5.1 HERAS^AF PEP IMPLEMENTATION ........ 67
5.2 PACKAGE ANNOTATION ........ 67
5.3 INTERFACES / ABSTRACT CLASSES ........ 68
5.3.1 AnnotationCache ........ 69
5.3.2 AnnotationParser ........ 69
5.4 CONCEPTS ........ 69
5.4.1 Annotation gathering ........ 69
5.5 DESIGN DECISIONS ........ 70
5.5.1 Java annotation gathering algorithm ........ 70
5.5.2 Parsing request information ........ 71
**6 GENERAL TOPICS ........ 73**
6.1 EXCEPTION HANDLING ........ 73
6.1.1 Exception Types ........ 74
6.1.2 Client exception handling ........ 74

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

HERAS^AF PEP

**PART IV: IMPLEMENTING ADDITIONAL INTEGRATION MODULES ........ 75**

**1 OVERVIEW ........ 76**
**2 INTEGRATION OF AN INTERCEPTOR TECHNOLOGY ........ 76**
2.1 COMPONENTS IMPLEMENTATION ........ 76
2.1.1 Pointcut ........ 76
2.1.2 Advice ........ 76
2.1.3 DecisionContextFactory ........ 77
2.1.4 DecisionContext ........ 77
2.1.5 Resolver ........ 77
**3 INTEGRATION OF AN XACML 2.0 IMPLEMENTATION ........ 78**
3.1 COMPONENTS IMPLEMENTATION ........ 78
3.1.1 RequestContext ........ 78
3.1.2 RequestContextFactory ........ 78
3.1.3 ResponseContext ........ 78
3.1.4 Transformer ........ 79

**PART V: TESTING ........ 80**

**1 OVERVIEW ........ 81**
**2 MODULE / - UNIT TESTS ........ 81**
2.1 CORE MODULE ........ 81
2.1.1 TestDefaultDecisionHandler ........ 81
2.1.2 TestDefaultExceptionHandler ........ 82
2.1.3 TestDefaultObligationProcessorImpl ........ 82
2.2 HERASAFXACML MODULE ........ 83
2.2.1 TestActionTransformer ........ 83
2.2.2 TestEnvironmentTransformer ........ 83
2.2.3 TestRequestContext ........ 83
2.2.4 TestResourceTransformer ........ 83
2.2.5 TestSubjectTransformer ........ 83
2.3 UTILITIES MODULE ........ 84
2.3.1 TestAnnotationParser ........ 84
2.3.2 TestAnnotationUtils ........ 84
**3 INTEGRATION TESTS ........ 85**
3.1 HERASAF-PEP-INTEGRATIONTESTS ........ 85
3.1.1 TestAopProtectedClass ........ 85
**4 TESTING HINTS ........ 86**
4.1 CODE COVERAGE - ECLEMMA ........ 86
4.2 DO NOT RUN TESTS IN ECLIPSE ........ 86

**PART VI: DEPLOYMENT ........ 87**

**1 OVERVIEW ........ 88**
**2 APACHE MAVEN 2 ........ 88**
2.1 PLUGINS ........ 88
**3 HERASAF-PEP ........ 91**

4 HINTS ........ 92
4.1 ECLIPSE AND MAVEN ........ 92

APPENDIX A: GENERAL ........ 93

1 GLOSSARY ........ 94
2 BIBLIOGRAPHY ........ 96
2.1 SPECIFICATIONS AND STANDARDS ........ 96
2.2 HERAS$^{AF}$ DOCUMENTS ........ 96
2.3 OTHER RESOURCES ........ 98

APPENDIX B: ISSUES, EXTENSIONS AND REFACTORINGS ........ 100

1 OVERVIEW ........ 101
2 OPEN ISSUES ........ 101
3 EXTENSION POINTS ........ 101
4 REFACTORINGS ........ 102

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

HERAS^AF
PEP

# Part I: Introduction

# 1 Intended audience

*General*

This Developer's Guide provides information on the architecture and design of the HERAS^AF PEP Implementation. The guide will assist developers in understanding the design.

They will find information about packages, interfaces, classes and flows. They also get an insight into the developers' train of thoughts of the HERAS^AF PEP Implementation.

Additionally hints and other help topics are provided which makes it easier for further development.

The developers should bring basic understanding in the following technologies, frameworks and standards:

- Java 1.5
- HERAS^AF
- Spring Framework (Core [Spring], Spring AOP [SpringAOP], Spring Security [SpringSecurity])
- Maven2 [Maven]
- TestNG [TestNG]
- XACML 2.0 [XACML]
- UML

# 2 Overview

## 2.1 Definition

*HERAS$^{AF}$*

HERAS$^{AF}$ is an Open Source project in its formation phase.

It assists the entire authorization process based on the OASIS XACML 2.0 standard. That means all accesses to protected resources are intercepted by a PEP and sent to a PDP, which evaluates the request on the basis of applicable Evaluatables. In case of a positive answer the PEP allows the access to the resource. Additionally the maintenance is possible through a PAP.

*HERAS$^{AF}$ PEP*

The HERAS$^{AF}$ PEP provides mechanisms to implement intercepting agent's which enforces access control in protection worth applications. Its main responsibility is building authorization requests and interpreting authorization responses.

*HERAS$^{AF}$ PEP Web Service*

The HERAS$^{AF}$ PEP Web Service (WS) implements a Web Service client that can be integrated into the HERAS$^{AF}$ PEP architecture. This integration makes it possible that a PEP can send decision requests to a Web Service interface of a remote PDP.

HERAS$^{AF}$ PDP

HERAS$^{AF}$ PDP implements the entire logic for decision making of accesses. This covers fast locating of potential policies, evaluation of a request with the aid of the located policies as well as combining the obtained results.
The main focus of the HERAS$^{AF}$ PDP is performance. To achieve this, performance-enhancing mechanisms such as indexing and fast comparison algorithms are developed. Another essential point is accessing the data sink only during the initialization phase of the PDP. Afterwards the policies are kept in the RAM.

The HERAS$^{AF}$ PDP will be used as a central unit, which interacts with several PEPs. [DOH07DADev]

*HERAS$^{AF}$ PDP Context-WS*

HERAS$^{AF}$ PDP Context-WS is part of a HERAS$^{AF}$ PDP and implements a Web Service endpoint for a Spring Web Service to process decision requests sent by PEPs.

HERAS$^{AF}$ PAP

The HERAS$^{AF}$ PAP is responsible to build and administrate policies. Additionally it deploys the policies to the PDP's. For the building process of complex policies a graphical user interface was developed. For further information to this component see the paper HERAS$^{AF}$ PAP. [NZ08PAPDev]

*HERAS$^{AF}$ PIP*

The HERAS$^{AF}$ PIP is responsible to resolve missing attributes of the request. If a HERAS$^{AF}$ PDP needs further information to evaluate a request, the HERAS$^{AF}$ PIP is called. If possible the HERAS$^{AF}$ PIP returns additional attributes.

*XACML*

XACML is an OASIS standard that describes both a policy language and an access control decision request/response language (both written in XML). The policy language is used to describe general access control requirements, and has standard extension points for defining new functions, data types, combining

logic, etc. The request/response language lets you form a query to ask whether or not a given action should be allowed, and interpret the result. The response always includes an answer about whether the request should be allowed using one of four values: Permit, Deny, Indeterminate (an error occurred or some required value was missing, so a decision cannot be made) or Not Applicable (the request can't be answered by this service). [XACML]

## 2.2 HERAS^AF

*Background*

HERAS^AF is an open source project in its beginning phase. Initially started in January 2006, after eight month of research and development of ideas, conception and projection by Wolfgang Giersche, Yan Graf and René Eggenschwiler.

A proof of concept was made by Yan Graf [Graf06] and René Eggenschwiler [Egg06] in mid 2006. They built nearly all the components as prototypes for a working HERAS^AF, based on Sun's XACML implementation.

In 2007 Massimo Cerqui and Sandro Strebel build a Policy Enforcement Point based on interceptors for Spring AOP and AspectJ. They also build a prototype for a policy based security management system with Spring/JSF [CS07PAP]. In the same year Sacha Dolski, Stefan Oberholzer, Florian Huonder created a XACML PDP Web Service endpoint based on Sun's XACML implementation. They soon found out that Sun's XACML was not exactly what they were looking for. Therefore they decided to develop an own XACML 2.0 implementation. It was finished at the end of 2007.

*Purpose*

HERAS^AF assists the entire authorization process based on the OASIS XACML 2.0 standard. That means all accesses to protected resources are intercepted by a PEP and sent to a PDP, which evaluates the request on the basis of applicable Evaluatables. In case of a positive answer the PEP allows the access to the resource. Additionally the maintenance is guaranteed through a PAP which allows administrating policies.

*Architecture*

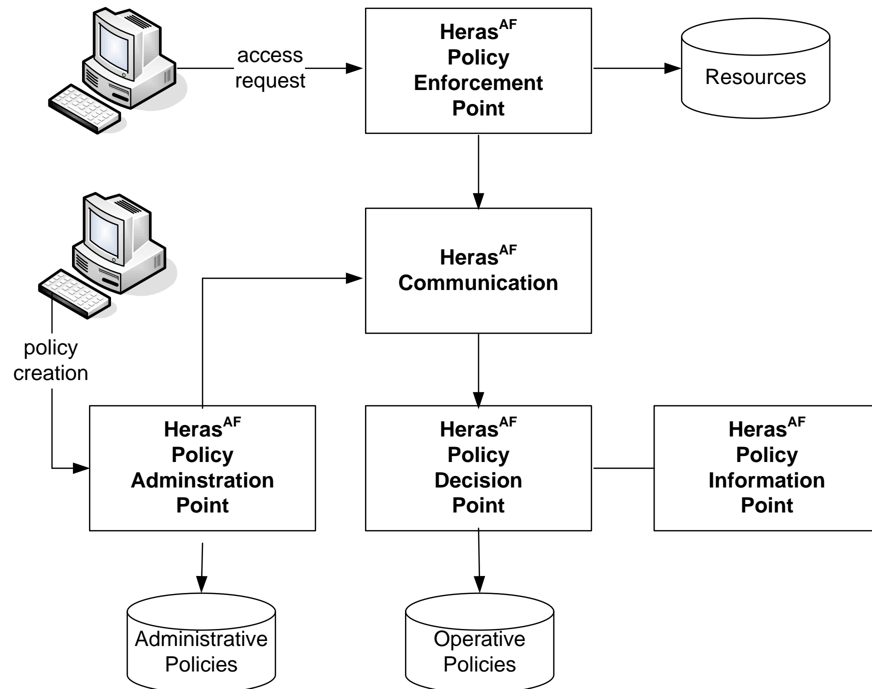



**Figure 1: HERASAF components**

**Policy Administration Point (PAP)**
The PAP is an administration component. With a PAP you are able to administrate policies. This includes deploying the policies to a PDP.

**Policy Enforcement Point (PEP)**
The PEP is an enforcement component (performer). He interrupts accesses on secured resources and creates an authorization-request which he will forward to a Context Handler. Depending on the authorization-respond (decision) of the PDP the access will be allowed or denied. Additionally obligations included in the decision must be handled and fulfilled from the PEP before access is allowed.

**Policy Decision Point (PDP)**
The PDP is the decision maker. He gets the authorization-request and generates an authorization-respond depending on policies. If the policies contain obligation the PDP needs to send those in the respond as well.

**Policy Information Point (PIP)**
The PIP is the additional information expert. He provides additional information about subjects, resources, environment or actions. This information can be referenced by policies.

*Special Characteristic*

Compared to the XACML 2.0 specification HERAS^AF has one major difference: There is no ContextHandler module.

The functionality of the ContextHandler has been integrated directly into the HERAS^AF PEP and HERAS^AF PDP. The HERAS^AF PEP already generates XACML-complaint requests and the HERAS^AF PDP enquires additional information directly from a PIP.

## 2.3 Requirements of the PEP

*Introduction*

There are several requirements to an implementation of a Policy Enforcement Point. This chapter lists the most important requirements and gives a short explanation on these requirements.

*Functional requirements*

| **Requirements** | |
|---|---|
| Intercepting / Filter method | A way must be provided to intercept an application flow before a secured item can be accessed.<br><br>See part 3 chapter 3.5.1 for more information. |
| Request information | For each secured item the user needs to be able to give additional request information about the subject, resource, action and environment as specified by the XACML 2.0 specification.<br><br>See part 3 chapter 3.5.2 for more information. |
| XACML request creation | The PEP must be able to generate a standard-compliant XACML 2.0 requests, based on the request information and data from other sources.<br><br>See part 3 chapter 4.4.1 for more information. |
| Handling of XACML response | The PEP must assure that obligations in an authorization decision are handled properly and that they influence the decision handling depending on the decision and the fulfillment of the obligations.<br><br>See part 3 chapter 2.12.4 for more information. |
| Concurrency | The PEP must handle concurrency because it can be used in a concurrent application. |
| Relay response to application | The PEP must provide a method to relay the response based on the 4 decisions *Permit*, *Deny*, *Indeterminate* and *NotApplicable* to the application. The additional information contained in the response must be included.<br><br>See part 3 chapter 2.12.4 for more information. |
| Extensibility | There must be interfaces to extend the current PEP implementation.<br><br>• Interception: The integration of any technology that intercepts the application flow must be supported.<br><br>• Request information: The integration of any technology to add additional request information for a secured item must be supported. (e.g. request information in a XML file or a data base)<br><br>• PDP: The integration of any PDP implementation must be supported. (e.g. PDP Web Service Stub)<br><br>• Attribute data types: The integration or usage of any attribute data type must be supported.<br><br>• Response handling: The integration of an |

application specific response handler must be supported.

- Obligation handling: The integration of custom obligation handler into the obligation handling process must be supported.
- XACML implementation: The integration of any XACML 2.0 implementation must be supported.

*Non-functional requirements*

**Usability**

| | |
|---|---|
| Documentation | The API of HERAS^AF PEP must be documented with JavaDoc in English. The documentation should be understandable for any developer.<br>A developer's and user's guide must be provided in English. The developer's guide will help to understand the current implementation and gives a deep understanding into where and how it can be extended. The user's guide will show how to use and configure the HERAS^AF PEP and its modules. |
| Operability | Settings should be made in configuration files which are self-explaining or documented. |

**Maintainability, Changeability**

| | |
|---|---|
| Analysis | For the HERAS^AF PEP implementation every single design decision and the reason for the decision must be documented. This will help for later extensions or changes. |
| Verifiability | Integration and module tests must be provided to verify the implementation. |

**Reliability**

| | |
|---|---|
| Fault tolerance | The PEP cannot tolerate any faults because the PEP is the enforcement point and is the last station in the whole access control process. The security depends on the PEP as much as where the policies are evaluated.<br>If a system exception inside the PEP arises it must been thrown to the client and access to the resource must always be denied. |

## 2.4 Module Design

*Overview*

The HERAS^AF PEP implementation is divided up in 5 modules. This chapter briefly explains the modules. The modules separate the HERAS^AF PEP into logical units with concrete responsibilities.

*Module Overview*

Figure 2 shows the module overview of the HERAS$^{AF}$ PEP. The purple highlighted modules could be replaced by another implementation. These are the integration modules.

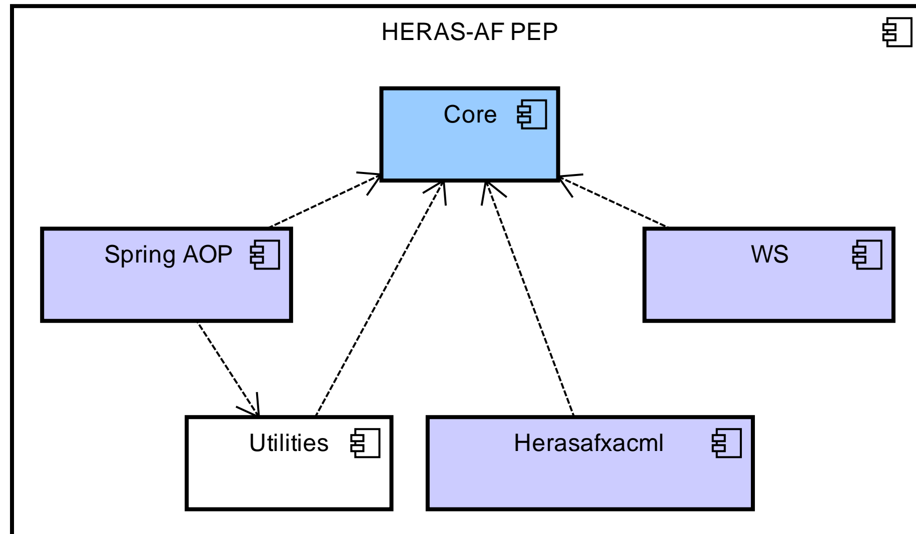



**Figure 2: Module overview**

*Modules*

**Core**

The Core module contains the structural models, the core functionality as well as the HERAS$^{AF}$ PEP logic. It provides various integration interfaces/classes for an intercepting technology and XACML implementation.

**Spring AOP**

The Spring AOP module contains the Spring AOP interceptor technology, the resolvers and the Spring AOP context. The resolvers are used to get the data from the annotated/marked elements. This data is kept in the Spring AOP context which is used to create a request.

This module is used whenever the intercepting technology is Spring AOP. The whole module could be replaced by an AspectJ module implementation or another intercepting technology.

**Herasafxacml**

The Herasafxacml module contains the integration of the PEP and the specific HERAS$^{AF}$ XACML 2.0 implementation. The transformation from the HERAS$^{AF}$ PEP data structure into a HERAS$^{AF}$ XACML 2.0 request is made here.

This module is used only in connection with HERAS$^{AF}$ XACML. The whole module could be replaced by another vendor's XACML 2.0 implementation.

**Utilities**

The Utilities module contains general annotation collecting classes. Those are used to collect annotations on classes and interfaces because they are not inherited by the java language. An annotation cache algorithm for optimized annotation collecting is in this module as well.

**WS**

The WS module contains the classes to connect the HERAS$^{AF}$ PEP to a PDP

Web Service. This includes a SAML Handler and some classes to guarantee the SAML over SOAP standard-conformance.

This module is only used in connection with a HERAS^AF PDP Web Service. Otherwise it is replaced by an own integration implementation. This module will not be further explained in this document. See the paper [RW08PEPDev].

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

HERAS^AF
PEP

# Part II: Configuration

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

HERAS[AF]
PEP

# 1 Overview

*Introduction*

The modules of the HERAS[AF] PEP implementation are configurable through the Spring Framework application context [Spring].

Plugability is a very important issue because of the various extension and adaption points in these two modules. The Spring application context provides the means to plug-in specific implementations without changing a single line of code.

This chapter explains what can be configured and how it is done.

*Configuration Topics*

The following list shows the configuration topics of the HERAS[AF] PEP Implementation.

**Spring configuration – Module and Components wiring**

The modules and the components are wired through the Spring Framework. This allows customizing the HERAS[AF] PEP implementation with different components.

**Spring AOP configuration**

An interceptor technology is used when a protected method is called. The Spring AOP configuration describes this with the Spring AOP [SpringAOP] integration module.

**HERAS[AF] XACML 2.0 Implementation**

The XACML 2.0 implementation integration is used by the HERAS[AF] PEP. This configuration describes the HERAS[AF] PEP with the HERAS[AF] XACML 2.0 Implementation.

*Configuration files*

The configuration of the HERAS[AF] PEP implementation is divided into several significant configuration files. Each configurable part of the implementation is described in the following chapter.

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

HERAS[AF]
PEP

# 2 Spring configuration – Module and Components wiring

*Overview*

The HERAS[AF] PEP implementation uses the Spring Framework to wire and configure its modules. The bean graph of each section shows the dependency of each bean to other beans.

Where possible, user-friendly configuration files are defined and embedded using the Spring Framework feature of Extensible XML Authoring. In cases where this was impossible the configuration must take place in the application context configuration files.

## 2.1 Configuration files

*Overview*

This part shows how to embed the configuration files of the components into the application context of the Spring Framework.

The configurations files are not complete and will not work the way there are published here. For example package paths to the classes are shortened. The purposes of the examples are to understand the wiring and possibilities.

*Common elements*

All the configuration files have some common elements. These are outlined here.

```
<beans
xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema
/beans
http://www.springframework.org/schema/beans/spring-beans-
2.5.xsd">
```

**Example 1: Bean namespaces**

*code structure*

```
<beans …>
      <import resource="Security.xml" />
      <import resource="Core.xml" />
      <import resource="HerasafXACML.xml" />
      <import resource="Pdp.xml" />
      <import resource="SpringAop.xml" />
</beans>
```

**Example 2: Configuration imports**

*Hint*

The order of the imports does not matter but the imports must occur at the beginning of the beans part. No bean-definition may appear before.

| Element description | |
|---|---|
| import | The import statement to import the configuration files. The resource attribute contains the path of the configuration file. |

## 2.2 Spring Security (Security.xml)

*Overview*

This part shows the Spring Security [SpringSecurity] configuration.

*code structure*

The code snippet shows a Spring Security authentication and authorization infrastructure. The authentication manager utilizes an authentication provider that has users, passwords and roles configured within this XML file.
In live-system you would certainly find LDAP or JDBC-based providers here.

```
<bean id="authenticationManager"
      class="…ProviderManager">
      <property name="providers">
            <list>
              <ref bean="daoAuthenticationProvider" />
            </list>
      </property>
</bean>

<bean id="daoAuthenticationProvider"
      class="…DaoAuthenticationProvider">
      <property name="userDetailsService">
            <ref bean="inMemoryDaoImpl" />
      </property>
</bean>

<bean id="inMemoryDaoImpl"
      class="...InMemoryDaoImpl">
      <property name="userMap">
            <value>
            <![CDATA[
                  admin=adminPw,ROLE_ADMIN
                  user=userPW,ROLE_USER
            ]]>
            </value>
      </property>
</bean>
```

**Example 3: Spring Security configuration**

| Element | description |
|---|---|
| bean with id authenticationManager | This element configures which implementation of a ProviderManager should be used. The ProviderManager has a list of AuthenticationProviders to use. |
| bean with id daoAuthenticationProvider | This element configures which implementation of an AuthenticationProvider should be used. The used DaoAuthenticationProvider has a property to set a UserDetailService. |
| bean with id inMemoryDaoImpl | This element configures which implementation of a UserDetailService is used. The used InMemoryDaoImpl has a property to set a UserMap. |

*Bean graph*

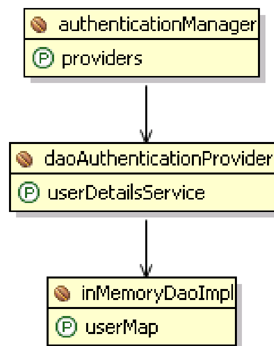



**Figure 3: Security bean graph**

## 2.3 Core (Core.xml)

*Overview*

This part shows an example of a Core module configuration.

*code structure*

```
<bean id="herasDecisionManager"
      class="…DecisionManagerImpl">
      <property name="pdpLocator" ref="pdpLocator"/>
      <property name="decisionHandler"
      ref="decisionHandler"/>

      <property name="requestFactory"
      ref="requestFactory"/>
      <property name="exceptionHandler"
      ref="exceptionHandler" />
</bean>

<bean id="decisionHandler"
      class="…DefaultDecisionHandler">
      <property name="obligationProcessor"
      ref="obligationProcessor" />
</bean>

<bean id="obligationProcessor"
      class="…ObligationProcessorImpl">
      <property name="obligationHandlers">
            <util:map>
                  <entry key="obligation-db"
                  value-ref = "dbObligationHandler" />
            </util:map>
      </property>
</bean>

<bean id="exceptionHandler"
      class="…DefaultExceptionHandler" />
```

**Example 4: Core configuration**

*Hint*

The configuration needs an additional namespace definition.

```
xmlns:util="http://www.springframework.org/schema/util"
xsi:schemaLocation="http://www.springframework.org/schema
/util http://www.springframework.org/schema/util/spring-
util-2.5.xsd"
```

**Example 5: Additional namespace definition**

| **Element description** | |
|---|---|
| bean with id `herasDecisionManager` | This element configures which implementation of a `DecisionManager` should be used. The `DecisionManager` has a property pdpLocator for a `PdpLocator` implementation, a property decisionHandler for a `DecisionHandler` implementation and a requestFactory property for a `RequestContextFactory`. |
| bean with id `decisionHandler` | This element configures which implementation of a `DecisionHandler` should be used. The `DecisionHandler` has a property for an `ObligationProcessor` implementation. |
| bean with id `obligationProcessor` | This element configures which implementation of an `ObligationProcessor` should be used. The `ProviderManager` has a list of `ObligationHandler` to use. |
| bean with id `exceptionHandler` | This element configures which implementation of a `ExceptionHandler` should be used. |

*Bean graph*

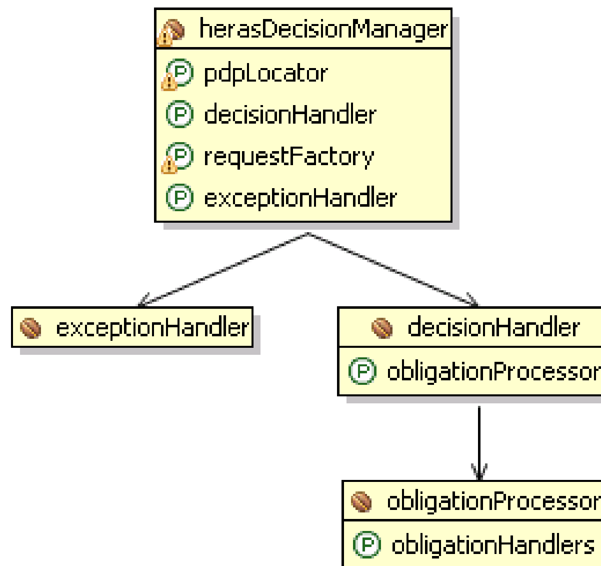



**Figure 4: Core bean graph**

## 2.4 HERAS$^{AF}$ XACML integration (HerasfXACML.xml)

*Overview*

It is recommended to configure a HERAS$^{AF}$ PEP with HERAS$^{AF}$ XACML [DOH07DA].

This chapter describes this configuration.

*code structure*

This code snippet shows a configuration example for an integration of the HERAS$^{AF}$ XAML 2.0 Implementation.

```
<import resource="ContextAndPolicyConfiguration.xml" />

<bean id="requestFactory"
      class="…HerasRequestContextFactory"/>
```

**Example 6: HERAS$^{AF}$ XACML integration configuration**

| Element description | |
|---|---|
| import with resource `ContextAndPolicy-Configuration.xml` | The import statement to import the configuration files. The resource attribute contains the path to the configuration file.<br>See chapter 2.7 for further information. |
| bean with id `requestFactory` | This element configures which implementation of a `RequestContextFactory` should be used. |

## 2.5 PDP (Pdp.xml)

*Overview*

This part shows an example configuration for a Pdp.

*code structure*

```
<bean id="pdpLocator" class="…JVMLocalPdpLocator">
      <property name="pdp">
            <ref bean="remotePdp" />
      </property>
</bean>
```

**Example 7: Pdp configuration**

| Element description | |
|---|---|
| bean with id `pdpLocator` | This element configures which implementation of a `PdpLocator` should be used. The used `PdpLocator` has a property pdp to define a `Pdp` implementation. |

*Bean graph*

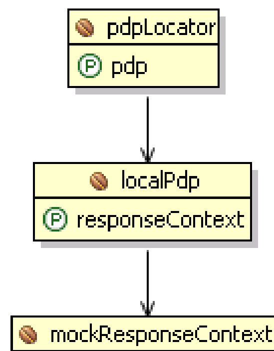



**Figure 5: Pdp bean graph**

## 2.6 Spring AOP (SpringAop.xml)

*Overview*

This part shows an example configuration for the integration with the Spring AOP intercepting technology.

*code structure*

```
<import resource="DataTypeAttributes.xml" />

<aop:config proxy-target-class="true">
      <aop:aspect id="aroundAdvice"
ref="accessControlAdvice">
      <aop:around pointcut-
ref="expressionAnnotationPointcut"
            method="secureMethod" />
      </aop:aspect>
</aop:config>

<bean id="expressionAnnotationPointcut"
      class="…ExpressionAnnotationPointcut">
      <property name="expression"
      value="execution(* org.herasaf.secured.*.*(..))" />
</bean>

<bean id="accessControlAdvice"
      class="…AccessControlAdvice">
            <property
      name="springAopDecisionContextFactory"
            ref="springAopDecisionContextFactory" />
            <property name="decisionManager"
      ref="herasDecisionManager" />
</bean>

<bean id="springAopDecisionContextFactory"
      class="…SpringAopDecisionContextFactory">
      <property name="subjectResolver"
      ref="subjectResolver" />
      <property name="resourceResolver"
      ref="resourceResolver" />
      <property name="actionResolver"
```

```
        ref="actionResolver" />
        <property name="environmentResolver"
        ref="environmentResolver" />
</bean>

<bean id="abstractResolver" abstract="true">
        <property name="attributeDataTypes"
              ref="attributeConfiguration" />
</bean>

<bean id="subjectResolver"
        class="…AcegiSpringAopSubjectResolver"
              parent="abstractResolver" />

<bean id="resourceResolver"
        class="…AnnotationSpringAopResourceResolver"
              parent="abstractResolver">
        <property name="annotationParser"
        ref="annotationParser" />
</bean>

<bean id="actionResolver"
        class="…AnnotationSpringAopActionResolver"
        parent="abstractResolver">
        <property name="annotationParser"
ref="annotationParser" />
</bean>

<bean id="environmentResolver"
        class="…AnnotationSpringAopEnvironmentResolver"
        parent="abstractResolver">
        <property name="annotationParser"
ref="annotationParser" />
</bean>

<bean id="annotationParser"
        class="…AnnotationParserImpl">
        <property name="annotationCache">
              <bean class="…SynchronizedAnnotationCache" />
        </property>
        <property name="attributeDataTypes"
              ref="attributeConfiguration" />
        </bean>
```

**Example 8: Spring AOP configuration**

| **Element description** | |
|---|---|
| import with resource `DataType-Attributes.xml` | The import statement to import the configuration files. The resource attribute contains the path the configuration file.<br><br>See chapter 2.6.1 for further information. |
| aop:config element | This element defines a Spring AOP aspect and an around advice in connection with its pointcut. Whenever the pointcut gets active the `secureMethod` method is called. |

| | |
|---|---|
| bean with id `expressionAnnotationPointcut` | This element configures which implementation of an `AspectJExpressionPointcut` should be used. The used `ExpressionAnnotationPointcut` has a property expression to define an expression for the pointcut. |
| bean with id `accessControlAdvice` | This element configures which implementation of an `AccessControlAdvice` should be used. The used `AccessControlAdvice` has a property springAopDecisionContextFactory to define a `SpringAopDecisionContextFactory` implementation and a property decisionManager to define a `DecisionManager` implementation. |
| bean with id `springAopDecisionContextFactory` | This element configures which implementation of a `SpringAopDecisionContextFactory` should be used. The used `SpringAopDecisionContextFactory` has four properties:<br>• subjectResolver for an `AbstractSpringAopSubjectResolver` implementation<br>• resourceResolver for an `AbstractSpringAopResourceResolver` implementation<br>• actionResolver for an `AbstractSpringAopActionResolver` implementation<br>• environmentResolver for an `AbstractSpringAopEnvironmentResolver` implementation |
| bean with id `abstractResolver` | This element configures an abstract bean definition with common behavior. The common behavior is the attributeDataTypes property. |
| bean with id `subjectResolver` | This element configures which implementation of an `AbstractSpringAopSubjectResolver` should be used. The used `AcegiSpringAopSubjectResolver` inherits all properties of the `abstractResolver`. |
| bean with id `resourceResolver`, `actionResolver`, `environmentResolver` | This element configures which implementation of an `AbstractSpringAopXXXResolver` should be used. The used Resolver inherits all properties of the `abstractResolver` and has an addition property annotationParser to define an `AnnotationParser` implementation. |
| bean with id `annotationParser` | This element configures which implementation of an `AnnotationParser` should be used. The used `AnnotationParserImpl` has a property annotationCache to define an `AnnotationCache` implementation and a property attributeDataTypes |

to define an `AttributeDataTypes` implementation.

*Hint*

The configuration needs an additional namespace definition.

```
xmlns:aop="http://www.springframework.org/schema/aop"
xsi:schemaLocation="http://www.springframework.org/schema
/aop http://www.springframework.org/schema/aop/spring-
aop-2.5.xsd"
```

**Example 9: Additional namespace definition**

*Bean graph*

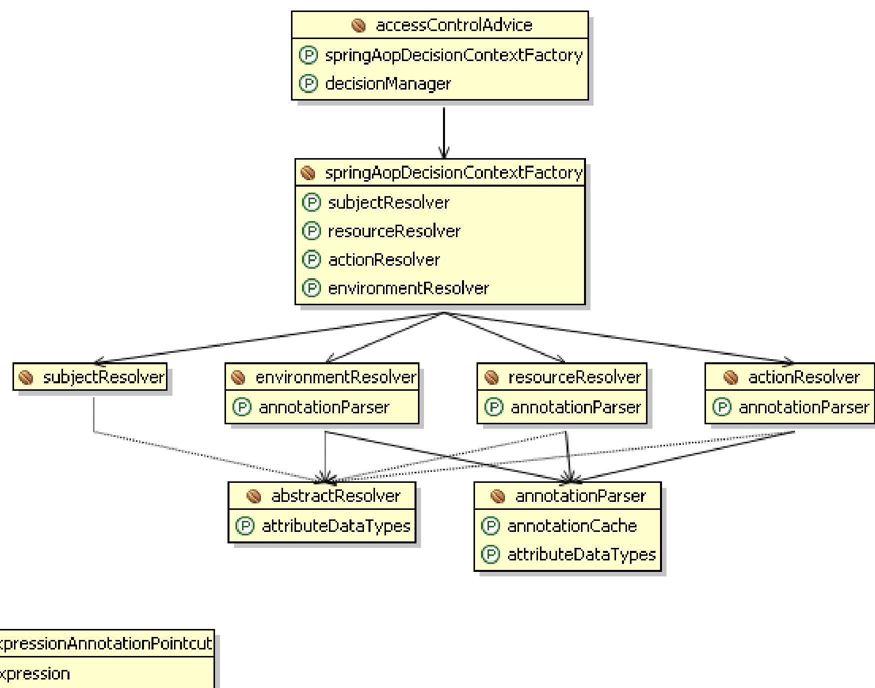



**Figure 6: Spring AOP bean graph**

### 2.6.1 DataTypeAttributes (DataTypeAttributes.xml)

*code structure*

This code snippet shows a configuration example for the data type mapping in connection.

```
<bean id="attributeConfiguration"
      class="…AttributeDataTypes">
      <property name="attributeDataTypes">
      <map>
            <entry key="A:A1:resource:1"
      value="http://www.w3.org/2001/XMLSchema#String" />
            <entry key=" A:A2:resource:1"
      value="http://www.w3.org/2001/XMLSchema#String" />
      </map>
      </property>
</bean>
```

**Example 10: DataTypeAttribute mapping**

**Element description**

| | |
|---|---|
| bean with id `attributeConfiguration` | This element configures which implementation of the `AttributeDataTypes` is used. The used `AttributeDataTypes` has a property map where the mapping between the Attribute ID and the type is defined. |

## 2.7 ContextAndPolicyConfiguration (ContextAndPolicyConfiguration.xml)

*Overview*

The ContextAndPolicyConfiguration.xml describes the various configuration options of the context and policy files.

*Configuration*

For a detailed configuration guide see the developer's guide of HERAS^AF XACML 2.0 [DOH07DADev].

# 3 Hints

*Introduction*

This chapter gives some hints while working with the Spring Framework to simplify the bean configuration.

## 3.1 Spring IDE

*Description*

Spring IDE is a graphical user interface for the configuration files used by the Spring Framework. It's built as a set of plugins for the Eclipse platform. [SpringIDE]

Spring IDE provides some useful features:

- Project nature which supports a list of Spring bean configuration files
- Incremental builder which validates all modified Spring bean configuration files defined in a Spring project
- View which displays a tree with all Spring projects and their Spring bean configuration files
- Image decorator which decorates all Spring projects, their bean configuration files and all Java classes which are used as bean classes
- Graph which shows all beans (and their relationships) defined in a single configuration file or a configuration set
- XML editor for Spring beans configuration files

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

HERAS^AF
PEP

# Part III: Architecture & Design

# 1 Overview

*Introduction*

This part of the Developer's Guide explains the architecture and the design of the HERAS$^{AF}$ PEP implementation.

## 1.1 Architecture

*Overview*

This chapter explains the architecture of the HERAS$^{AF}$ PEP implementation.

The HERAS$^{AF}$ PEP implementation has a modular architecture. This ensures that different parts of the logic can be changed without affecting the rest of the HERAS$^{AF}$ PEP. The core module affects all other modules and cannot be changed.

In this chapter the responsibilities of the different module are explained, how they interact and how the implementation is done.

### 1.1.1 Modules

*Overview*

This chapter explains the dependencies and gives a short description of each module.

*Module diagram*

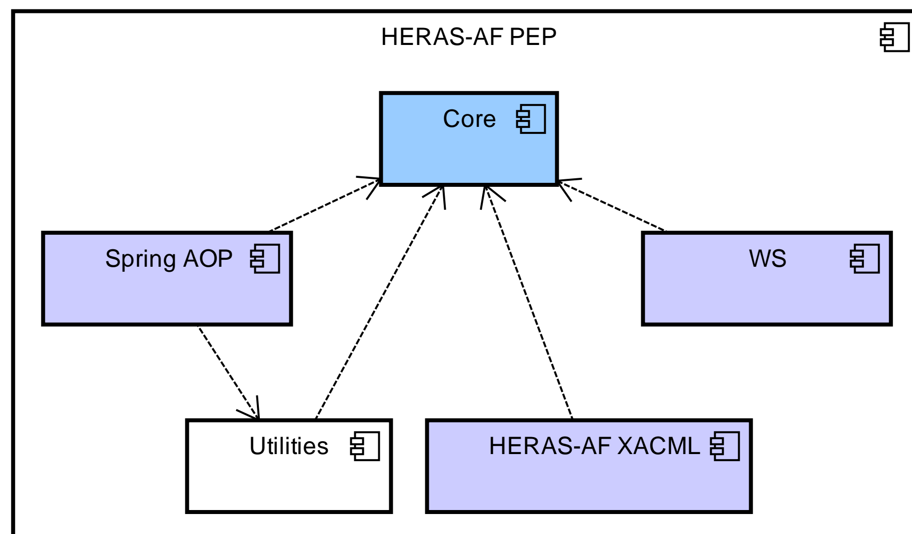



**Figure 7: Module overview**

*Module Core*

The Core module contains the structural models, the core functionality as well as the HERAS$^{AF}$ PEP logic. It provides various interfaces/classes for an integration of an intercepting technology and/or XACML implementation.

*Module Spring AOP*

This module contains the Spring AOP intercepting technology, the resolvers and the Spring AOP context. The resolvers are used to gather the data from the

annotated elements. This data is kept in the Spring AOP context which is used to create a request.

*Module Herasafxacml*

The Herasafxacml module contains the integration of the PEP and the specific HERAS^AF XACML 2.0 implementation. The transformation from the HERAS^AF PEP data structure into a XACML 2.0 request is made here.

*Module Utilities*

The Utilities module contains general annotation collecting and gathering classes. Those are used to collect annotations on classes and interfaces. An annotation cache algorithm for optimized annotation collecting is contained as well.

*Module WS*

The WS module contains the classes to connect the HERAS^AF PEP to a PDP Web Service. This includes a SAML Handler and some classes to guarantee the SAML over SOAP standard-conformance.

### 1.1.2 Interaction scenario

*Overview*

This chapter explains the interaction scenario of the HERAS^AF. It gives an overview of the most important parts in the process.

*Interaction scenario*

**Description:**

0. An administrator annotates a class which he wants to protect with the `@Protected` annotation. Creates a policy and deploys it to a PDP. Then a `@Protected` class method is called by a user.
1. The joint point described by the pointcut is reached and the `AccessControlAdvice` is called.
2. Within the advice a `DecisionContext` is created through the factory. All the annotated elements are resolved by the Resolvers, collected and set in the decision context object inside the factory.
3. The decide method of a `DecisionManager` is called with the decision context as parameter.
4. The `DecisionManager` creates a `RequestContext` based on the values of the decision context via a factory.
5. The factory uses an XACML implementation to transform the PEP request representation into an equal conformed XACML request.
6. A `Pdp` is located using the locate method in the `PdpLocator` and a `Pdp` instance is returned.
7. The evaluation of the XACML request is made with the evaluate method on the `Pdp` instance which returns a decision context.
8. A `DecisionHandler` handles the result (decision context) and returns an access allowance.

Note: The described flow is kept simple on purpose. In reality there is more activity involved.

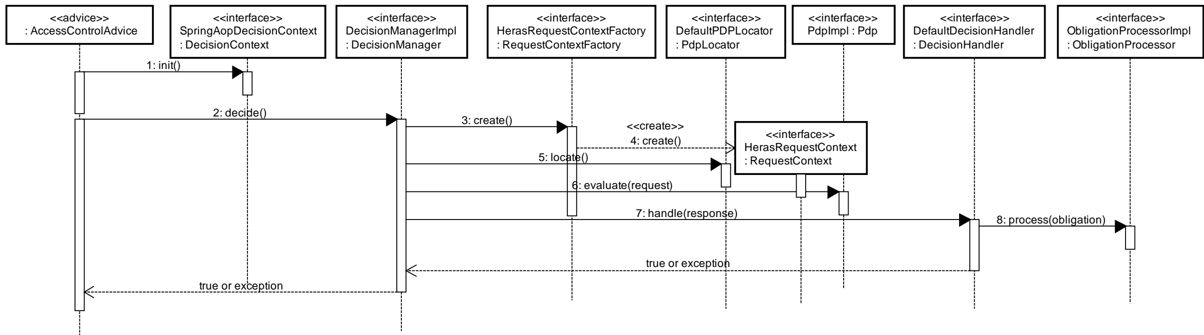



**Figure 8: Main interaction scenario**

# 2 Core

*Overview*

This chapter explains the core module.

The Core module is the heart of the HERAS$^{AF}$ PEP implementation. It contains the annotations to annotate all elements for a XACML request and the data structure to save the request as well as the response.

The module also provides various interfaces to create the integration layer for a different XACML implementation and intercepting technologies. The whole process of a PEP on how to initiate and handle an XACML request/response is in this module too.

## 2.1 HERAS$^{AF}$ PEP Implementation

*Overview*

This chapter explains the packages and classes of the core module.

*Package diagram*

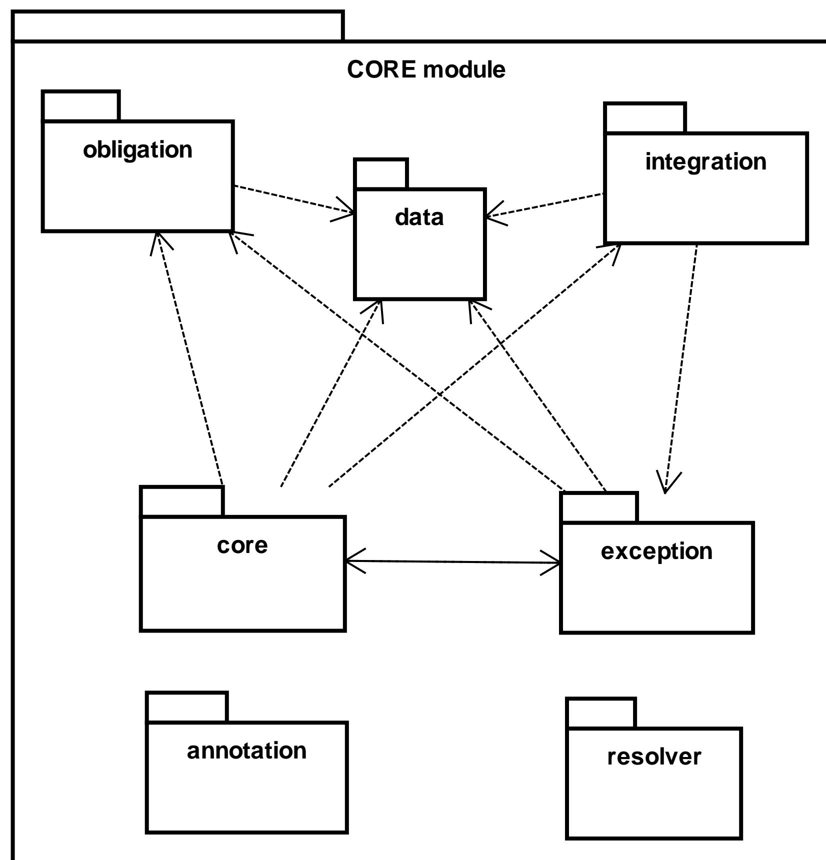



**Figure 9: Core module package diagram**

*Package description*

**annotation**

The annotation package contains all annotation to annotate the elements for a

XACML request and the annotation to protect a class.

**core**

The core package contains the core classes of the PEP. It provides default implementations of core components (e.g. `DefaultDecisionHandler`) and all component interfaces for custom implementations.

**data**

The data package contains value object classes which represent a complete XACML request and response.

**exception**

This package contains all exceptions defined by the core module of the HERAS^AF PEP.

**integration**

This package contains interfaces to create the integration of a precise PDP. It includes interfaces for a `Pdp`, `RequestContext`, `RequestContextFactory` and `ResponseContext`. Those interfaces are implemented whenever an integration of a XACML implementation is made.

**obligation**

The obligation package contains components in connection with an obligation e.g. classes and interfaces to process and handle an obligation.

**resolver**

This package contains an abstract implementation of a resolver which defines a common way to access attribute data types. A resolver is used to resolve elements on annotated classes/interfaces or resolve runtime information.

**util**

This package contains utilities for the PEP. For now it only contains constants used in the process of handling annotations.

## 2.2 Package annotation

*Overview*

The annotation package contains all annotation to annotate the elements for a XACML request and the annotation to protect a class. The most import ones are:

- `@Protected`
- `@Subject`
- `@Resource`
- `@Action`
- `@Environment`

*class diagram*

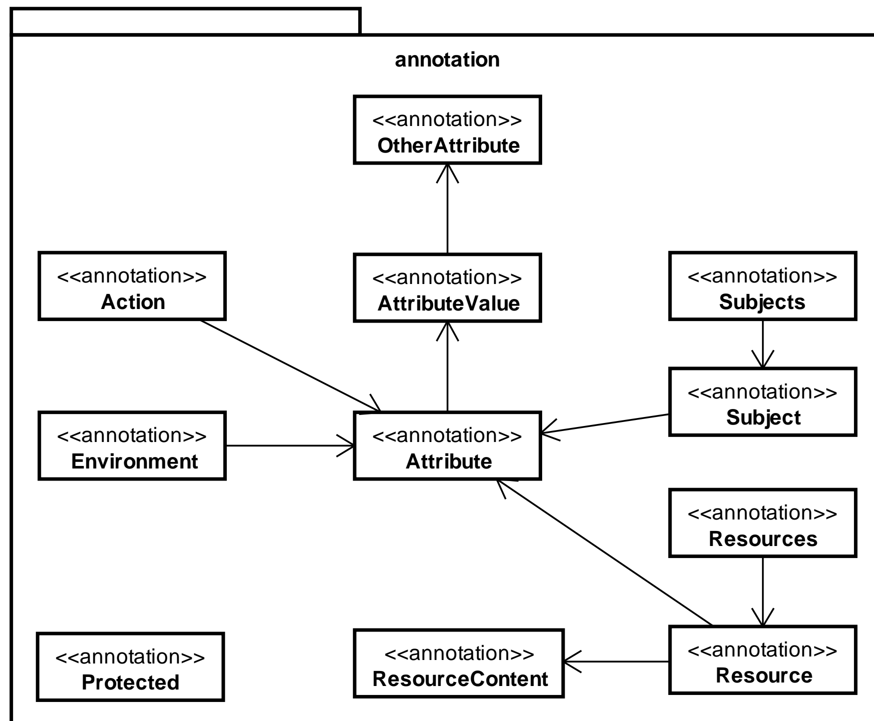



**Figure 10: Class diagram for annotation package**

## 2.3 Package core

*Overview* The core package contains the core classes of the PEP. It provides default implementations of core components (e.g. `DefaultDecisionHandler`) and contains all component interfaces for a custom implementation.

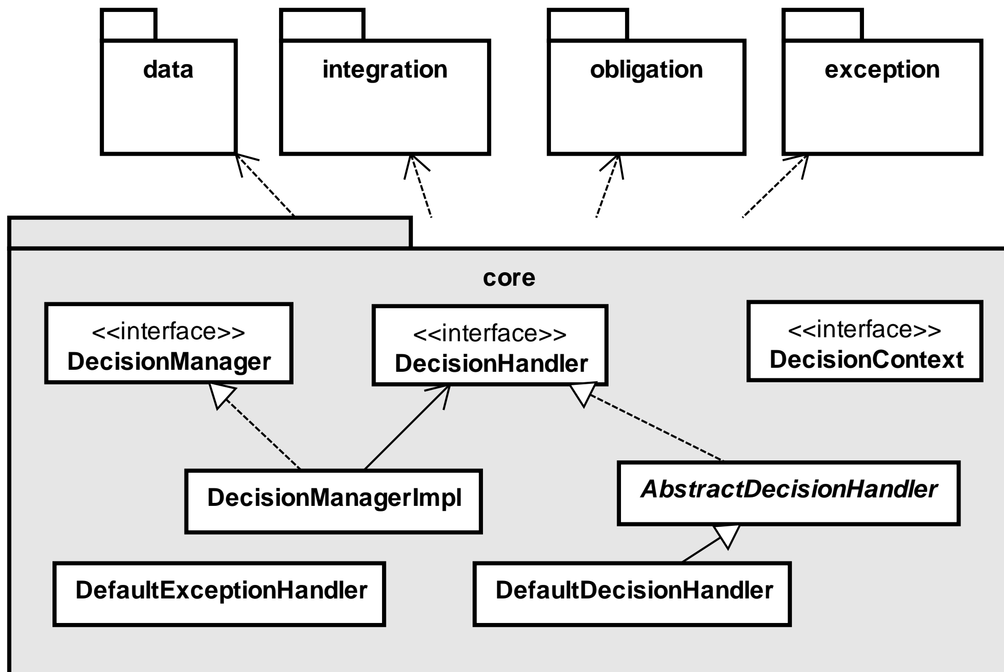



**Figure 11: Class diagram for core package**

## 2.4 Package data

*Overview* The data package contains value object classes which represent a complete XACML request and XACML response.

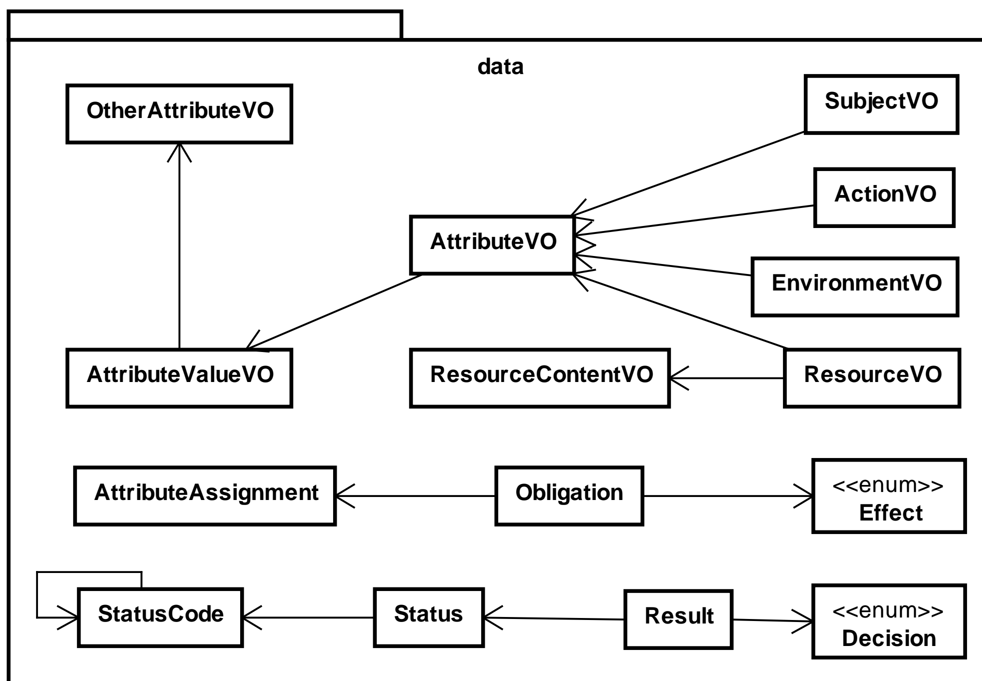



**Figure 12: Class diagram for data package**

## 2.5 Package exception

*Overview*

The exception package contains `Exception` classes. See the chapter 6.1 for more information about `Exceptions` of the HERAS^AF PEP.

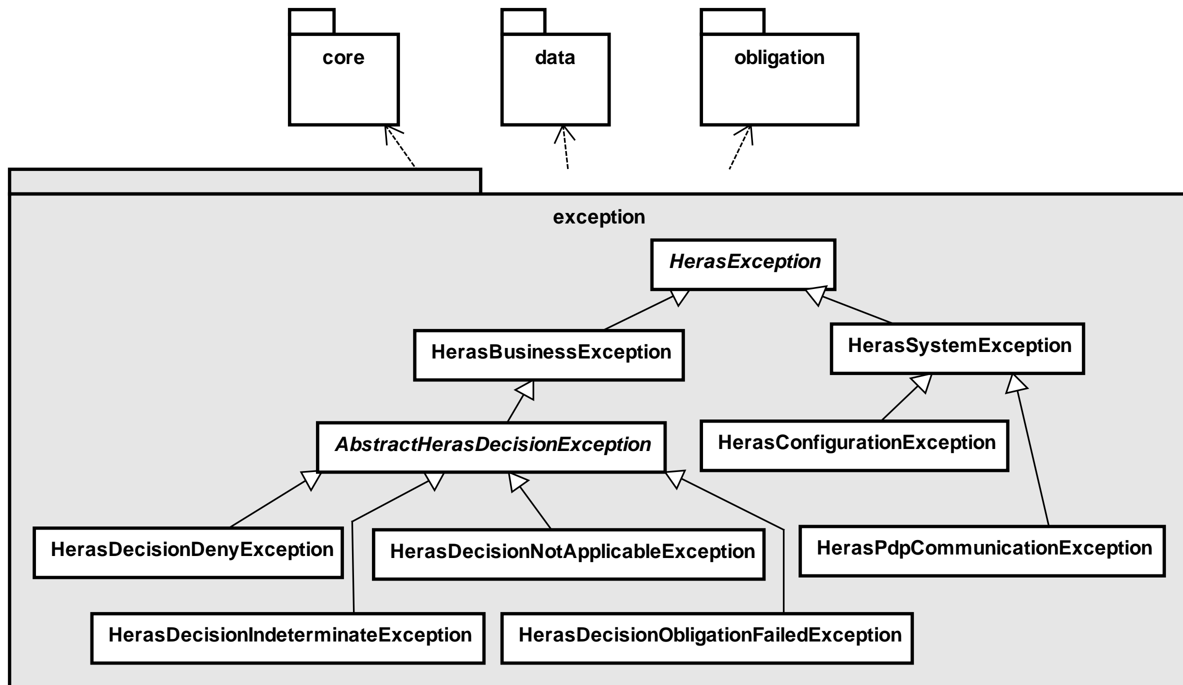



**Figure 13: Class diagram for exception package**

## 2.6 Package integration

*Overview*

This package contains interfaces to create the integration of a precise PDP. It includes interfaces for a `Pdp`, `RequestContext`, `RequestContextFactory` and `ResponseContext`. Those interfaces must be implemented whenever an integration of a XACML 2.0 implementation is done.

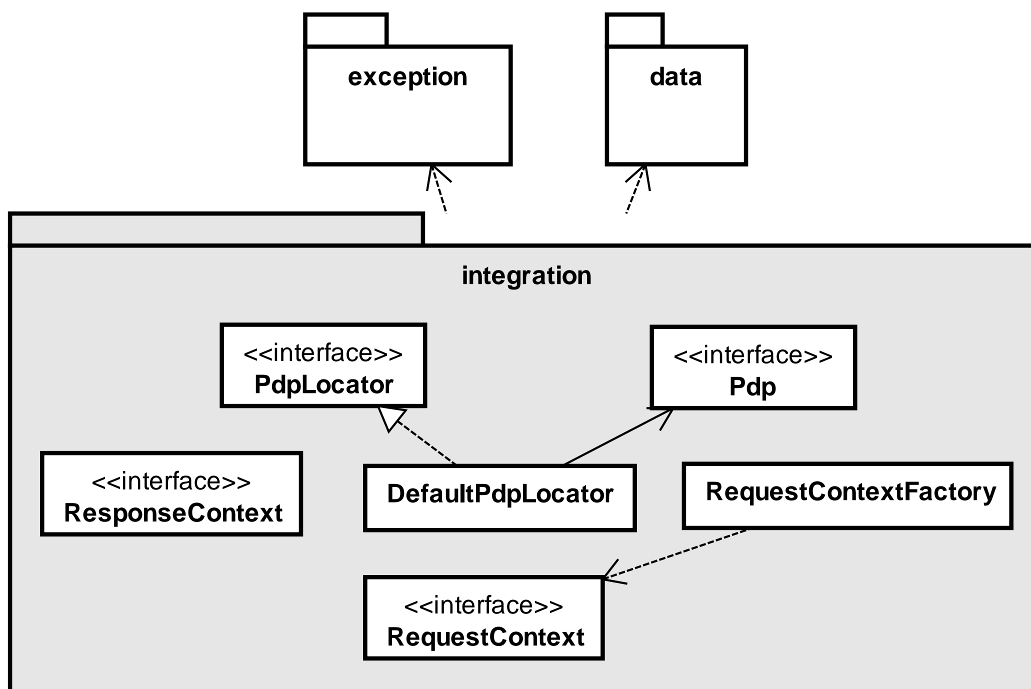



**Figure 14: Class diagram for integration package**

## 2.7 Package obligation

*Overview*

The obligation package contains components in connection with an obligation. It includes classes and interfaces to process and handle an obligation.

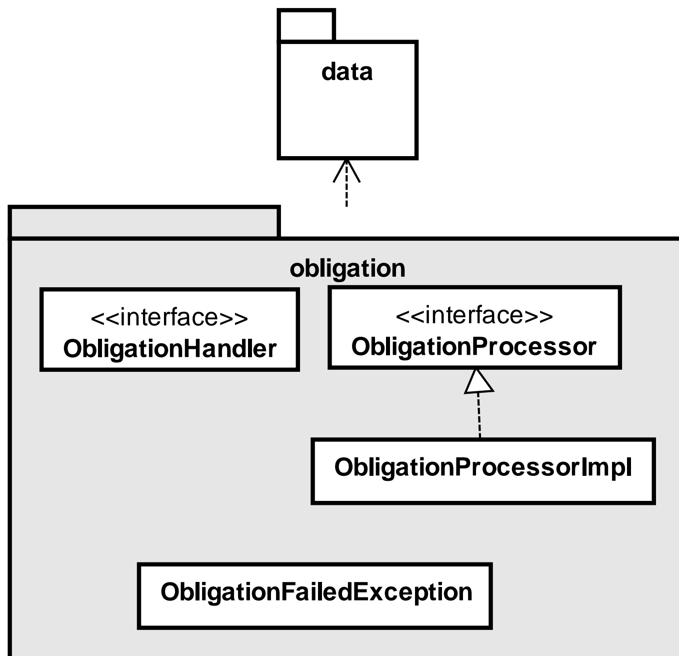



**Figure 15: Class diagram for obligation package**

## 2.8 Package resolver

*Overview*

This package contains an abstract implementation of a resolver which defines a common way to access attribute data types. In general a resolver is used to resolve elements on annotated classes/interfaces or resolve runtime information.

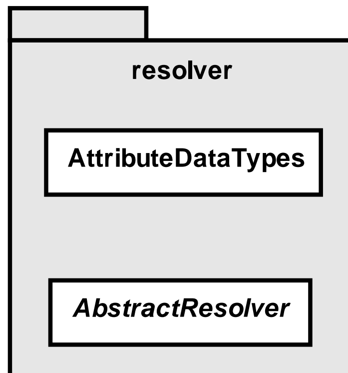



**Figure 16: Class diagram for resolver package**

## 2.9 Package util

*Overview*

This package contains utilities for the HERAS^AF PEP. By now it only contains constants used in the process handling annotations.

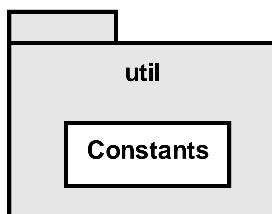



**Figure 17: Class diagram for util package**

## 2.10 Interfaces / Abstract classes

*Overview*

This chapter explains the important interfaces and abstract classes inside the core module which are used to extend or change the implementation behavior of the HERAS^AF PEP. The structure and some of the important methods are described. All other methods descriptions are in the javadoc.

### 2.10.1 AbstractDecisionHandler

| *AbstractDecisionHandler* |
| --- |
| |
| + handle(decisionContext : DecisionContext, responseContext : ResponseContext) : boolean<br>*# handleResult(result : Result, decisionContext : DecisionContext) : boolean*<br>*# handleFailedObligation(result : Result, decisionContext : DecisionContext, exception : ObligationFailedException) : void*<br>+ setObligationProcessor(obligationProcessor : ObligationProcessor) : void |

*Description*

The `AbstractDecisionHandler` class is a basis implementation of a `DecisionHandler`.

It provides a method for the handling process of a decision authorization returned by a PDP. This template method makes sure that optional contained Obligations are processed by an `ObligationProcessor` and guarantees that the Result is delegated to the `handleResult` template method.

### 2.10.2 AbstractHerasDecisionException

| *AbstractHerasDecisionException* |
| --- |
| - resourceId : String |
| + AbstractHerasDecisionException(resourceId : String, status : Status)<br>+ getResourceId() : String<br>+ getStatus() : Status |

*Description*

The `AbstractHerasDecisionException` class is a basis implementation of a `HerasDecisionException`.

This class defines two attribute `ResourceId` and `Status` which are used to set information about the reason why the exception was raised.

### 2.10.3 DecisionContext

| <<interface>><br>**DecisionContext** |
| --- |
| |
| *+ getSubjects() : List<SubjectVO>*<br>*+ getResources() : List<ResourceVO>*<br>*+ getAction() : ActionVO*<br>*+ getEnvironment() : EnvironmentVO* |

*Description*

The `DecisionContext` interface defines methods for a specific

implementation of a DecisionContext.

Implementations are expected to return SubjectVO, ResourceVO, ActionVO and EnvironmentVO objects which are the basis of an access request.

### 2.10.4 DecisionHandler

| <<interface>><br>**DecisionHandler** |
| --- |
| |
| *+ handle(decisionContext : DecisionContext, responseContext : ResponseContext) : boolean* |

*Description*

The DecisionHandler interface defines a method for a specific implementation of a DecisionHandler.

The AbstractDecisionHandler implements this interface.

This interface can be used if the handling process of a decision returned by the PDP should be handled different then the handling process of the AbstractDecisionHandler.

### 2.10.5 DecisionManager

| <<interface>><br>**DecisionManager** |
| --- |
| |
| *+ decide(context : DecisionContext) : boolean* |

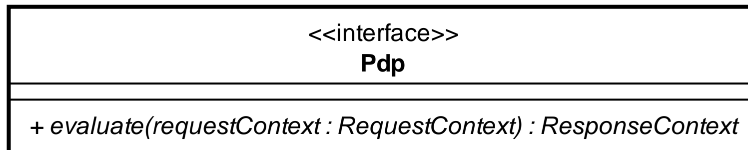
*Description*

The DecisionManager interface defines a method for a specific implementation of a DecisionManager.

The DecisionManagerImpl implements this interface and provides a default implementation of a DecisionManger.

This interface can be used if the deciding process based on a DecisionContext needs to be customized.

### 2.10.6 Pdp



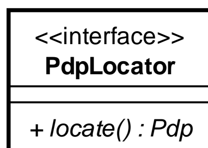

*Description*

The `Pdp` interface defines a method for a specific implementation of a PDP.

A PDP is responsible to evaluate requests and grant access based on stored policies.

This interface is used whenever the PEP is connected with a PDP.

### 2.10.7 PdpLocator

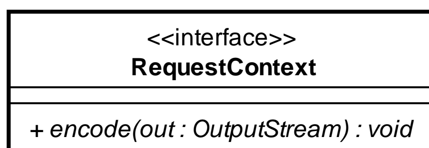



*Description*

The `PdpLocator` interface defines a method for a specific implementation of a PDP locator.

A `PdpLocator` is used whenever a user wants to get a PDP without the knowledge of where a PDP is located. The responsibility to locate a PDP is fully taken by the `PdpLocator`.

### 2.10.8 RequestContext




*Description*

The `RequestContext` interface defines a method for specific implementation of a Request Context.

A `RequestContext` encapsulates the authorization request. The contained information is collected by the PEP and created through the `RequestContextFactory`.

The `encode` method encodes the decision request to an XACML output stream.

### 2.10.9 RequestContextFactory

| <<interface>><br>**RequestContextFactory** |
| --- |
| |
| *+ create(sub : List<SubjectVO>, res : List<ResourceVO>, act : ActionVO, env : EnvironmentVO) : RequestContext* |

*Description*

The `RequestContextFactory` interface defines a method for specific implementation of this factory.

A `RequestContextFactory` is responsible to create a `ResponseContext` on basis of subject-, resource-, action- and environment information.

### 2.10.10 ResponseContext

| <<interface>><br>**ResponseContext** |
| --- |
| |
| *+ getResults() : List<Result>* |

*Description*

The `ResponseContext` interface defines a method for a specific implementation of a ResponseContext.

A `ResponseContext` encapsulates the authorization decision returned by a PDP. An XACML authorization decision contains one or more `Results` (for each requested `Resource`).

### 2.10.11 ObligationHandler

| <<interface>><br>**ObligationHandler** |
| --- |
| |
| *+ handle(obligation : Obligation) : void* |

*Description*

The `ObligationHandler` interface defines a method for a specific implementation of an ObligationHandler.

An `ObligationHandler` handles a specific obligation. For example an ObligationHandler which does a database query.

### 2.10.12 ObligationProcessor

| <<interface>><br>**ObligationProcessor** |
| --- |
| |
| *+ process(obligations : List<Obligation>, decision : Decision) : void* |

*Description*

The `ObligationProcessor` interface defines a method for a specific implementation of an ObligationProcessor.

An `ObligationProcessor` processes a list of Obligations and contains a set of `ObligationHandler` which handles the Obligations.

The `ObligationProcessorImpl` implements this interface and provides a default implementation of an Obligation Processor.

## 2.11 Scenarios

*Overview*

This chapter explains the most important sequences inside the core module.

### 2.11.1 Decision Handling

*General*

This part explains the Decision Handling flow.

There are basically two different cases which can occur. A decision contains a permit-decision or a deny-decision whereas the deny-decision is spilt up in 3 possibilities DENY, INAPPROPRIATE or NOTAPPLICABLE. The deny-decision always throws a `HerasException`.

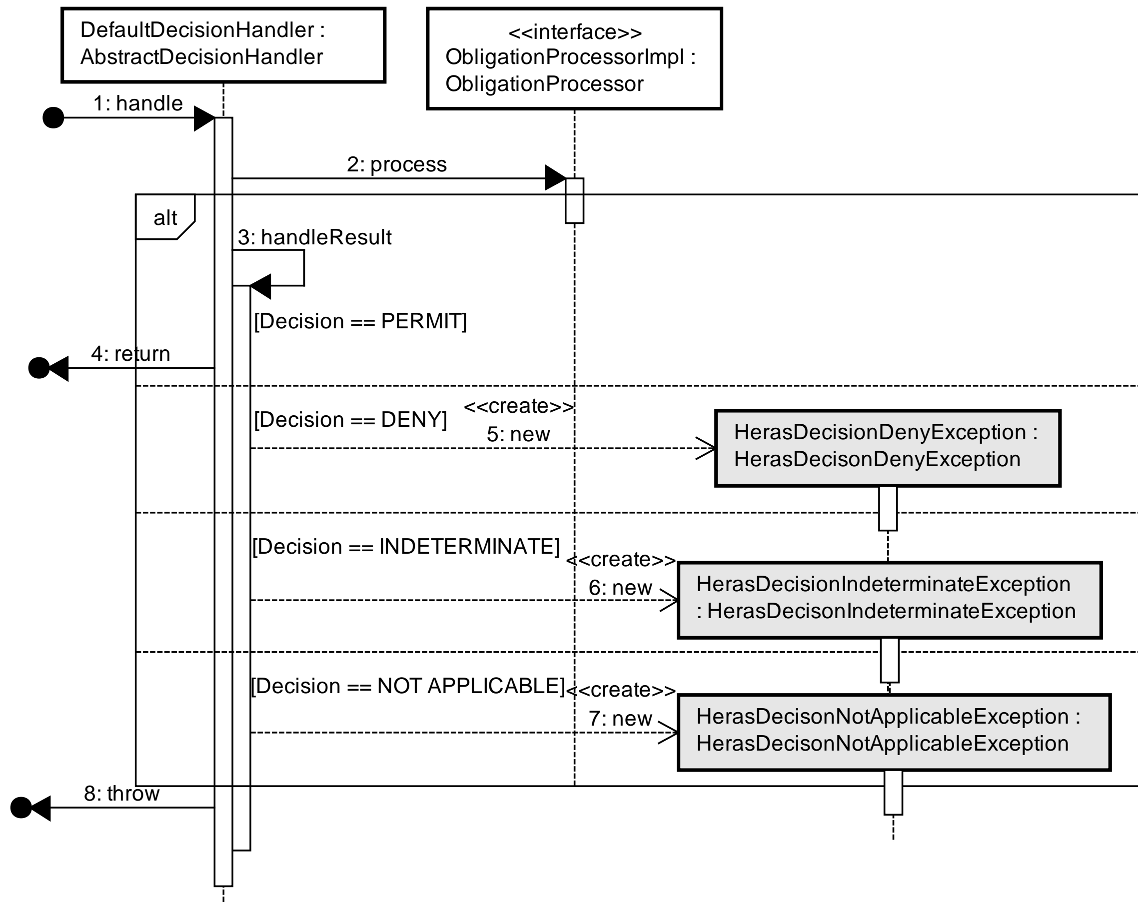



**Figure 18: Decision Handling flow**

*Decision handling description*

1. The `DecisonHandlers handle()` is called by the `DecisionManager`.
2. The Obligation processing is done (see chapter 0).
3. The `DecisonHandlers handleResult()` is called by the `DecisionManager`.
4. If the `Decision` is PERMIT then true is returned.
5. If the `Decision` is DENY then a new `HerasDecisionDenyException` is created and thrown.
6. If the Decision is INDETERMINATE then a new `HerasDecisionIndeterminateException` is created and thrown.
7. If the Decision is NOTAPPLICABLE then a new `HerasDecisionDenyException` is created and thrown.

## 2.11.2 Obligation Handling

*Overview*

This part explains the Obligation Handling flow.

There are basically two different cases which occur. An obligation can be successfully handled or an obligation cannot be handled – the error case. These two are described in the following sequence diagrams.

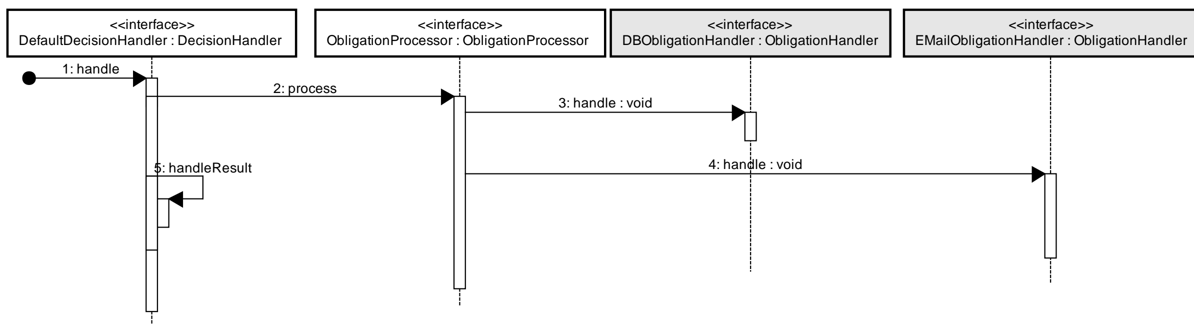



**Figure 19: Successful Obligation handling sequence diagram**

*Successful Obligation handling*

The XACML response can have 0-n Obligations. In this example it has 2 Obligations to handle (DB- and Email Obligation) and all `ObligationHandlers` are correctly mapped.

0. An XACML request is sent to a PDP and the XACML response is received by the `DecisionManager`.
1. The `DecisionHandlers handle()` is called by the `DecisionManager`. Pre-processing of the response is done (has nothing to do with the Obligation Handling).
2. The `ObligationProcessors process()` is called with the obligations contained in the XACML response. The `ObligationProcessor` has a map with `ObligationHandlers`
3. The `ObligationProcessor` does a lookup in the map with the Obligation-Id and sends the Obligation to handle to the found `ObligationHandler.`
4. The `ObligationProcessor` does a lookup in the map with the Obligation-Id and sends the Obligation to handle to the found `ObligationHandler.`
5. The `DecisionHandler` continues with handling the results which is contained in the XACML response. Obligation handling process had successfully ended.

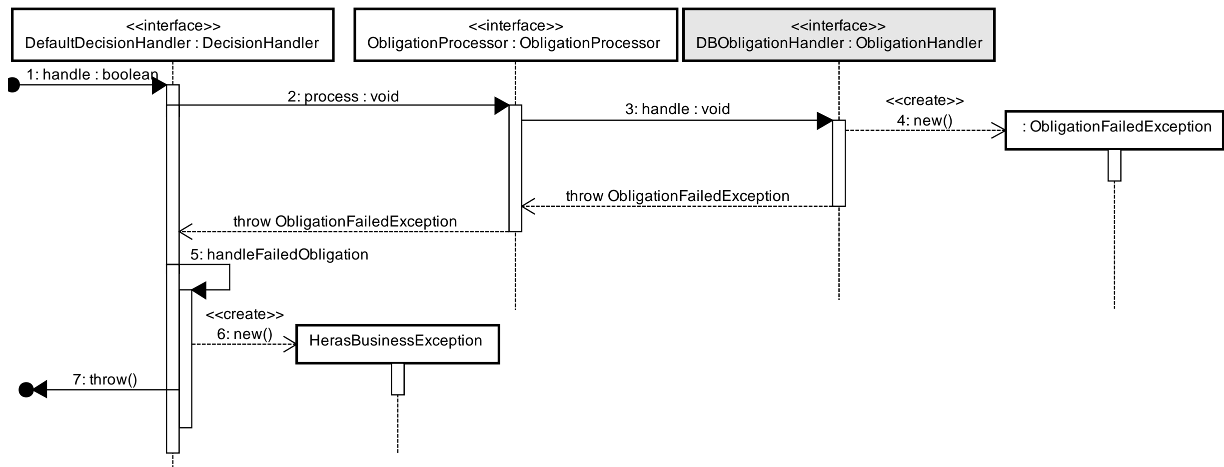



**Figure 20: Failed Obligation handling sequence diagram**

*Failed Obligation Handling*

The XACML response can have 0-n Obligations. In this example it has 2 Obligations to handle (DB- and Email Obligation) and all `ObligationHandlers` are correctly mapped.

0. An XACML request is sent to a PDP and the XACML response is received by the `DecisionManager`.
1. The `DecisionHandlers handle()` is called by the `DecisionManager`. Pre-processing of the response is done.
2. The `ObligationProcessors process()` is called with the obligations contained in the XACML response. The `ObligationProcessor` has a map with `ObligationHandlers`
3. The `ObligationProcessor` does a lookup in the map with the Obligation-Id and sends the Obligation to handle to the found `ObligationHandler.`
4. The `ObligationHandler` can't handle the Obligation and throws an ObligationFailedException.
5. The `DecisionHandler` catches the `ObligationFailedException` and calls the `handleFailedObligation().`
6. The `hanledFaileObligation()` takes the `ObligationFailedException` and encapsulates the exception inside a `HerasBusinessException` which is thrown afterwards.

## 2.12 Design decisions

### 2.12.1 Attribute data types

*Overview*

Attributes resolved from request information sources can be of different data types.

Sending a decision request to a PDP requires the PDP to know how to handle the attributes in the request. This is achieved by deploying handlers for each required attribute data type on the PDP.

The PEP must be able to handle all kind of data types, yet there are different alternatives how this can be achieved.

*Alternatives*

**Alternative 1: All data types are allowed.**

The PEP allows all attribute data types and doesn't verify the attribute's content. The only requirement to all attributes is that they implement the `toString` method to return their XML representation, to be written into the decision request.

| Advantage | Disadvantage |
|---|---|
| • Simple design and low costs.<br>• The PEP can be used with any kind of attribute data type. | • The user is responsible to only use data types that are accepted by the PDP.<br>• All data types need to implement the conversion to the XML form directly in their `toString` method. |

**Alternative 2: Injection of attribute data type handlers.**

Handler for all supported attribute data types are injected into the PEP. These handlers allow transforming a certain type of objects into their XML form as String. If a handler for a data type is not found, an exception is thrown and access is denied.

| Advantage | Disadvantage |
|---|---|
| • The conversion between an object of a certain data type into its XML form is separated from the object itself.<br>• The handler can still use the `toString` method, if this approach is preferred or necessary. | • The set of data type handlers needs to be synchronized manually with the set of data types accepted by the PDP. |

**Alternative 3: Dynamic supply of attribute data type handlers.**

Handler for all supported attribute data types are provided dynamically by a data type handler locator.

Depending on the type of locator, the data type handlers could be injected

directly into the locator or gathered from another source.

A possible source could be the PDP where the decision request is supposed to be sent to.

| Advantage | Disadvantage |
|---|---|
| • Allows obtaining data type handlers from any possible source in a dynamic way. | • Complex design in association with the PDP locator. |

*Decision*

The chosen solution is that all data types are allowed (alternative 1). We chose this solution because of the simple design and the low cost of time. We also thought about the fact that the person who does the configuration of the PEP is aware of what data types are used. So he knows if the data types implement the `toString` method correctly or not.

### 2.12.2 Finding/locating a PDP

*Overview*

The PEP needs a PDP to evaluate its XACML request but the PEP should not know where the PDP is located or which implementation of a PDP he is using (e.g. a real PDP or a network client to a remote PDP).

*Alternatives*

**Alternative 1: Integration with locator pattern**

A PDP locator is injected into the PEP that can be used to locate a PDP. The PDP locator can be implemented with special behavior. For example with a specialized PDP finding algorithm.

| Advantage | Disadvantage |
|---|---|
| • PEP doesn't know which PDP is given to him<br>• Multiple PDPs could be configured within the PDP locator. | • Users implementing a simple PDP without special locating behavior would still need to use and understand the locator pattern. |

**Alternative 2: Direct injection of PDPs**

PDPs are directly injected into the PEP.

If special behavior is required (e.g. load balancing or using a backup PDP), this functionality would need to be integrated into the PDP implementation.

| Advantage | Disadvantage |
|---|---|
| • PEP doesn't know which PDP is given to him<br>• Multiple PDPs could be configured within the PDP implementation.<br>• Users implementing a simple PDP without special behavior | • No separation of concerns, the functionality of locating the right PDP is hidden within the PDP implementation. |

| | |
|---|---|
| | are not bothered by the concept of locating a PDP. |

*Decision* The chosen solution is to create a PDP locator (alternative 1). We chose this solution because of the separation of concern. The functionality of locating the right PDP should not be the task of the PEP. The PEP just wants to use a PDP and does not care where it is located or what kind of implementation he uses.

### 2.12.3 XACML response

*Overview* The authorization decision received from the PDP in the form of a XACML response must be transformed into a readable and accessible read-only data structure at the PEP. This data structure is then used for the decision handling. See [XACML] for further information about the content of an XACML Response.

*Alternatives*

**Alternative 1: Use the JAXB generated data structure from the HERAS^AF XACML implementation [DOH07DA]**

The PEP could directly use the XACML response data structure of the HERAS^AF XACML implementation or regenerate the data structure with JAXB 2.0 from XSD files.

| Advantage | Disadvantage |
|---|---|
| • Minimal costs | • Dependency to JAXB and the XACML implementation<br>• Not read-only. |

**Alternative 2: Create own data structure**

Create an own data structure and map XACML responses into it.

| Advantage | Disadvantage |
|---|---|
| • Some XACML elements can be simplified or ignored if not needed. (e.g. enumerations)<br>• Independent from the XACML implementation. | • Higher costs |

*Decision* The chosen solution is to implement an own data structure for the PEP only (alternative 2). The main reason this solution was chosen is that there should not be any dependencies to JAXB and the HERAS^AF XACML 2.0 implementation in the core module.

### 2.12.4 Decision handling

*Overview* Decision handlers are responsible to handle the XACML responses returned by a PDP. See [XACML] for further information about the elements of such a

response.

Based on the content of an XACML response, access to a resource can be permitted or denied. To do this, the interceptor needs to be informed if the execution of an action should be continued or aborted. It must also be decided, which possibilities the application protected by HERAS^AF PEP should have to handle the response.

*Alternatives*

**Alternative 1: Throwing Runtime Exception**

The decision handler throws a specialized Runtime Exception, if the decision is not "permit". Whenever the decision is "permit" the decision handler simply returns true, as there is no need to know why it was permitted.

The Runtime Exception contains the additional information (e.g. status, indicating why it was denied) and will be thrown up to the level of the application protected by the PEP.

| **Advantage** | **Disadvantage** |
|---|---|
| • Fast way to go through the long program stack hierarchy back to the application level.<br>• The exception contains all decision information so that it is accessible at the application level.<br>• Applications can define a common way how to handle this specific runtime exceptions (e.g. in a web application define a rule for this exception).<br>• Possibility to define own Runtime Exceptions for easier integration into an existing application. | • If access is not permitted, there is always an exception thrown. If an application wants a silent deny (no exception, the called method is just returned with a null value), this alternative is not viable. |

**Alternative 2: Return simple Boolean value**

The decision handler just returns a Boolean value to indicate if the execution of an action should be continued or aborted.
(True, if the decision of a response is "PERMIT", false otherwise.)

The decision handler itself is responsible to handle the response information before returning the Boolean value.

| **Advantage** | **Disadvantage** |
|---|---|

| | |
|---|---|
| • Simple way to handle decisions. | • The information about a decision is lost, if not handled by the decision handler.<br>• Integration into existing systems is not possible, if it depends on exceptions.<br>• A common way to handle deny-decisions is harder to achieve. |

*Decision*

The chosen solution is to implement the decision handling in connection with runtime exceptions (alternative 1). The decision fell on this alternative because it is the fastest way to go through the long program stack hierarchy back to the application level and we can contain information into the exceptions (e.g. why it failed).

The application itself can define a common way to handle the exception and doesn't need additional custom handlers.

A big advantage of this solution is that the integration of the PEP into a legacy system with predefined runtime exceptions is possible. The person who does the configuration of the PEP can choose to use own exception for the handling process.

### 2.12.5 Obligation handling

*Overview*

The response generated by the PDP can contain optional obligations. The final decision depends on these obligations. The PEP must make sure that all the obligations are handled correctly. If an obligation cannot be handled or fulfilled, the PEP must change the decision accordingly.

All the obligations are company/application-specific (e.g. database queries, sending an email), thus the PEP must provide an interface for the injection of for custom obligation handlers.

*Alternatives*

**Alternative 1: Default obligation processor with configurable handlers**

Obligations in a response are handled by a default obligation processor. Custom obligation handlers are injected into a list of handlers that can be used by the obligation processor.

The application flow of handling a response would then be:

1. Get the response from the PDP.
2. Check for obligations. If obligations are found process those with the obligation processor.
    a. For each obligation the processor finds the correct custom handler.
    b. The custom handler handles the obligation.
    c. The result of the handler is processed by the obligation processor.
3. Make the final decision based on decision and the overall result of

the obligations processor.

| Advantage | Disadvantage |
|---|---|
| • Obligations are always handled (not possible to ignore obligations at all).<br>• Easy configuration of handlers with spring.<br>• Because of a default implementation the user doesn't have to implement an obligation processor. | • The user can still implement an "all-matcher-obligator-handler" which returns always a true even if the obligation could not be fulfilled. |

**Alternative 2: Delegate obligation handling to decision handlers.**

The handling of obligations is delegated to the decision handler.

| Advantage | Disadvantage |
|---|---|
| • Flexibility<br>• The user can implement decision handlers that ignore obligations or handle them in a special way.<br>• Within the decision handler, obligations could still be processed by a default obligation processor and benefit from the easy configuration of handlers with spring. | • With a decision handler that ignores obligations, the PEP would violate its specified behavior. |

*Decision*

The chosen solution is to provide a default obligation processor with a possibility to configure handlers (alternative 1). The decision fell on this alternative because it is not possible to ignore the obligation processing. The person who does the configuration of the PEP can simple add new obligation handlers to the obligation processor and doesn't need to care about the whole process itself.

# 3 Spring AOP

## 3.1 HERAS^AF PEP Implementation

*Overview*

This chapter explains the packages and classes of the springaop module.

The module contains an integration implementation of the Spring AOP intercepting technology to intercept a method invocation. The implementation of the pointcut definition when a class is protected and the entry point of the access control are done here. The included resolvers are responsible to resolve the annotated elements and provided the information to the decision context.

*Package diagram*

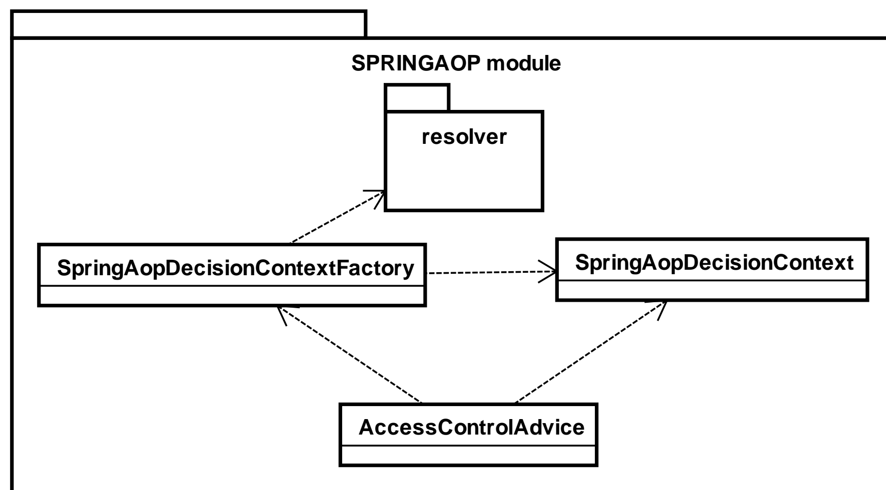



**Figure 21: Spring AOP module package diagram**

*Package description*

**(default) springaop**

This package contains the classes for the Spring AOP intercepting technology.

**resolver**

This package contains the implementation of each resolver for the Spring AOP interceptor technology.

## 3.2 Package springaop

*Overview*

The springaop package contains the classes for the Spring AOP intercepting technology. This includes a pointcut, classfilter, advice and the specific `SpringAopDecisionContext` as well as the factory.

This is a start point of the HERAS^AF PEP. The interception of protected class and the initiation of the decision process are done here.

The resolver sub package contains the implementation of each resolver in connection with Spring AOP. A resolver is used to resolve elements on annotated classes/interfaces or resolve runtime information.

For example a resolver to resolve resource elements is contained in this

package (`AnnotationSpringAopResourceResolver`).

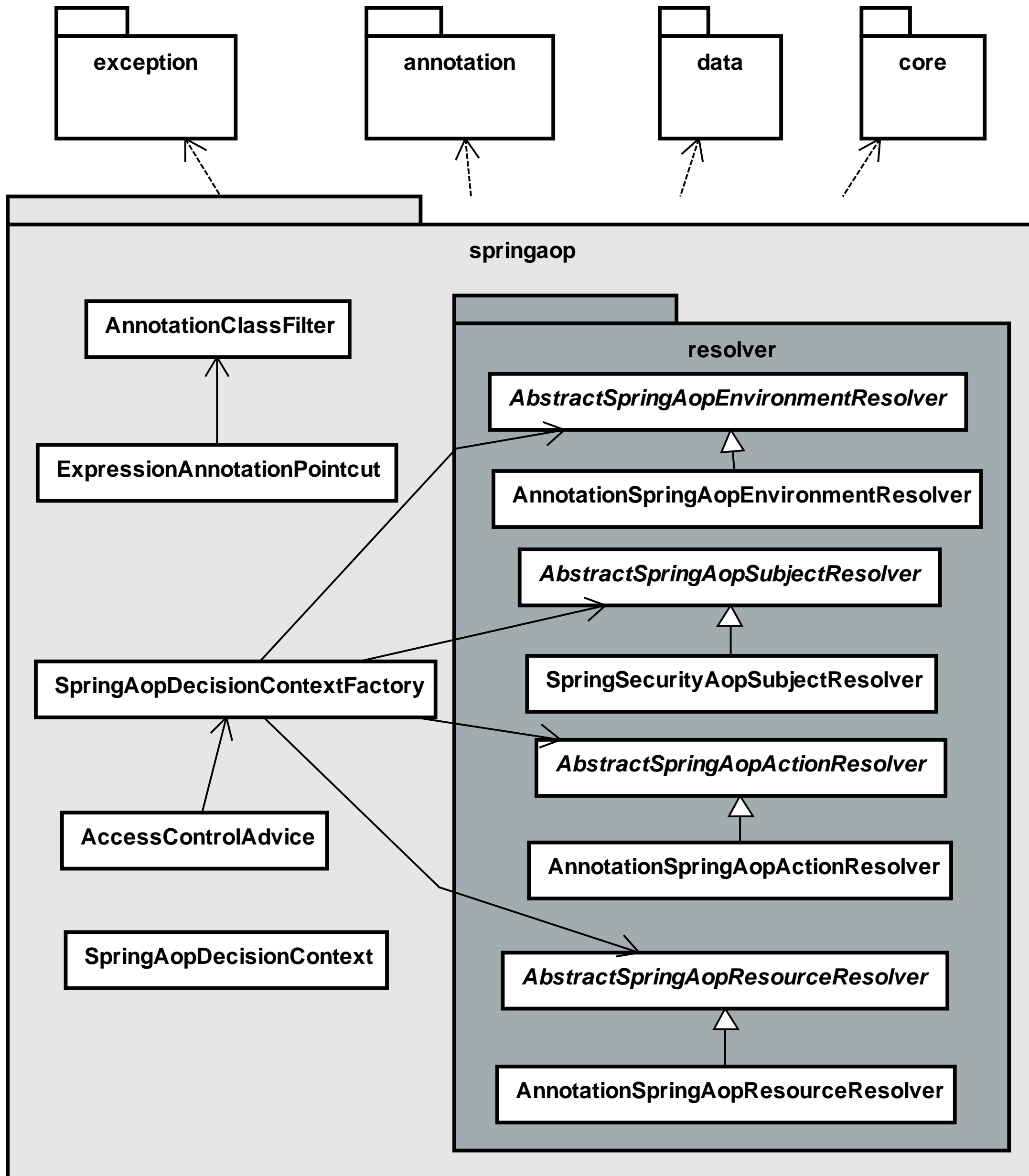



**Figure 22: Class diagram for springaop package**

## 3.3 Interfaces / Abstract classes

*Overview*

This chapter explains the important interfaces and abstract classes inside the springaop module which are used to extend or change the implementation behavior of the PEP. The structure and some of the important methods are described.

### 3.3.1 AbstractSpringAopXXXResolver

| ***AbstractSpringAopResourceResolver*** |
|---|
| |
| *+ resolve(proceedingJoinPoint : ProceedingJoinPoint) : List<ResourceVO>* |

| ***AbstractSpringAopSubjectResolver*** |
|---|
| |
| *+ resolve(proceedingJoinPoint : ProceedingJoinPoint) : List<SubjectVO>* |

| ***AbstractSpringAopEnvironmentResolver*** |
|---|
| |
| *+ resolve(proceedingJoinPoint : ProceedingJoinPoint) : EnvironmentVO* |

| ***AbstractSpringAopActionResolver*** |
|---|
| |
| *+ resolve(proceedingJoinPoint : ProceedingJoinPoint) : ActionVO* |

*Description*

The `AbstractSpringAopResourceResolver`, `AbstractSpringAopSubjectResolver`, `AbstractSpringAopEnvironmentResolver` and `AbstractSpringAopActionResolver` abstract classes define common behavior for Spring AOP resolvers. Each resolver for his own type (`Resource`, `Subject`, `Environment` and `Action`).

An implementation for each of these is provided on basis of Java Annotations and Spring Security (`Subject` only).

## 3.4 Scenario

*Overview*

This chapter explains the most important sequences inside the Spring AOP module.

### 3.4.1 Intercepting

*General* This part explains the Decision Handling flow.

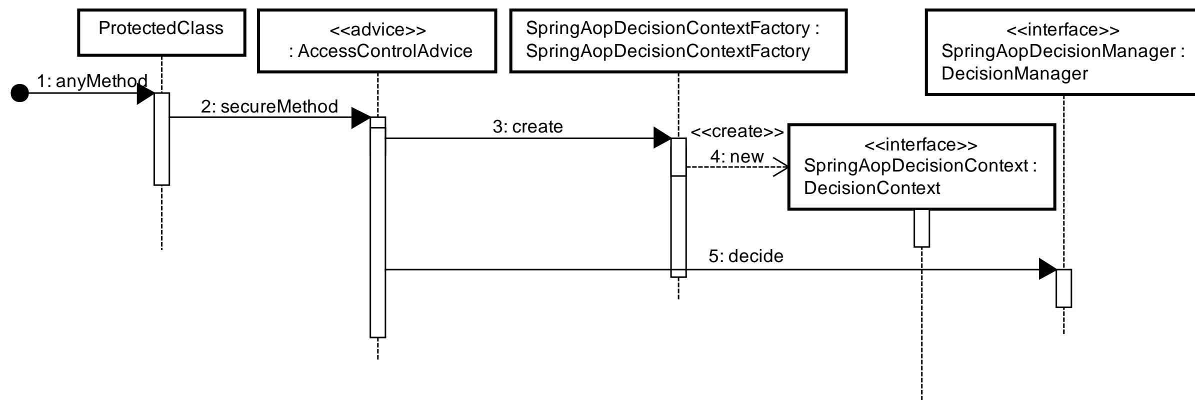



**Figure 23: Decision handling flow**

*Decision handling flow*

1. The Application calls the `anyMethod()` on the `ProtectedClass`. The `ProtectedClass` is annotated with `@Protected` annotation.
2. The join point in the pointcut definition is reached and the associated `AccessControlAdvice secureMethod()` is called.
3. The AccessControlAdvice invokes the `SpringAopContextFactory create()`.
4. A new instance of a `SpringAopContext` is created. The creation contains the resolving of the elements with the Resolvers.
5. The `SpringAopDecisionManager decide()` is called.

## 3.5 Design decisions

### 3.5.1 Interception of a method call

*Overview* A method invocation must be intercepted by the PEP whenever the method is marked for protection by HERAS^AF.

The general technology to intercept a method call is to define an aspect with a pointcut. Various intercepting technologies like Spring AOP or AspectJ exist and they all work the same way. A pointcut definition matches a certain method call, interrupts it and calls an advice before the method call is continued. When a pointcut matches a method call, a proxy object of the called object is created and everything runs through this proxy object.

*Alternatives* **Alternative 1: Define a pointcut with Spring AOP syntax**

The definition of the pointcut is in Spring AOP syntax. There is no use to

run everything through a proxy object. Only the classes which are protected by the PEP should be run through the proxy object.

This implementation already exists from the proof of concept of the last thesis by Massimo Cerqui and Sandro Strebel. [CS07PEP]

| Advantage | Disadvantage |
|---|---|
| • A solution already exists from the proof of concept | • All method calls are made through a proxy object, because there is no possibility to define a package.<br>Makes the application slower.<br>• Configuration is very complicated to run some packages or a few classes through the proxy<br>• The user cannot cast to a concrete type anymore. Only interfaces are supported. |

**Alternative 2: Define a pointcut with Spring AOP and AspectJ syntax**

Since Spring AOP 2.0 the user can specify the pointcut definition with the same syntax as in AspectJ. In addition to that a custom `AspectJExpressionPointcut` class is provided to simple define an expression when a pointcut should match. This gives a good way to specify which classes should be run through the proxy.

An example of a bean definition for the pointcut could look like:

```
<bean id="pointcut"
            class="pointcutclass">
<property name="expression"
      value="execution(*org.herasaf.secure.*.*(..))"
/>
</bean>
```

**Example 11: Spring AOP AspectJ syntax pointcut code**

| Advantage | Disadvantage |
|---|---|
| • Configuration is simple, to run only some packages or a few classes through the proxy<br>• Only classes which are defined with the regular expression are run through the proxy<br>• The user is able to cast to concrete types. | • none |

*Decision* The chosen solution is to implement the pointcut with the AspectJ syntax (alternative 2). The decision fell on this alternative because it allows the user to simply configure which packages have classes to protect. This means that only

those classes are run through the proxy. For this reason the chosen solution has better performance.

### 3.5.2 Resolving request information

*Overview*

Request information must be resolved after intercepting the application flow. Various intercepting technologies (e.g. AspectJ, Spring AOP) and different sources for request information (e.g. code annotations, XML files or databases) could be used.

The request information should be held in a specialized decision context. This context needs to provide access to the attributes (Subject, Resources, Action, and Environment) required for a decision. The context is used to create a standard-compliant XACML 2.0 decision request.

*Alternatives*

**Alternative 1: A resolver for each interceptor technology and request information source.**

For each combination of an interceptor technology, request information source and attribute type a resolver is created.

If we have 2 interceptor technologies, 3 request information sources and the 4 XACML-attribute types, it will result in 2*3*4=24 resolvers.

Because of the lack of a common interface for all these resolvers, we need a decision context for each combination of interceptor technology and request information source. This leads to another 2*3=6 decision context classes.

Figure 24 shows an example with 1 interceptor technology (pink), 2 request information sources (green) and 4 XACML Attributes (yellow). The resulting resolver classes are grey.

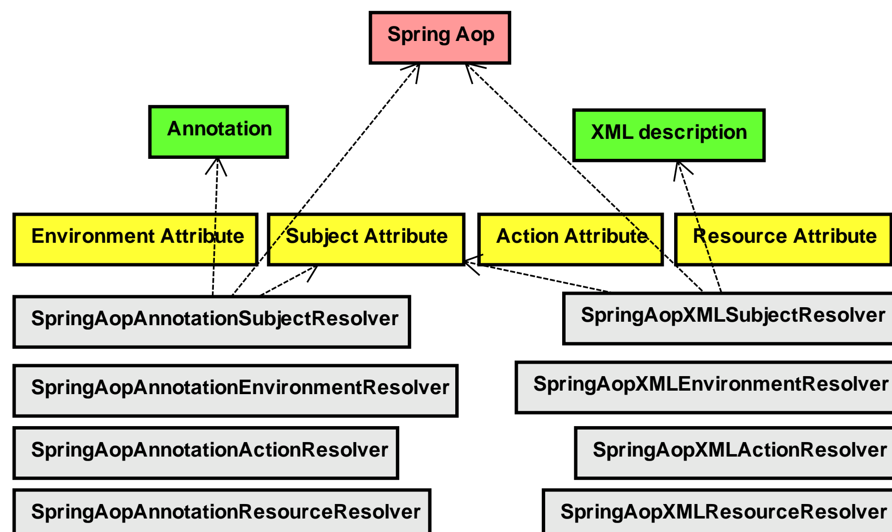



**Figure 24: Resolvers alternativ 1**

| Advantage | Disadvantage |
|---|---|
| • The user can use or create a decision context and resolvers specifically for the interceptor technology and request information source he needs. | • High implementation costs, if resolvers and decision contexts for multiple interceptor technologies and request information sources should be implemented<br>• Each decision context is doing almost the same work. |

**Alternative 2: An abstract resolver for each interceptor**

For each combination of an interceptor technology and attribute type an abstract resolver is created. For each combination of request information source and attribute type a concrete resolver is created that extends the corresponding abstract resolver.

This allows creating decision contexts that are independent from the request information source and need only to know the interceptor technology specific abstract resolvers.

If we have 2 interceptor technologies, 3 request information sources and the 4 XACML-attribute types, it will result in 2 decision contexts, 2*4=8 abstract resolvers and 3*4=12 concrete resolvers.

Figure 25 shows an example with 1 interceptor technologies (pink), 2 request information sources (grey) and 4 XACML Attributes (yellow). The resulting resolver classes are grey.

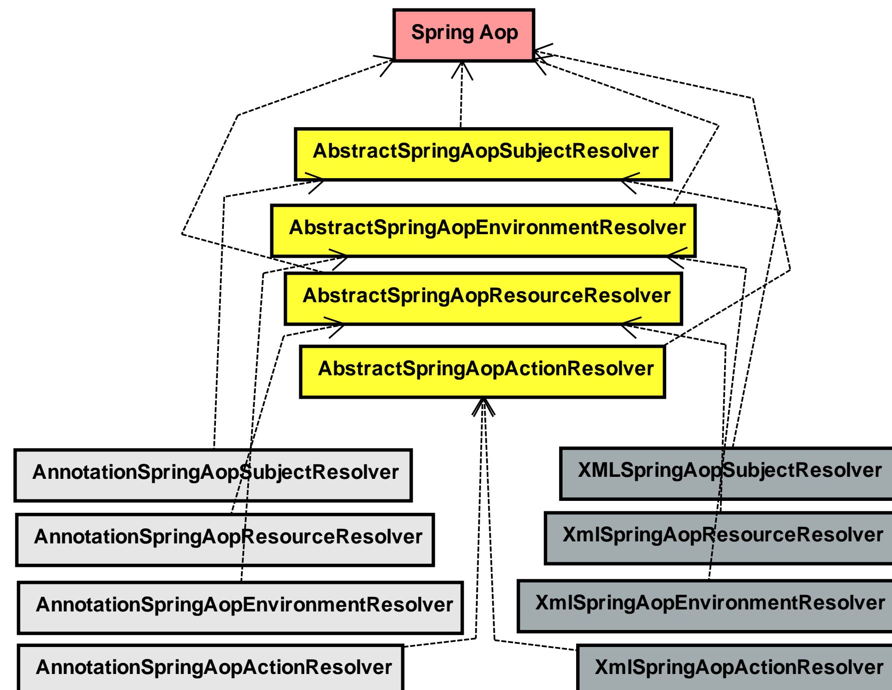



**Figure 25: Resolvers alternativ 2**

| Advantage | Disadvantage |
|---|---|
| • Decision contexts only depend on the interceptor technology.<br>• The user can use or create a decision context and resolvers, specifically for the interceptor technology and request information source he needs. | • Medium implementation costs, if resolvers and decision contexts for multiple interceptor technologies should be implemented |

*Decision*

The chosen solution is to implement an abstract resolver for each interceptor technology (alternative 2). The decision fell on this alternative because now the decision context only depends on the interceptor technology.

# 4 Herasafxacml

## 4.1 HERAS^AF PEP implementation

*Overview*

This chapter explains the packages and classes of the Herasafxacml module.

The Herasafxacml module contains an integration implementation of the HERAS^AF XACML 2.0 and the HERAS^AF PEP implementation.

This includes the creation of the request context and the transformation of the data structure used intern by the HERAS^AF PEP implementation into the data structure used by the HERAS^AF XACML 2.0 implementation.

*Package diagram*

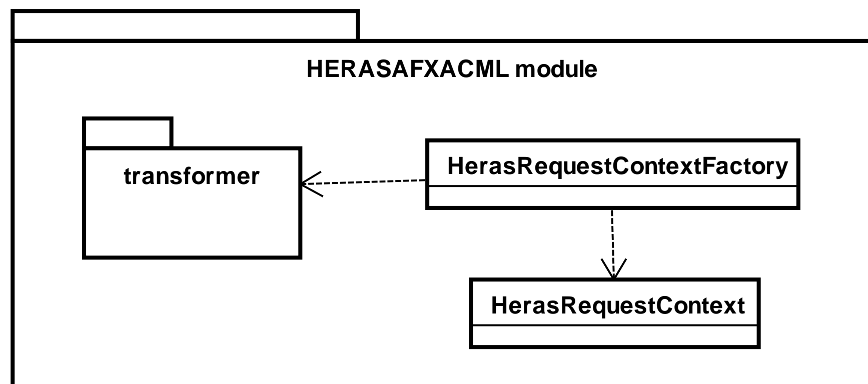



**Figure 26: Package diagram for herasafxacml module**

*Package description*

**(default) herasafxacml**

The herasafxacml package contains a precise implementation of a `RequestContextFactory` and a `RequestContext`. It includes a sub package transformer.

**transformer**

This package contains classes to transform the internally data structure used by the HERAS^AF PEP into the data structure used by the HERAS^AF XACML 2.0 implementation.

## 4.2 Package herasafxacml

*Overview*

The herasafxaml package contains a precise implementation of a `RequestContextFactory` and a `RequestContext`. It uses the HERAS^AF XACML 2.0 and the HERAS^AF PEP implementation. It is the integration layer between those implementations.

The transformer sub package contains classes to transform the internally data structure used by the HERAS^AF-PEP into the data structure used by the HERAS^AF XACML 2.0 implementation.

This is achieved by implementing the provided `Transformable` interfaces by the HERAS^AF XACML 2.0 implementation.

*class diagram*

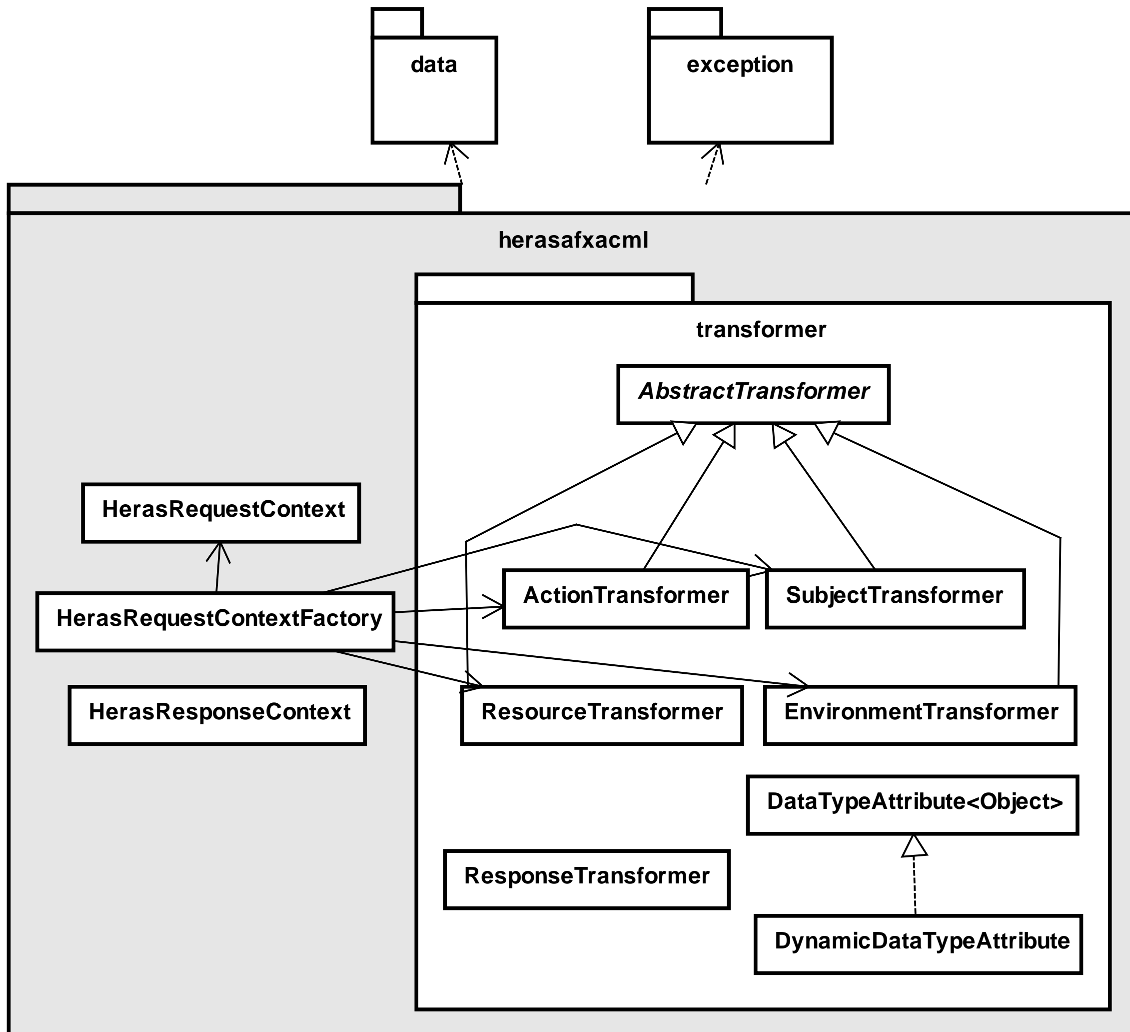



**Figure 27: Class diagram for herasafxacml package**

## 4.3 Interfaces / Abstract classes

*Overview*

This chapter explains the important interfaces and abstract classes inside the Herasafxacml module which are used to extend or change the implementation behavior of the PEP. The structure and some of the important methods are described.

### 4.3.1 AbstractTransformer

| ***AbstractTransformer*** |
|---|
| |
| # getAttributeTypes(attributes : List<AttributeVO>) : List<AttributeType><br># parseOtherAttributes(list : List<OtherAttributeVO>, to : Map<QName,String>) : void |

*Description*

The `AbstractTransformer` class defines common behavior for an implementation of a specific Transformer.

A Transformer transforms the PEP representation of an authorization request element into the HERAS^AF XACML 2.0 element representation. For example the `ActionTransformer` transforms an `ActionVO` into an HERAS^AF XACML 2.0 `ActionType`.

## 4.4 Design decisions

### 4.4.1 XACML request context creation

*Overview*

The request context encapsulates the XACML decision request. The contained elements of the request context are collected by the PEP.

Because the request context is unique for each decision request (stateful), it cannot be wired through the singleton approach of Spring. But the context uses static elements to transform the PEP data structure into the data structure of the used XACML implementation.

There are two alternatives how to implement the creation of his context.

*Alternatives*

**Alternative 1: Spring wiring with scope attribute to prototype**

With the scope attribute Spring provides a possibility to define a bean as non-singleton (stateful bean). The scope attribute would need to be set to "prototype". This is an inelegant way and bad design. Nothing from a static application context should be non-static.

The prototype-scoped bean definition of spring is just as somewhat of a replacement for the Java "new" operator. So this could be used instead.

| Advantage | Disadvantage |
|---|---|
| • Everything is wired through spring. | • Bad design, inelegance.<br>• The context object is not a data transfer object. |

**Alternative 2: Creation with a factory class**

A factory could be responsible for the creation of a request context. The factory would be wired through Spring and would include all the static

transformers.

A new request context would be created (through the Java "new" operator) whenever the factory method is called. The request context object itself wouldn't need any behavior. It would act as normal data transfer object.

| Advantage | Disadvantage |
|---|---|
| • Factory is a singleton and could manage the creation and the transformation.<br>• The transformer objects could be injected through spring into the factory.<br>• The request context would act as data transfer object and wouldn't contain any logic. | • none |

*Decision*

The chosen solution is to create the XACML request context with a factory class (alternative 2). The decision fell on this alternative because the wiring of the application static parts is made with Spring and the dynamic (stateful) parts are made through Java's "new" operator. This is good practice.

The creation of the XACML request context through the factory makes the object a data transfer object and more lightweight.

# 5 Utilities

*Overview*

This chapter explains the Utilities module.

This module contains all utility classes that may be used by certain implementations of interfaces in the Core module. For example resolvers or interceptor technologies that have to work with Java Annotations can use Annotation utility classes from this module.

## 5.1 HERAS^AF PEP Implementation

*Overview*

This chapter explains the packages and classes of the utilities module.

*Package diagram*

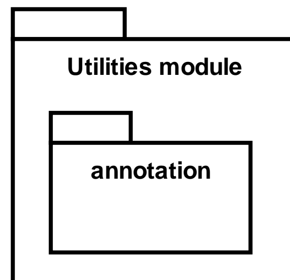



**Figure 28: package diagram for utilities module**

*Package description*

**Annotation**

This package contains utility classes that help working with Java Annotations.

## 5.2 Package annotation

*Overview*

This package contains utility classes that help working with Java Annotations.

For example it contains an `AnnotationParser` that provides methods to gather all data from HERAS^AF PEP annotations present on a certain class and its class hierarchy (all super classes and interfaces). In addition there is an `AnnotationCache` that can be used by `AnnotationParsers` to save time by caching the data for later reuse.

*class diagram*

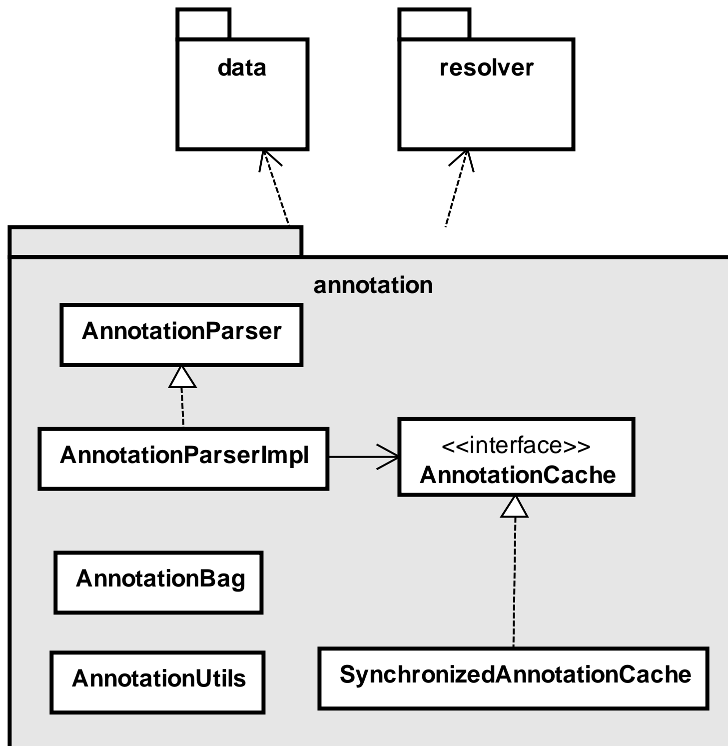



**Figure 29: Class diagram for annotation package**

## 5.3 Interfaces / Abstract classes

*Overview*

This chapter explains the important interfaces and abstract classes inside the Utilities module which are used to extend or change the implementation behavior of the PEP. The structure and some of the important methods are described.

### 5.3.1 AnnotationCache

<<interface>>
**AnnotationCache**

*+ hasTypeData(clazz : Class<?>) : boolean*
*+ hasMethodData(method : Method) : boolean*
*+ hasParameterData(method : Method) : boolean*
*+ getTypeAnnotations(clazz : Class<?>) : AnnotationBag*
*+ getMethodAnnotations(method : Method) : AnnotationBag*
*+ getParameterAnnotations(method : Method) : List<AnnotationBag>*
*+ setTypeAnnotations(clazz : Class<?>, annotations : AnnotationBag) : void*
*+ setMethodAnnotations(method : Method, annotations : AnnotationBag) : void*
*+ setParameterAnnotations(method : Method, annotations : List<AnnotationBag>) : void*

*Description* This interface defines a cache for value objects containing information from annotations. The cache can hold `AnnotationBags` and map each bag to a specified class or method.

### 5.3.2 AnnotationParser

<<interface>>
**AnnotationParser**

+ getSubjects(clazz : Class<?>, method : Method, arguments : Object[]) : List<SubjectVO>
+ getResources(clazz : Class<?>, method : Method, arguments : Object[]) : List<ResourceVO>
+ getAction(clazz : Class<?>, method : Method, arguments : Object[]) : ActionVO
+ getEnvironment(clazz : Class<?>, method : Method, arguments : Object[]) : EnvironmentVO

*Description* This interface defines a parser that parses annotations into the corresponding value objects.

## 5.4 Concepts

### 5.4.1 Annotation gathering

*Overview* The gathering logic for Java annotations can differ between implementations of `AnnotationParser`. This chapter will describe how the `AnnotationParserImpl` gathers annotations.

Definition: The term "class" circumscribes classes as well as interfaces.

*Gathering* Java annotations are gathered from all levels of a class's hierarchy. On each class annotations can be placed on the type, method and parameter level.

The order of the request information found in the annotations is defined by the

level it is placed on in the class (type, method and parameter) and then the annotations position in the class hierarchy.

For each annotation level (type, method and parameter), the gathering algorithm traverses through the class hierarchy in the following order:

1. The class
2. The interfaces of the class.
3. The super class of the class.

For each super class and interface this cycle is repeated until there is no more super class or interface to parse annotations from.

The class to start this algorithm is usually the class of the object whose method was invoked and intercepted.

*Combining*

The request information defined by Java annotations consists of four main types Subject, Resource, Action and Environment. Each of these elements contains Attributes with an ID and value.

When adding annotations on different levels of a class or class hierarchy it is possible that multiple @Subject, @Resource or @Attribute annotations describe the same request information elements.

To handle these cases the gathering algorithm determines the identity of the request information element contained in these annotations. If an element with the same identity has already been gathered, the content of the newly gathered element is merged into the existing one.

The identity of these request information elements is defined as follows:

- **Subjects:** The identity of a subject is defined by its group identifier and subject category.
- **Resources:** The identity of a resource is defined by its group identifier and resource content.
- **Attributes:** The identity of an attribute is defined by its identifier and issuer.

# 5.5 Design decisions

## 5.5.1 Java annotation gathering algorithm

*Overview*

One method to add request information for the XACML decision request is by using Java annotations. There are different alternatives how an algorithm can gather these Java annotations from the different classes and how annotations on different levels of the class hierarchy override or supplement each other.

*Alternatives*

**Alternative 1: Default annotation inheritance logic of Java**

The default annotation inheritance logic of Java is used.

Classes inherit annotations on class level from their super class if they are not annotated by the same annotation themselves. Annotations on method

or parameter level or on interfaces are not inherited.

| Advantage | Disadvantage |
|---|---|
| • Low costs, because the existing annotation inheritance logic of Java 1.5 can be used.<br>• Good performance, because all information is already present on the processed class and no class hierarchy has to be traversed. | • Little flexibility on how to place annotations into the code.<br>• The data of the same annotations on different levels in the class hierarchy can't supplement each other. |

**Alternative 2: Custom annotation inheritance and combination logic**

The class hierarchy is being traversed class by class to be able to find every annotation in classes, super classes and interfaces.

| Advantage | Disadvantage |
|---|---|
| • High flexibility on how to place annotations into the code.<br>• The data of the same annotations on different levels in the class hierarchy can supplement each other. | • High costs, because a custom logic has to be implemented.<br>• Bad performance, because the information has to be gathered from the whole class hierarchy up to the root class. |

*Decision*

The chosen solution is to implement a custom annotation inheritance algorithm (alternative 2). The main reason why we chose this solution is the high flexibility on how to place annotations into the code. It's worth the higher costs to be able to support the use of Java annotations on different levels of the class hierarchy and in different combinations.

To counteract the bad performance of this alternative we chose to use a cache to drastically reduce the amount of times the whole class hierarchy has to be traversed.

### 5.5.2 Parsing request information

*Overview*

Every time an action is intercepted resolvers must provide request information in the internal data structure of the PEP.

The transformation of this information into the PEP data structure can be delegated to parsers. These parsers will then parse the data from, for example, XML files or Java annotations. Because most of the data is static, caching strategies might be useful for these parsers.

*Alternatives*

**Alternative 1: Parse data on every call (no caching)**

The request information is parsed every time the parser is called.

A parser for XML data would unmarshal an XML file, or one for Java annotations, iterate through the class hierarchy on every call.

| Advantage | Disadvantage |
|---|---|
| • New request information and changes to this information is possible at runtime (without a restart, redeploy or recompilation of the application) | • Poor performance |

**Alternative 2: Parse data only once (with cache)**

The request information is parsed only once, if they are not currently present in the cache, and then cached for later reuse.

| Advantage | Disadvantage |
|---|---|
| • New request information can be added to the system at runtime.<br>• Good performance | • Changes to already parsed request information are not possible at runtime. |

**Alternative 3: Parse and cache data only on startup (with cache)**

The request information is parsed only once when the system is started and then cached for later reuse.

| Advantage | Disadvantage |
|---|---|
| • Good performance | • New request information and changes to this information are not possible at runtime. |

*Decision*

The chosen solution is to parses request information only once if it is not already in the cache (alternative 2). The main reason why we chose this solution is that this parser has a good performance and the caching is flexible enough to support the parsing and caching of newly added request information. The only disadvantage of alternative 2, that changes to already parsed request information cannot be detected by the parser, can be ignored because, in most cases, Java annotations are used to add request information. Changes to these Java annotations leads to a recompilation and redeployment of the application resulting in a new empty annotation cache.

# 6 General topics

*Overview* This chapter explains the HERAS$^{AF}$ PEP implementation wide topics.

## 6.1 Exception Handling

*Overview* This part explains the HERAS$^{AF}$ PEP Exception handling.

*General* HERAS$^{AF}$ PEP defines a `HerasException` which is thrown up to the client application. Any other exception is handled inside the HERAS$^{AF}$ PEP itself.

The communication back to the application is via `RuntimeExceptions` only. See 6.1.2 for an example how a client application could handle exceptions.

Figure 30 shows the Excepting Handling within an application in connection with the HERAS$^{AF}$ PEP.

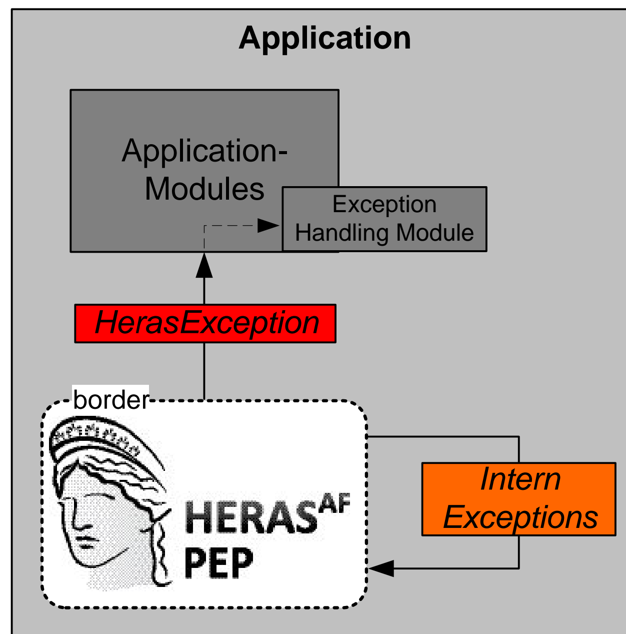



**Figure 30: HERAS$^{AF}$ PEP Exception Handling**

*Design decision* The goal was to define security issues with a single aspect and not have control constructs wherever security matters occur. So there is no possibility to return a value from the HERAS$^{AF}$ PEP back to the application.

A `RuntimeException` does not need to be catched and still can contain valuable information. Furthermore the whole PEP application stack can be traversed very fast back to the client if an Exception (for the application flow – e.g. Decision Exception) is thrown.

Normally it is bad practice to use Exceptions to control the application flow but for the functionality of the PEP it is the best and fastest solution we found.

### 6.1.1 Exception Types

*Overview* This part explains the different types of `HerasExceptions`.

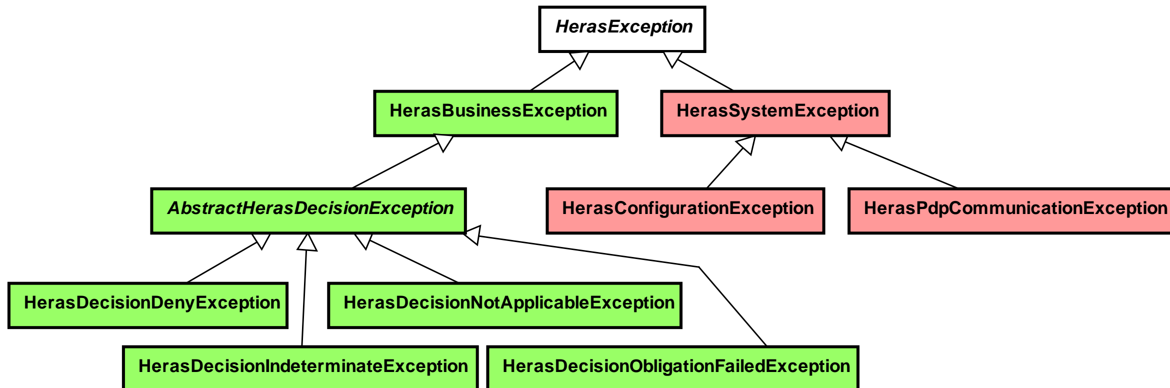



**Figure 31: Exception inheritance hierachy**

*Description* **HerasException**

This `RuntimeException` is used to mark all exceptions thrown by the HERAS^AF PEP into an application.

**HerasBusinessException**

This `HerasException` represents a normal business case of the HERAS^AF PEP. It is thrown when the execution of an intercepted method has to be aborted.

**HerasSystemException**

This `HerasException` is used to mark a failure in the system.

### 6.1.2 Client exception handling

*Overview* This part gives an example how a client could handle exceptions of the PEP.

*Scenario* A basic web application configured with the HERAS^AF PEP.

*Solution* A simple solution is to configure your web application's error page in the WEB-INF/web.xml file.

Example 12 shows a code snippet which configures that whenever a `HerasException` arises a default webpage is shown.

```
<error-page>
	<exception-type>HerasException</exception-type>
	<location>./notAuthorized.jsp</location>
</error-page>
```

**Example 12: Web.xml error-page snippet**

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

HERAS$^{AF}$
PEP

# Part IV: Implementing additional integration modules

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

HERAS$^{AF}$ PEP

# 1 Overview

*Introduction*

The HERAS$^{AF}$ PEP Implementation provides integration modules for specific technologies. This part of the Developer's Guide explains how new integration modules can be implemented and included into the existing HERAS$^{AF}$ PEP Implementation.

The basic Implementation of the HERAS$^{AF}$ PEP provides integration for the Spring AOP interceptor technology [SpringAOP] and declares various interfaces for further technologies. The first chapter explains the integration of an additional interceptor technology.

The HERAS$^{AF}$ PEP Implementation is using HERAS$^{AF}$ XACML as standard XACML 2.0 integration. An integration of another XACML 2.0 implementation can be done by implementing an integration module. The second chapter explains the integration of this additional XACML 2.0 implementation.

# 2 Integration of an interceptor technology

*Overview*

This part describes the integration of an additional interceptor technology.

## 2.1 Components Implementation

*Overview*

This part describes the components which are mandatory for the integration of an additional interceptor technology. The descriptions contain the details for the implementation.

See the Spring AOP module (chapter 2.6) for an example implementation of an intercepting technology.

### 2.1.1 Pointcut

*Description*

A pointcut describes the joint point when a method invocation should be intercepted because of its protection This contains the filtering of the classes which is marked, annotated to be protected.

The pointcut has an advice which is attached and called when the join point is reached. The advice initiates the whole HERAS$^{AF}$ PEP process.

The HERAS$^{AF}$ PEP does not provide an Interface for a pointcut.

*Dependency*

The pointcut uses an Advice.

### 2.1.2 Advice

*Description*

The advice initiates the HERAS$^{AF}$ PEP process by first creating the

DecisionContext and then calling the DecisionManagers decide method. The advice decides, based on the result of the DecisionManager, if he continues the poincut interception or not.

The HERAS^AF PEP does not provide an interface for an advice. The advice implementation must provide simple set-/getter methods in java bean naming convention for the specific DecisionManager and DecisionContextFactory.

*Dependency*

The advice uses a DecisionContextFactory to create a DecisionContext and uses the DecisionManager to start the decision process of the HERAS^AF PEP.

### 2.1.3 DecisionContextFactory

*Description*

A DecisionContextFactory is a factory to create a DecisionContext. It creates the DecisionContext based on the information of the intercepted method. Easiest way is to provide the whole joint point object if possible. Otherwise the DecisionContext needs the following information about the intercepted call:

- Class
- Runtime information of the method call

The HERAS^AF PEP does not provide a DecisionContextFactory interface.

*Dependency*

The DecisionContextFactory could optionally use the Resolvers to gather the information of the request elements.

### 2.1.4 DecisionContext

*Description*

A DecisionContext encapsulates the decision request of the HERAS^AF PEP. It simply contains all the information a XACML 2.0 request is based on.

The HERAS^AF PEP provides a DecisionContext interface.

### 2.1.5 Resolver

*Description*

A Resolver basically gathers request information from an element. For example a Resolver gathers information about a XACML 2.0 Resource attribute from a class method and Java Annotation.

It is highly recommend to

The HERAS^AF PEP provides an abstract class for a resolver. It is highly recommended to implement the same abstraction as used by the Spring AOP module implementation.

*Note*

These components are not mandatory.

The HERAS^AF PEP provides utility classes to gather XACML 2.0 attribute information from annotated class elements (Java 1.5 Annotation). See chapter 5.2 in part 5 for further information.

# 3 Integration of an XACML 2.0 Implementation

*Overview*

This part describes the integration of another additional XACML 2.0 implementation.

## 3.1 Components Implementation

*Overview*

This part describes the components which are mandatory for the integration. The descriptions contain the details for the implementation.

See the herasafxacml module (chapter 2.4) for an example integration of an XACML 2.0 implementation.

### 3.1.1 RequestContext

*Description*

A `RequestContext` encapsulates the decision request of the HERAS^AF PEP. It simply contains all the information used for a specific XACML 2.0 request.

The HERAS^AF PEP provides a `RequestContext` interface.

### 3.1.2 RequestContextFactory

*Description*

A RequestContextFactory is a factory to create a `RequestContext`. It creates the `RequestContext` based on the information of an XACML 2.0 request.

The HERAS^AF PEP provides a `RequestContextFactory` interface.

*Dependency*

The `RequestContextFactory` could optionally use the `Transformers` for the transformation.

### 3.1.3 ResponseContext

*Description*

A `ResponseContext` encapsulate the decision response of the HERAS^AF PEP. It simply contains all the information used for an XACML 2.0 response.

The HERAS^AF PEP provides a `ResponseContext` interface.

### 3.1.4 Transformer

*Description*

A transformer is used to transform the data structured used by the HERAS^AF PEP into the data structure of a XACML 2.0 Implementation. The elements of a XACML request/response need to be transformed. These are:

- Request: `Action`, `Environment`, `Resource`, `Subject` and `DataTypes`
- Response: `Obligation`, `Result`, `Status`, `StatusCode`, `Effect` and `Result`

The HERAS^AF PEP Implementation does not provide an interface for a `Transformer` because the vendor of an XACML implementation should provide a `Transformable` interface.

*Note*

This component is not mandatory.

# Part V: Testing

# 1 Overview

*Introduction*

For the development and verification of the HERAS^AF PEP implementation the need of testing is obvious.

In this part all the issues around and about the testing from the HERAS^AF PEP implementation are discussed and outlined.

The HERAS^AF PEP implementation is tested with two testing approaches. The functionality of each class is tested with unit tests and the functionality with all modules together is tested with integrations tests.

The functionality of the classes in the HERAS^AF PEP is tested with the TestNG Framework [TestNG].

See the chapter 2.1 in part 6 for a description how the tests are integrated in Maven [Maven].

# 2 Module / - Unit tests

*Overview*

This chapter describes the module and unit tests of the HERAS^AF PEP.

There are two types of tests performed.

- Tests which prove the success way
- Tests which simulate the fault way

*Test setup*

To test the classes separately some references to other classes were exchanged with mock objects. These mock objects don't contain any logic they just return a value which is suspected.

The TestNG Framework provides a way to run the tests with different data inputs which they call testing with data providers. Nearly all the tests are run with this approach to tests as many different data combination as possible.

## 2.1 Core module

*Introduction*

In this chapter the entire tests from the module core are listed.

### 2.1.1 TestDefaultDecisionHandler

*Test cases*

| **handlePermitDecision** |
| --- |
| Tests if the `DefaultDecisionHandler` can handle a PERMIT decision and does not throw an Exception. |

**handleDenyDecision**

Tests if the `DefaultDecisionHandler` can handle a DENY decision and looks for a `HerasBusinessException` to be thrown.

**handleIndeterminateDecision**

Tests if the `DefaultDecisionHandler` can handle an INDETERMINATE decision and looks for a `HerasBusinessException` to be thrown.

**handleNotApplicableDecision**

Tests if the `DefaultDecisionHandler` can handle a NOTAPPLICABLE decision and looks for a `HerasBusinessException` to be thrown.

### 2.1.2 TestDefaultExceptionHandler

*Test cases*

**handleHerasPdpCommunicationException**

Tests if the `DefaultExceptionHandler` can handle a `HerasPdpCommunicationException`.

### 2.1.3 TestDefaultObligationProcessorImpl

*Test cases*

**uninitializedProcessor**

Tests the `DefaultObligationProcessor` with no `ObligationHandlers` and expects an `ObligationFailedException` to be thrown.

**handlerNotFound**

Tests the `DefaultObligationProcessor` with a list of `ObligationHandlers` but no matching obligation-ids and expects an `ObligationFailedException` to be thrown.

**handlerThrowsException**

Tests the `DefaultObligationProcessor` with a list of `ObligationHandlers` and matching obligation-ids but the ObligationHandler throws an `ObligationFailedException`.Testcases expects an `ObligationFailedException` to be thrown.

## 2.2 Herasafxacml module

*Introduction* In this chapter the entire tests from the module herasafxacml are listed.

### 2.2.1 TestActionTransformer

*Test cases*

**transform**

Tests an `ActionTransformer` if a transformation from an `ActionVO` into an `ActionType` object is done correctly. Checks if the objects contain the same values after the transformation.

### 2.2.2 TestEnvironmentTransformer

*Test cases*

**transform**

Tests an `EnvironmentTransformer` if a transformation from an `EnvironmentVO` into an `EnvironmentType` object is done correctly. Checks if the objects contain the same values after the transformation.

### 2.2.3 TestRequestContext

*Test cases*

**encode**

Tests the `HerasRequestContextFactory` creation and the correct encoding of the `HerasRequestContext`. Uses XMLUnit to diff the encoded XML.

### 2.2.4 TestResourceTransformer

*Test cases*

**transform**

Tests a `ResourceTransformer` if a transformation from a `ResourceVO` into a `ResourceType` object is done correctly. Checks if the objects contain the same values after the transformation.

### 2.2.5 TestSubjectTransformer

*Test cases*

**transform**

Tests a `SubjectTransformer` if a transformation from a `SubjectVO` into a `SubjectType` object is done correctly. Checks if the objects contain the same values after the transformation.

## 2.3 Utilities module

*Introduction*

In this chapter the entire tests from the module utilities are listed.

### 2.3.1 TestAnnotationParser

*Test cases*

**parseAllAnnotations**

Tests the parsing of all 4 annotation types as well as the handling of `Attributes` without a value.

**mergeAttributes**

Tests the merging of `Attributes`.

### 2.3.2 TestAnnotationUtils

*Test cases*

**getFindTypeAnnotation**

Tests the search for a specific type annotation.

# 3 Integration tests

*Introduction*

The functionality of the HERAS^AF PEP as one piece is tested with integration tests. The module herasaf-pep-integrationtests contains an integration test scenario.

Some modules contain simple integration tests within its module to test the interaction with classes inside the module itself. These are described in the previous chapter 2 in part 5.

## 3.1 herasaf-pep-integrationtests

*Overview*

In this chapter the tests from the module herasaf-pep-integrationtests are listed. These tests perform a full integration test for the HERAS^AF PEP but without the Web Service client.

### 3.1.1 TestAopProtectedClass

*Test cases*

**testGenerics**

Tests the process of intercepting a method, collecting request information and the creation of the `RequestContext` (encoding). Checks the encoded XML with XMLUnit. Uses a MockResponseContext to set the response to permit.

**testDerivedClassDenyDecision**

Tests the process with a DENY `MockResponseContext`. Expects a `HerasBusinessException` to be thrown.

**testDerivedClassNotApplicableDecision**

Tests the process with a NOTAPPLICABLE `MockResponseContext`. Expects a `HerasBusinessException` to be thrown.

**testDerivedClassObligationFailed**

Tests the process and creates an `ObligationFailedException` on purpose. Expects a `HerasBusinessException` to be thrown.

# 4 Testing hints

*Introduction* This chapter gives testing hints to verify and simplify the testing.

## 4.1 Code coverage - EclEmma

*Description* EclEmma is a free Java code coverage tool for Eclipse, available under the Eclipse Public License. Internally it is based on the great EMMA Java code coverage tool, trying to adopt EMMA's philosophy for the Eclipse workbench:

- Fast develop/test cycle: Launches from within the workbench like JUnit test runs can directly be analyzed for code coverage.
- Rich coverage analysis: Coverage results are immediately summarized and highlighted in the Java source code editors.
- Non-invasive: EclEmma does not require modifying your projects or performing any other setup. [EclEmma]

## 4.2 Do not run tests in Eclipse

*Description* The HERAS^AF PEP is developed with Maven and its ability to run tests. This testing approach and only this should be used. Do not run tests in Eclipse and make the assumption everything works properly if Eclipse says so. The development of the HERAS^AF PEP showed a few times that this is an arbitrary assumption.

The reason for is simple. The Eclipse installation contains a lot of libraries, which are may used by your application. So whenever the tests are running inside the Eclipse container all of the libraries are available but if executed with Maven there not.

Use Maven's simple command `mvn test` to make sure the tests are running properly.

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

HERAS^AF
PEP

# Part VI: Deployment

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

HERAS^AF
PEP

# 1 Overview

*Introduction*

To build the whole HERAS^AF PEP Implementation the need of automated software build is obvious. This part describes how the library dependencies, the automated running of tests, and the deployment of HERAS^AF are managed.

Maven 2 [Maven] is that detailed in this guide because the information for configuration of Maven 2 and its plugins is spread over several web pages and books. It is much easier if this information is collected at one point.

In the first paragraph is an overview over Apache Maven 2. Then the responsibilities of the herasaf-pep root module are explained.

# 2 Apache Maven 2

*Introduction*

To use Maven 2 as automated software build, it is required to install and configure Maven 2 properly.

*settings.xml*

In the settings.xml from Maven configuration-settings for Maven are specified, for example where the Local-Maven-Repository is located or define profiles. [MVNSettings]

*POM*

A Project Object Model is the fundamental unit of work in Maven. It is an XML file that contains information about the project and configuration details used by Maven to build the project.

*External libraries*

To use external libraries in a project, you have to declare a dependency in the pom.xml which points to the library in the repository.

*Project dependencies*

Project dependencies are also declared in the pom.xml. Like external libraries, you have to declare a dependency to each project.

*Tests*

The directory `src/test/java` is the Maven default source directory for test classes. Every project could contain tests.

The configurations to run automated tests declare the Maven-surefire-plugin. See Figure 34 in chapter 2.1.

## 2.1 Plugins

*Introduction*

Maven is a plugin execution framework; all work is done by plugins. The listened plugins in this chapter were used for the development.

*Javadoc generation*

Maven provides a plugin to generate javadoc and pack it as a jar.

**maven-javadoc-plugin [MVNJavaDoc]**

```
<plugin>
      <artifactId>maven-javadoc-plugin</artifactId>
      <executions>
            <execution>
                  <phase>package</phase>
                  <goals>
                        <goal>jar</goal>
                  </goals>
            </execution>
      </executions>
      <configuration>
            <aggregate>true</aggregate>
            <quiet>true</quiet>
      </configuration>
</plugin>
```

**Figure 32: Example maven-javadoc-plugin plugin configuration**

Hint: Make sure to define the execution phase <phase>package</phase>. Other phases can cause NullPointerExceptions while generating the generation.

*Source packing*

Maven provides a plugin to create a jar archive of the source files.

**maven-source-plugin [MVNSource]**

```
<plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-source-plugin</artifactId>
      <executions>
            <execution>
                  <id>attach-sources</id>
                  <phase>verify</phase>
                  <goals>
                        <goal>jar</goal>
                  </goals>
            </execution>
      </executions>
</plugin>
```

**Figure 33: marven-source-plugin plugin configuration**

*Testing plugin*

Maven provides a plugin to run tests during the test phase of the build lifecycle to execute the unit tests of an application.

**maven-surefire-plugin [MVNSurefire]**

```
<plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-surefire-plugin</artifactId>
      <configuration>
            <skip>false</skip>
      </configuration>
</plugin>
```

**Figure 34: Example maven-surefire-plugin plugin configuration**

*Deploying to containers plugin*

Codehaus.org provides a plugin which wraps the Cargo Java API. The plugin provides functionality to start containers for integration and functional tests.

**cargo-maven2-plugin [MVNCargo]**

The following configuration deploys a war file to a web container, starts it, runs the tests and stops the container again.

```
<plugin>
      <groupId>org.codehaus.cargo</groupId>
      <artifactId>cargo-maven2-plugin</artifactId>
      <executions>
            <execution>
                  <id>start-container</id>
                  <phase>pre-integration-test</phase>
                  <goals>
                        <goal>deploy</goal>
                        <goal>start</goal>
                  </goals>
            </execution>
            <execution>
                  <id>stop-container</id>
                  <phase>post-integration-test</phase>
                  <goals>
                        <goal>stop</goal>
                  </goals>
            </execution>
      </executions>
            <configuration>
            <wait>false</wait>
            <container>
                  <containerId>
                  ${appserver.containerId}
                  </containerId>
                  <home>${appserver.home}</home>
            </container>
            <configuration>
                  <type>existing</type>
                  <home>${appserver.home}</home>
            </configuration>
                  <deployer>
                        <deployables>
                        <deployable>
                              <groupId>
                                    aGroupId
                              </groupId>
                              <artifactId>
                                    anArtifactId
                              </artifactId>
                              <type>war</type>
                              <properties>
                                    <context>
                                          webContext
                                    </context>
                              </properties>
                        </deployable>
                        </deployables>
                  </deployer>
```

```
            </configuration>
</plugin>
```

**Figure 35: cargo-maven2-plugin plugin configuration**

Hint: The configuration contains variables (e.g. **${appserver.home}**). These are specified in the user-specific configuration file for Maven (settings.xml)

Figure 36 shows an example configuration for the above cargo-maven2-plugin configuration. The bold elements are the needed for the configuration in Figure 35.

```
<?xml version="1.0" encoding="UTF-8"?>
<settings xmlns="http://maven.apache.org/POM/4.0.0"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/settings-1.0.0.xsd">
<localRepository>
      C:\projects\HERAS_MavenRepository
</localRepository>
 <profiles>
  <profile>
   <id>tomcat</id>
   <properties>
    <appserver.containerId>
      tomcat5x
    </appserver.containerId>
    <appserver.deploy.dir> </appserver.deploy.dir>
    <appserver.home>
      C:\apache-tomcat-5.5.26
    </appserver.home>
   </properties>
  </profile>
 </profiles>
 <activeProfiles>
      <activeProfile>tomcat</activeProfile>
 </activeProfiles>
</settings>
```

**Figure 36: Maven user-specific configuration for tomcat**

# 3 herasaf-pep

*Introduction*

The herasaf-pep is the main project of the HERAS[AF] PEP implementation. It is used to link to the other project and perform a complete Maven build.

*Standard build*

Go to the herasaf-pep directory in the command console to perform a Maven build with all the integration tests and execute:

```
mvn clean install
```

# 4 Hints

*Introduction*

This chapter gives some hints while working with Maven. Some useful discoveries were made during the development. These are mentioned in this chapter.

## 4.1 Eclipse and Maven

*Q for Eclipse Plugin*

Q is an Apache Maven plugin for Eclipse that will allows to use Maven2 from the Eclipse IDE. [Q4Eclipse]

It brings some useful features like the Dependency Analysis which helps to clean up the dependencies of the modules. During the development of the HERAS^AF PEP library version conflicts occurred. All of them could be fixed with this plugin.

The plugin is still in its beginning phase and still contains a lot of defects which makes sometimes the work which Eclipse quite painful.

*Maven build and errors*

Eclipse suddenly shows many errors in the project. Eclipse shows a lot of errors after each build in Maven. Just execute the `clean` Action in the Project context menu from Eclipse after you built with Maven. All the errors will disappear again.

# Appendix A: General

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

HERAS^AF
PEP

# 1 Glossary

| | |
|---|---|
| *Action* | An operation on a resource |
| *Attribute* | Characteristic of a subject, resource, action or environment that may be referenced in a predicate or target. |
| *Authorization decision* | Returned by the PDP to the PEP as a result of evaluating a decision request. |
| *Bag* | An unordered collection of values, in which there may be duplicate values |
| *Decision* | The result of evaluating a rule, policy or policy set |
| *Decision context* | Context containing request information required to create a decision request. |
| *Decision request* | The request from a PEP to a PDP to render an authorization decision. |
| *Effect* | The intended consequence of a satisfied rule (either "Permit" or "Deny") |
| *Environment* | The set of attributes that are relevant to an authorization decision and are independent of a particular subject, resource or action. |
| HERAS^AF | Holistic Enterprise Ready Application Security ^Architecture Framework |
| *Mock object* | Mock objects are simulated objects that mimic the behavior of real objects in controlled ways |
| *OASIS* | Organization for the Advancement of Structured Information Standards |
| *Obligation* | An operation specified in a policy or policy set that should be performed by the PEP in conjunction with the enforcement of an authorization decision |
| *PAP* | Policy Administration Point |
| *PDP* | Policy Decision Point |
| *PEP* | Policy Enforcement Point |
| *PIP* | Policy Information Point |

| | |
|---|---|
| *Pluggability* | Applications which were designed to provide default functionality that should be suitable for most any application. However in order to provide complete flexibility pluggablity applications allow its foundational components to be over-written with custom implementations. |
| *Policy* | A set of rules, an identifier for the rule-combining algorithm and (optionally) a set of obligations. May be a component of a policy set |
| *Prototype* | A prototype is often software in a development stage, focusing on a subset of the total requirements for a product. |
| *Request context* | Context containing a decision request. |
| *Request information* | Information about the Subjects, Resources, Action and Environment that is used to generate the content of an XACML request. |
| *Resolver* | An adaption point of the PEP that acquires request information from a data source and parses it into a data structure known by the PEP. |
| *Resource* | Data, service or system component. |
| *Response context* | Context containing an authorization decision. |
| *SAML* | Security Assertion Markup Language |
| *SOAP* | Simple Object Access Protocol |
| *SSL* | Secure Socket Layer |
| *Subject* | An actor whose attributes may be referenced by a predicate. |
| *TLS* | Transport Layer Security |
| *UML* | Unified Modeling Language is a standardized visual specification language for object modeling. |
| *XACML* | eXtensible Access Control Markup Language |
| *XML* | eXtensible Markup Language |

# 2 Bibliography

## 2.1 Specifications and Standards

[XACML] OASIS
eXtensible Access Control Markup Language (XACML), Version 2, 5 Jul 2007
http://www.oasis-open.org/committees/download.php/24548/ (06.06.2008)

[SAML]
[SAMLBind] OASIS
Security Assertion Markup Language v2.0
http://docs.oasis-open.org/security/saml/v2.0/saml-2.0-os.zip (06.06.2008)

[SAMLXACML] OASIS
SAML 2.0 Profile of XACML, Version 2, Working Draft 5, 19 July 2007
http://www.oasis-open.org/committees/download.php/24679 (06.06.2008)

[SOAP] W3C
Simple Object Access Protocol v1.1
http://www.w3.org/TR/2000/NOTE-SOAP-20000508/ (06.06.2008)

[OpenSource] The Open Source Definition
http://www.opensource.org/docs/definition.php (06.06.2008)

## 2.2 HERAS^AF documents

[RW08PEP] Daniel Regli, Yannick Winiger:
HERAS^AF PEP
Bachelor Thesis
June 2008
Bachelor Thesis at the University of Applied Sciences Rapperswil (HSR)

[RW08PEPDev] Daniel Regli, Yannick Winiger:
HERAS^AF PEP
Developer's Guide
June 2008
Bachelor Thesis at the University of Applied Sciences Rapperswil (HSR)

[RW08PEPPDP Dev] Daniel Regli, Yannick Winiger:
HERAS^AF PEP-PDP Communication

| | |
|---|---|
| | Developer's Guide<br>June 2008<br>Bachelor Thesis at the University of Applied Sciences Rapperswil (HSR) |
| *[RW08PEPUsr]* | Daniel Regli, Yannick Winiger:<br>HERAS$^{AF}$ PEP<br>User's Guide<br>June 2008<br>Bachelor Thesis at the University of Applied Sciences Rapperswil (HSR) |
| *[RW08PEPPDP Usr]* | Daniel Regli, Yannick Winiger:<br>HERAS$^{AF}$ PEP-PDP Communication<br>User's Guide<br>June 2008<br>Bachelor Thesis at the University of Applied Sciences Rapperswil (HSR) |
| *[NZ08PAP]* | Patrick Neyer, Christoph Zellweger:<br>HERAS$^{AF}$ Policy Deployment Modul<br>Bachelor Thesis<br>June 2008<br>Bachelor Thesis at the University of Applied Sciences Rapperswil (HSR) |
| *[NZ08PAPDev]* | Patrick Neyer, Christoph Zellweger:<br>HERAS$^{AF}$ PAP<br>Developer's Guide<br>June 2008<br>Bachelor Thesis at the University of Applied Sciences Rapperswil (HSR) |
| *[NZ08PAPUsr]* | Patrick Neyer, Christoph Zellweger:<br>HERAS$^{AF}$ PAP<br>User's Guide<br>June 2008<br>Bachelor Thesis at the University of Applied Sciences Rapperswil (HSR) |
| *[CS07PAP]* | Massimo Cerqui, Sandro Strebel:<br>HERAS$^{AF}$: Policy Administration Point<br>November 2007<br>Diploma Thesis at the University of Applied Sciences Rapperswil (HSR) |
| *[CS07PEP]* | Massimo Cerqui, Sandro Strebel: |

HERAS^AF: Interzeptoren für Spring AOP und AspectJ

July 2007

Diploma Thesis at the University of Applied Sciences Rapperswil (HSR)

*[DOH07DA]* Sacha Dolski, Stefan Oberholzer, Florian Huonder:

HERAS^AF: XACML 2.0 Implementation
Hauptdokument

November 2007

Diploma Thesis at the University of Applied Sciences Rapperswil (HSR)

*[DOH07DADev]* Sacha Dolski, Stefan Oberholzer, Florian Huonder:

HERAS^AF: XACML 2.0 Implementation
Developer's Guide

November 2007

Diploma Thesis at the University of Applied Sciences Rapperswil (HSR)

*[DOH07SA]* Sacha Dolski, Stefan Oberholzer, Florian Huonder:

HERAS^AF: XACML PDP Web Service Endpoint

July 2007

Diploma Thesis at the University of Applied Sciences Rapperswil (HSR)

*[Egg06]* René Eggenschwiler:

HERAS^AF – Holistic Enterprise-Ready Application Security Architecture Framework
Manageable policy-based access control for J2EE.

Mai 2006

Diploma Thesis at the University of Applied Sciences Rapperswil (HSR)

*[Graf06]* Yan Graf:

Distributed Access Control Policies – Enterprise Ready

December 2006

Diploma Thesis at the University of Applied Sciences Rapperswil (HSR)

## 2.3 Other Resources

*[EclEmma]* http://www.eclemma.org/

*[Maven]* http://maven.apache.org/

*[MVNCargo]* http://cargo.codehaus.org/Maven2+plugin

*[MVNJavaDoc]* http://maven.apache.org/plugins/maven-javadoc-plugin/

*[MVNSettings]* http://maven.apache.org/ref/2.0.8/maven-settings/settings.html

*[MVNSource]* http://maven.apache.org/plugins/maven-source-plugin/

*[MVNSurefire]* http://maven.apache.org/plugins/maven-surefire-plugin/

*[Q4Eclipse]* http://code.google.com/p/q4e/

*[Spring]* http://www.springframework.org/

*[SpringAOP]* http://static.springframework.org/spring/docs/2.5.x/reference/aop.html

*[SpringIDE]* http://www.springide.org/

*[SpringSecurity]* http://static.springframework.org/spring-security/site/index.html

*[TestNG]* http://www.testng.org/doc/

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

HERAS$^{AF}$
PEP

# Appendix B: Issues, Extensions and Refactorings

# 1 Overview

*Introduction*

This appendix contains open issues, extension points and recommended refactorings of the HERAS$^{AF}$ PEP.

# 2 Open Issues

*Overview*

This chapter contains open issues of the HERAS$^{AF}$ PEP.

*Resource Content value*

The values of ResourceContent are often XML data.

The HERAS$^{AF}$ PEP provides an integration of HERAS$^{AF}$ XACML (herasafxacml module). The integration is used to unmarshall the XACML request into XML.

The HERAS$^{AF}$ XACML implementation has a problem with unmarshalling special characters into XML. Special characters are unmarshalled in HTML entities (e.g. "&" is unmarshalled in "&").

**Solution:**

No workaround is available. It will be fixed in a coming release of HERAS$^{AF}$ XACML.

# 3 Extension Points

*Overview*

This chapter contains the possible extension points to this implementation. Listed are capabilities which could make the work with the HERAS$^{AF}$ PEP more comfortable.

*Decision Cache*

The feature Decision Caching would make the HERAS$^{AF}$ PEP faster. A received decision could be cached temporarily. The HERAS$^{AF}$ PEP could look make a lookup in the decision cache before a decision request is sent to a PDP. With this enhancement a whole roundtrip could be save.

A major requirement for the cache would be the performance. The lookup operation and the caching of decision must be very fast. Furthermore an analysis of how long a decision should be cached must be performed.

*ReturnContext and InputContext Only*

The HERAS$^{AF}$ PEP in connection with a SAML PDP can use additional SAML Profile attributes to define special behavior for the decision evaluation of the PDP. These Boolean attributes are `ReturnContext` and `InputContextOnly`.

See page 23 of [SAMLXACML] for further information.

*Integration of intercepting technologies*

Implementation of integration modules for further intercepting technologies.

For example:

- AspectJ interceptors

- EJB interceptors
- Servlet filter

*Integration of request information sources*

Implementation of integration modules for further request information sources.

For example:

- XML files
- Databases

# 4 Refactorings

*Overview*

This chapter contains refactorings which should be done to improve performance or design of the HERAS^AF PEP implementation.

*Optimized implementation of Annotation Cache*

The `AnnotationCache` (`SynchronizedAnnotationCache`) implementation of the HERAS^AF PEP is using synchronized methods to be thread-safe. This makes the `AnnotationCache` slow and should be replaced by a high-performance implementation of an `AnnotationCache`.

*Renaming of Annotation Cache*

The concept behind the `AnnotationCache` could be applied to every request information sources, not only Java annotations.

Therefore we recommend renaming this interface to `RequestInformationCache`. Implementations would have to be renamed accordingly.

*Move and Rename SpringSecurity AopSubject Resolver*

The `SpringSecurityAopSubjectResolver` class should be moved to the Core module or a separate Spring Security module.

The resolver does not contain any logic which depends on Spring AOP and could be used as a general Spring Security Resolver. The class is inside the Springaop module because only 1 interceptor technology integration exists by now which is Spring AOP.