



User'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

Table of Contents

PART I: INTRODUCTION	4
1 INTENDED AUDIENCE.....	5
2 OVERVIEW	6
2.1 DEFINITION.....	6
3 RUNTIME ENVIRONMENT	7
PART II: BUILDING A HERAS^{AF} PEP	9
1 OVERVIEW	10
2 CONFIGURATION.....	10
2.1 CORE (CORE.XML).....	11
2.2 XACML INTEGRATION (XACML.XML).....	12
2.2.1 ContextAndPolicyConfiguration	12
2.3 ACCESS INTERCEPTOR (INTERCEPTOR.XML).....	12
2.3.1 AttributeDataTypes	16
2.4 PDP (PDP.XML)	16
2.5 SECURITY (SECURITY.XML).....	17
3 PROTECTING THE INTERFACE	19
4 RESULT.....	21
PART III: REFERENCE	23
1 OVERVIEW	24
2 ACCESS INTERCEPTION	25
2.1 SPRING AOP.....	25
2.1.1 Attribute Data Types	26
2.1.2 Request Information Acquisition (Resolvers).....	27
2.2 JAVA ANNOTATION PARSING	28
2.3 ANNOTATION CACHING	28
2.4 DECISION REQUEST GENERATION	29
2.4.1 HERAS ^{AF} XACML.....	29
3 PDP COMMUNICATION	30
3.1 LOCATING A PDP	30
3.1.1 DefaultPdpLocator	30
3.2 COMMUNICATE WITH A PDP	30
4 DECISION HANDLING	31
4.1 DEFAULTDECISIONHANDLER.....	31
5 EXCEPTION HANDLING	33
5.1 DEFAULTEXCEPTIONHANDLER.....	33
APPENDIX A: GENERAL	34
1 GLOSSARY.....	35

2	BIBLIOGRAPHY	37
2.1	SPECIFICATIONS AND STANDARDS	37
2.2	HERAS ^{AF} DOCUMENTS	37
2.3	OTHER RESOURCES	39

Part I: Introduction

1 Intended audience

General

This guide provides instructions for the usage and installation of the HERAS^{AF} PEP framework.

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

- Java 1.5
- HERAS^{AF}
- Spring Framework (Core [Spring], Spring AOP [SpringAOP])
- XACML 2.0 [XACML]

Installation instructions are only given for HERAS^{AF} components. References to external installation guides are provided whenever the component needs an additional installation of another vendor.

2 Overview

2.1 Definition

<i>HERAS^{AF}</i>	<p>HERAS^{AF} is an Open Source project in its formation phase.</p> <p>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.</p>
<i>HERAS^{AF} PEP</i>	<p>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.</p>
<i>HERAS^{AF} PDP</i>	<p>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.</p> <p>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.</p> <p>The HERAS^{AF} PDP will be used as a central unit, which interacts with several PEPs. [DOH07DADev]</p>
<i>HERAS^{AF} PAP</i>	<p>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]</p>
<i>XACML</i>	<p>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]</p>

3 Runtime environment

General The HERAS^{AF} PEP runs within a standard Java 1.5 Runtime Environment.

Modules It consists of the following modules.

herasaf-pep-core.jar

The Core module is the central part of the HERAS^{AF} PEP functionality. It contains the structural models as well as the PEP logic framework. It provides various interfaces/classes for the integration of intercepting technologies, XACML implementations and other extensions and adaptations.

herasaf-pep-springaop.jar

The Spring AOP module provides a Spring AOP interceptor technology implementation that integrates with the Core module. It allows you to use Spring AOP method interceptors to control access with HERAS^{AF} PEP.

It contains various resolvers and a Spring AOP decision context. The resolvers are used to get additional request information. This data is kept in the Spring AOP decision context which is used by the Core module to create a decision request.

herasaf-pep-herasafxacml.jar

The Herasafxacml module provides the integration of HERAS^{AF} XACML into the Core module for the generation of decision requests.

herasaf-pep-utilities.jar

The Utilities module provides utilities useful to several integration modules like the Spring AOP module. It contains a Java Annotation parser used to find and parse annotations on classes and interfaces, and an annotation cache to optimize the acquisition of request information.

herasaf-pep-ws.jar

The WS module provides a Web Service client that integrates with the Core module and allows evaluating decision requests by a remote PDP Web Service. It includes a basic SAML Handler that guarantees the standard-compliant handling of XACML over SAML over SOAP [SOAP].

Details about this module can be found in [RW08PEPPDPUsr].

Module Dependencies Figure 1 illustrates the HERAS^{AF} PEP modules and the dependencies between them. Arrows indicate dependencies, i.e. HERAS^{AF} PEP Spring AOP depends on HERAS^{AF} PEP Core and HERAS^{AF} PEP Utilities.

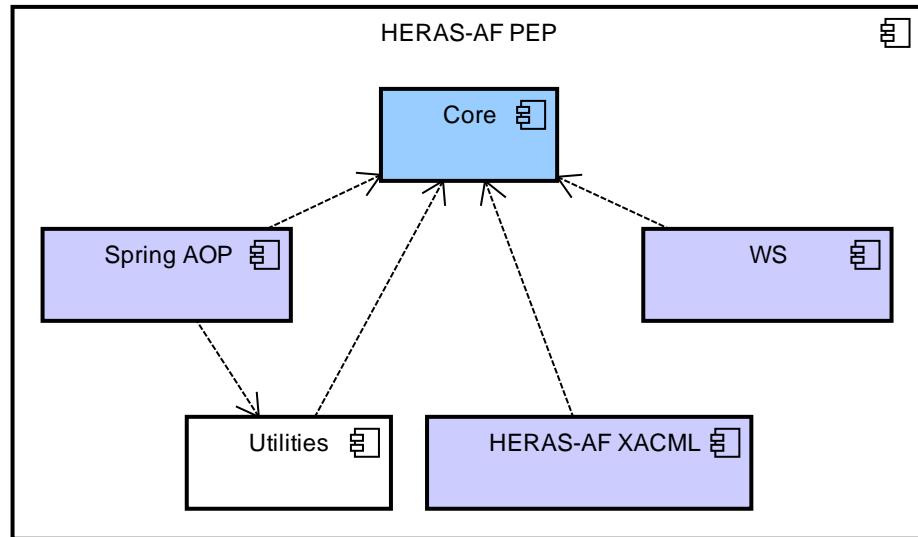


Figure 1: HERAS^{AF} PEP module dependencies

Part II: Building a HERAS^{AF} PEP

1 Overview

Introduction The tutorial in this part shows how a HERAS^{AF} PEP is build and used to control access in your application.

In this tutorial, we will add security to a simple application called *MedInfo* that manages medical information about patients in a hospital.

Scenario *MedInfo* contains a single service class to access the medical history of a patient. In the current state there is no access control and anyone can access the data of any patient. The service class implements the following interface shown in Example 1.

```
public interface PatientMgmt {
    public MedicalHistory getMedicalHistory(
        String patient);
}
```

Example 1: Unprotected *PatientMgmt* interface

The goal of this tutorial is it to control access to the `getMedicalHistory` method with the HERAS^{AF} PEP.

2 Configuration

Overview In this part we will use the Spring application context [Spring] to wire and configure the different components of the HERAS^{AF} PEP. Aside from the core logic, the specific behavior of the HERAS^{AF} PEP is mostly defined by multiple modules that can be plugged into the Core module.

The HERAS^{AF} PEP framework defines 5 main extension points that allows you to choose, how your specific PEP should behave. These extension points are:

- How the access to resources is intercepted. (Interceptor)
- How request information for the decision making process is acquired. (Resolver)
- With which XACML implementation decision requests are generated. (XACML Integration)
- How the PEP communicates to a PDP. (PDP Locator and PDP)
- How an authorization decision is handled/interpreted. (Decision Handler)

Common elements All Spring configuration files have the following common elements.

Beans namespaces

```
<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 2: Common elements of Spring configuration files

File structure

For this tutorial we separated the configuration of the HERAS^{AF} PEP into 5 different configuration files. Each of the following chapters describes one of these files.

```
<beans ...>
  <import resource="Core.xml" />
  <import resource="Xacml.xml" />
  <import resource="Interceptor.xml" />
  <import resource="Pdp.xml" />
  <import resource="Security.xml" />
</beans>
```

Example 3: Import of configuration files into the Spring application context

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.

2.1 Core (core.xml)

Overview

This part shows the core configuration of a HERAS^{AF} PEP.

Configuration

```
<bean id="decisionManager"
  class="...DecisionManagerImpl">
  <property name="pdpLocator" ref="pdpLocator"/>
  <property name="requestFactory"
  ref="requestFactory"/>
  <property name="decisionHandler"
  ref="decisionHandler"/>
</bean>
```

Example 4: Core configuration

Element description and values

[decisionManager](#) bean description

Description:	Configures the Decision Manager. The recommended <code>DecisionManagerImpl</code> defines the core logic and flow of the HERAS ^{AF} PEP.
Properties	<p><code>pdpLocator</code></p> <p>Sets the PDP Locator implementation. (required)</p> <p><code>requestFactory</code></p> <p>Sets the Request Context Factory implementation (required)</p> <p><code>decisionHandler</code></p> <p>Sets the Decision Handler implementation (required)</p>

2.2 XACML Integration (Xacml.xml)

Overview This part shows the XACML Integration configuration to define which integration of an XACML implementation is used to generate decision requests. It is recommended to use the integration of HERAS^{AF} XACML [DOH07DADev].

Configuration

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

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

Example 5: XACML Integration configuration

Element description and values

General Element description

<import resource... Imports an additional configuration file that is mandatory when using HERAS^{AF} XACML. Chapter 2.2.1 describes this file.

requestFactory bean description

Description: Configures the Request Context Factory. The used `HerasRequestContextFactory` allows the generation of decision request with HERAS^{AF} XACML.

2.2.1 ContextAndPolicyConfiguration

Overview The `ContextAndPolicyConfiguration.xml` describes the various configuration options of the context and policy files. For a detailed configuration guide see the developer's guide of HERAS^{AF} XACML.

2.3 Access Interceptor (Interceptor.xml)

Overview This part shows the Access Interceptor configuration to define which interceptor technology is used to intercept access to resources. For this tutorial we use the integration module for the Spring AOP interceptor technology [SpringAOP].

Configuration

```
<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.medinfo.*.*(..))" />
</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="...SpringSecurityAopSubjectResolver"
      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 6: Access Interceptor configuration using Spring AOP

Element
description and
values

General Element description

<code><aop:config..</code>	Defines a Spring AOP aspect and an around advice in connection with its pointcut. Whenever the pointcut gets active the <code>secureMethod</code> method is called.
-------------------------------	---

`expressionAnnotationPointcut` bean description

Description:	Configures which implementation of <code>AspectJExpressionPointcut</code> should be used.
Property	<code>expression</code> Sets an AspectJ expression for the pointcut.

`accessControlAdvice` bean description

Description:	Configures which implementation of an <code>AccessControlAdvice</code> should be used.
Property	<code>springAopDecisionContextFactory</code> Sets a <code>SpringAopDecisionContextFactory</code> implementation. (required) <code>decisionManager</code> Sets a <code>DecisionManger</code> implementation. (required)

`springAopDecisionContextFactory` bean description

Description:	Configures the <code>SpringAopDecisionContextFactory</code> .
Property	<code>subjectResolver</code> Sets the <code>AbstractSpringAopSubjectResolver</code> implementation. (required) <code>resourceResolver</code>

Sets the `AbstractSpringAopResourceResolver` implementation. (required)

`actionResolver`

Sets the `AbstractSpringAopActionResolver` implementation. (required)

`environmentResolver`

Sets the `AbstractSpringAopEnvironmentResolver` implementation. (required)

`abstractResolver` bean description

Description:	Configures an abstract bean definition with common behavior.
Property	<code>attributeDataTypes</code> Sets the <code>DataTypeAttributes</code> map. (required)

`subjectResolver` bean description

Description:	Configures which implementation of an <code>AbstractSpringAopSubjectResolver</code> should be used.
--------------	---

`resourceResolver`, `actionResolver`, `environmentResolver` bean description

Description:	Configures which implementation of an <code>AbstractSpringAopXXXResolver</code> should be used.
Property	<code>annotationParser</code> Sets an <code>AnnotationParser</code> implementation. (required)

`annotationParser` bean description

Description:	Configures which implementation of an <code>AnnotationParser</code> should be used.
Property	<code>annotationCache</code> Sets an <code>AnnotationCache</code> implementation. (required)
	<code>attributeDataTypes</code> Sets an <code>AttributeDataTypes</code> implementation. (required)

Hint This configuration file needs an additional namespace import.

```
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 7: Additional namespace import

2.3.1 AttributeDataTypes

Overview This part shows the configuration of the AttributeDataTypes used by the resolvers of the interceptor technology integration modules to lookup data types of attributes based on their ID.

Configuration

```
<bean id="attributeConfiguration"
class="...AttributeDataTypes">
<property name="attributeDataTypes">
<map>
<entry key="urn:medinfo:resource:service"
value="http://www.w3.org/2001/XMLSchema#String" />
<entry key="urn:medinfo:resource:name"
value="http://www.w3.org/2001/XMLSchema#String" />
<entry key="urn:medinfo:patient:name"
value="http://www.w3.org/2001/XMLSchema#String" />
<entry key="urn:medinfo:action"
value="http://www.w3.org/2001/XMLSchema#String" />
</map>
</property>
</bean>
```

Example 8: Configuration of the AttributeDataTypes

Element description and values

attributeConfiguration bean description

Description:	Configures the DataTypeAttributes and defines attribute ID to data type mappings.
Property	<p><code>attributeDataTypes</code></p> <p>The property is required and sets the id to data type mapping for attributes.</p>

2.4 PDP (Pdp.xml)

Overview This part shows the PDP configuration to define how PDPs are located and which PDP to use.

Configuration

```
<bean id="pdpLocator" class="...DefaultPdpLocator">
<property name="pdp">
<ref bean="..." />
</property>
```

</bean>

Example 9: XACML Integration configuration
 Element
 description and
 values
pdpLocator bean description

Description:	Configures which implementation of a PdpLocator should be used.
Property	pdp Sets a Pdp implementation. (required)

2.5 Security (Security.xml)

Overview

This part shows the Security configuration to define how request information about subjects is resolved. For this tutorial we use Spring Security [SpringSecurity].

Configuration

```

<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[
                dr.heras@medInfo.org=anyPw
            ]]>
    </value>
  </property>
</bean>

```

Example 10: Security configuration with a mock InMemoryDaoImpl
 Element
 description and
 values
authenticationManager bean description

Description:	Configures which implementation of a ProviderManager should be used.
--------------	--

Property `providers`
Sets a list of `AuthenticationProvider` to use.

`daoAuthenticationProvider` bean description

Description: Configures which implementation of an `AuthenticationProvider` should be used.

Property `userDetailsService`
Sets a `UserDetailsService` implementation.

`inMemoryDaoImpl` bean description

Description: Configures which implementation of a `InMemoryDaoImpl` is used. This is just a mock class. In a live system you would not need this as you get all the information from your `AuthenticationProvider`.

Property `userMap`
Sets a `UserMap`. This is a mock map for usernames and passwords.

3 Protecting the interface

- Overview* Now that the HERAS^{AF} PEP is wired up we can begin to protect our `PatientMgmt` interface. This is done by enabling protection for the interface and adding request information to the interface.
- Enable protection* In chapter 2.3 we injected the integration module for the Spring AOP interceptor technology into our PEP. This module uses the `@Protected` annotation defined in the Core module to annotate classes that should be protected.
- To enable the interception of all method invocations in classes implementing our interface we annotate the interface as in Example 11.

```
@Protected
public interface PatientMgmt {
    public MedicalHistory getMedicalHistory(
        String patient);
}
```

Example 11: Protected PatientMgmt interface

Add request information In XACML terminology, the invocation of the `getMedicalHistory` method represents an access attempt with the following request information:

- Access by a **Subject**.
- Access to a **Resource** defined by 3 values:
 - The class name.
 - The method name.
 - The value of the patient argument.
- A **READ Action**.
- Access in an **Environment** defined by the current date/time.

In the configuration of the Spring AOP interceptor in chapter 2.3 we used three `AnnotationSpringAopXXXResolvers` and a `SpringSecurityAopSubjectResolver` to acquire request information about subjects, resources, the action and the environment. These types of resolver acquire request information from specific Java annotations placed on the class, method or parameters of the intercepted method invocation.

Java Annotations Example 11 shows how to use the `@Resource` and `@Action` annotations of the Core module to add request information about the resource and action to the interface.

```
@Protected
@Resource(@Attribute(id = "urn:medinfo:resource:service"))
public interface PatientMgmt {
    @Resource(@Attribute(id = "urn:medinfo:resource:name"))
    @Action(@Attribute(id = "urn:medinfo:action", value = {
        @AttributeValue("READ")
    }))
    public MedicalHistory getMedicalHistory(
        @Resource(@Attribute(id = "urn:medinfo:patient:name"))
```

```
String patient);
```

```
}
```

Example 12: Protected PatientMgmt interface with request information

Notice that the value attribute is omitted in most of the `@Attribute` annotations. This tells the used `AnnotationSpringAopXXXResolver` to use the value of the annotated element (class, method, and parameter) as the value of the `@Attribute` annotation. In our case the name of the interface and method, as well as the `toString` value of the `patient` parameter.

Spring Security

The request information about the Subject is acquired by the `SpringSecurityAopSubjectResolver` which interacts with the `SecurityContextHolder` of Spring Security. The `SecurityContextHolder` stores details of the principal currently interacting with the application (e.g. Authentication). See [SpringSecurity] for further information.

Default XACML request information

The Environment information about the time/date of the method invocation is automatically added by HERAS^{AF} XACML.

See the [DOH07DADev] for more information about this default behavior.

4 Result

Overview With the configured HERAS^{AF} PEP and enabled protection for our interface we have successfully secured our application with HERAS^{AF}. Assuming a policy exists on the PDP.

The only part that has not been discussed by this tutorial is the communication to the PDP. We could inject a Web Service client to evaluate the generated decision request by a remote PDP. These components of HERAS^{AF} are described in the HERAS^{AF} PEP-PDP – User's Guide [RW08PEPPDPUsr].

decision request With this configuration of HERAS^{AF} PEP and the request information annotated to our interface we would get the XACML decision request shown in Example 13.

```
<xacml-context:Request>
  <xacml-context:Subject
SubjectCategory="urn:oasis:names:tc:xacml:1.0:subject-category:access-
subject">
  <xacml-context:Attribute
AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
DataType="http://www.w3.org/2001/XMLSchema#string">
  <xacml-context:AttributeValue>
    dr.heras@medinfo.org
  </xacml-context:AttributeValue>
  </xacml-context:Attribute>
</xacml-context:Subject>
<xacml-context:Resource>
  <xacml-context:Attribute
AttributeId="urn:medinfo:resource:service"
DataType="http://www.w3.org/2001/XMLSchema#string">
  <xacml-context:AttributeValue>
    PatientMgmt
  </xacml-context:AttributeValue>
  </xacml-context:Attribute>
  <xacml-context:Attribute
AttributeId="urn:medinfo:resource:name"
DataType="http://www.w3.org/2001/XMLSchema#string">
  <xacml-context:AttributeValue>
    getMedicalHistory
  </xacml-context:AttributeValue>
  </xacml-context:Attribute>
  <xacml-context:Attribute AttributeId="urn:medinfo:patient:name"
DataType="http://www.w3.org/2001/XMLSchema#string">
  <xacml-context:AttributeValue>Jane Doe</xacml-
context:AttributeValue>
  </xacml-context:Attribute>
</xacml-context:Resource>
<xacml-context:Action>
  <xacml-context:Attribute AttributeId="urn:medinfo:action"
DataType="http://www.w3.org/2001/XMLSchema#string">
  <xacml-context:AttributeValue>
```



```

        READ
        </xacml-context:AttributeValue>
    </xacml-context:Attribute>
</xacml-context:Action>
<xacml-context:Environment>
    <xacml-context:Attribute
AttributeId="urn:oasis:names:tc:xacml:1.0:environment:current-time"
DataType="http://www.w3.org/2001/XMLSchema#time">
        <xacml-context:AttributeValue>
            10:33:13.959+02:00
        </xacml-context:AttributeValue>
    </xacml-context:Attribute>
    <xacml-context:Attribute
AttributeId="urn:oasis:names:tc:xacml:1.0:environment:current-date"
DataType="http://www.w3.org/2001/XMLSchema#date">
        <xacml-context:AttributeValue>2008-06-02+02:00</xacml-
context:AttributeValue>
    </xacml-context:Attribute>
    <xacml-context:Attribute
AttributeId="urn:oasis:names:tc:xacml:1.0:environment:current-
dateTime" DataType="http://www.w3.org/2001/XMLSchema#dateTime">
        <xacml-context:AttributeValue>
            2008-06-02T10:33:13.959+02:00
        </xacml-context:AttributeValue>
    </xacml-context:Attribute>
</xacml-context:Environment>
</xacml-context:Request>

```

Example 13: XACML decision request generated on method invocation

Part III: Reference

1 Overview

Introduction

This part of the guide details the various components that comprise HERAS^{AF} PEP. This includes chapters about interceptor technologies, XACML implementations, decision handlers and much more.

Plugability

A main feature of the HERAS^{AF} PEP is its plugability. Multiple extension points allow the integration of specific implementations. The following figure shows the 5 main extension points of HERAS^{AF} PEP.

There are a few additional extension points like PDP Location and Exception Handling which are also discussed in the following chapters.

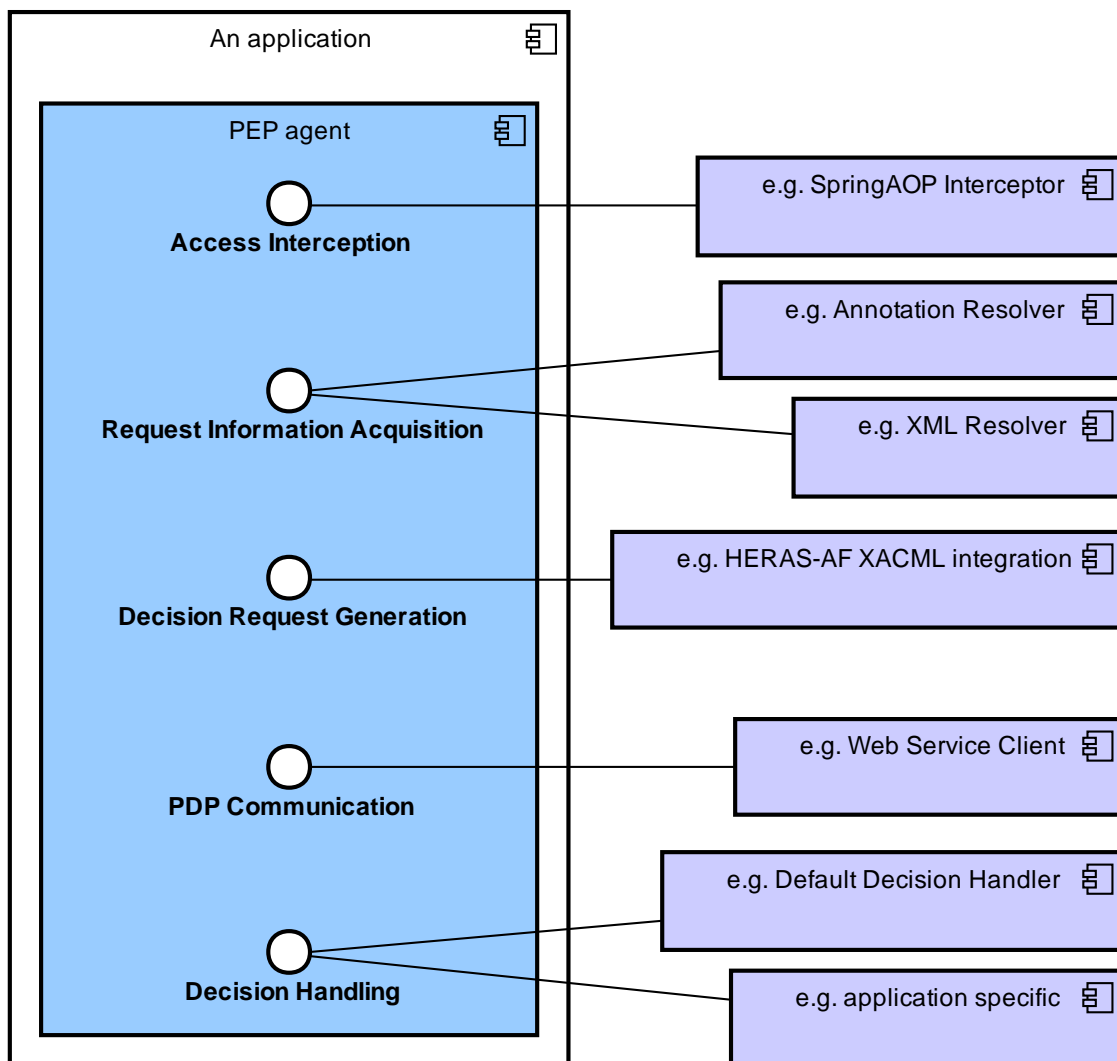


Figure 2: Main extension points of HERAS^{AF} PEP

2 Access Interception

Overview Access Interception is used to intercept access to a resource. There are many different interceptor technologies, for example for Java classes there is Spring AOP [SpringAOP] or AspectJ to intercept the invocation of a method.

HERAS^{AF} PEP currently contains an integration module for the interceptor technology Spring AOP.

2.1 Spring AOP

Overview The Spring AOP module provides a Spring AOP interceptor technology implementation that integrates with the Core module. It allows you to use Spring AOP method interceptors to control access with HERAS^{AF} PEP.

It contains various resolvers and a Spring AOP decision context. The resolvers are used to get additional request information. This data is kept in the Spring AOP decision context which is used by the Core module to create a decision request.

See the Spring Reference [Spring] for more information about the Spring AOP framework.

Wiring Example 14 shows how to wire the Spring AOP module into a HERAS^{AF} PEP.

```
<beans>
<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="org.herasaf.pep.integration.springaop.ExpressionAnno
tationPointcut">
  <property name="expression" value="execution(*
org.medicalInfo.service.*.*(..))" />
</bean>

<bean id="accessControlAdvice"
class="org.herasaf.pep.integration.springaop.AccessControlA
dvice">
  <property name="springAopDecisionContextFactory"
ref="springAopDecisionContextFactory" />
  <property name="decisionManager"
ref="decisionManager" />
</bean>

<bean id="springAopDecisionContextFactory"
```

```

class="org.herasaf.pep.integration.springaop.SpringAopDecisionContextFactory">
  <property name="subjectResolver"
ref="subjectResolver" />
  <property name="resourceResolver"
ref="resourceResolver" />
  <property name="actionResolver" ref="actionResolver"
/>
  <property name="environmentResolver"
ref="environmentResolver" />
</bean>
...
</beans>

```

Example 14: How to wire the Spring AOP module

The `AccessControlAdvice` uses a `DecisionManager` and the `SpringAopDecisionContextFactory` uses four `Resolvers` which are defined in chapter 2.1.2.

The configuration of the attribute data types is imported. This is used by resolvers and annotation parsers described in the following chapters.

2.1.1 Attribute Data Types

Overview The attribute data types define the mapping between attribute IDs and the data type of an attribute. It is used to simplify the writing of request information. Descriptions of attributes only need to contain an attribute ID but the data type can be omitted because of its 1:1 dependency to the ID.

Wiring Example 15 shows how to wire the attribute data types into the `AttributeDataTypes` class.

```

<beans>
<bean id="attributeConfiguration"
class="org.herasaf.pep.resolver.AttributeDataTypes">
  <property name="attributeDataTypes">
    <map>
      <entry key="urn:medinfo:resource:service"
value="http://www.w3.org/2001/XMLSchema#String" />
      <entry key="urn:medinfo:subject:name"
value="http://www.w3.org/2001/XMLSchema#String" />
      ...
    </map>
  </property>
</bean>
</beans>

```

Example 15: How to wire the attribute data types



2.1.2 Request Information Acquisition (Resolvers)

2.1.2.1 Java Annotations

<i>Overview</i>	The Spring AOP module for HERAS ^{AF} PEP implements three resolvers to acquire Resource, Action and Environment request information from Java annotations.
<i>Wiring</i>	<p>Example 16 shows how to wire the Java annotation resolvers of the Spring AOP module.</p> <pre> <bean id="resourceResolver" class="org.herasaf.pep.integration.springaop.resolver.AnnotationSpringAopResourceResolver"> <property name="attributeDataTypes" ref="attributeConfiguration" /> <property name="annotationParser" ref="annotationParser" /> </bean> <bean id="actionResolver" class="org.herasaf.pep.integration.springaop.resolver.AnnotationSpringAopActionResolver"> <property name="attributeDataTypes" ref="attributeConfiguration" /> <property name="annotationParser" ref="annotationParser" /> </bean> <bean id="environmentResolve" class="org.herasaf.pep.integration.springaop.resolver.AnnotationSpringAopEnvironmentResolver"> <property name="attributeDataTypes" ref="attributeConfiguration" /> <property name="annotationParser" ref="annotationParser" /> </bean> </pre>

Example 16: How to wire the Java annotation resolvers

All resolvers use `AttributeDataTypes` and an `AnnotationParser` which are defined in chapters 2.1.1 and 2.2.

2.1.2.2 Spring Security

<i>Overview</i>	The Spring AOP module for HERAS ^{AF} PEP implements one resolver to acquire Subject request information from Spring Security.
<i>Wiring</i>	<p>Example 17 shows how to wire the Java annotation resolvers of the Spring AOP module.</p> <pre> <bean id="subjectResolver" class="org.herasaf.pep.integration.springaop.resolver.Sprin </pre>



```
gSecurityAopSubjectResolver">
    <property name="attributeDataTypes"
        ref="attributeConfiguration" />
</bean>
```

Example 17: How to wire the Spring Security resolver

The `SpringSecurityAopSubjectResolver` uses an `AttributeDataTypes` which is defined in chapter 2.1.1.

2.2 Java Annotation Parsing

Overview The Utilities module provides classes for the parsing of Java annotations [JavaAnno]. These `AnnotationParser` are used by annotation resolvers of interceptor technology modules like Spring AOP.

Wiring Example 18 shows example how to wire the Java annotation parser of the Utilities module.

```
<bean id="annotationParser"
class="org.herasaf.pep.annotation.AnnotationParserImpl">
    <property name="annotationCache"
        ref="annotationCache" />
    <property name="attributeDataTypes"
        ref="attributeConfiguration" />
</bean>
```

Example 18: How to wire the Java annotation parser.

The `AnnotationParserImpl` uses an `AnnotationCache` and an `AttributeDataTypes` which are defined in chapters 2.3 and 2.1.1.

2.3 Annotation Caching

Overview The Utilities module provides classes for the caching of request information (annotations). These `AnnotationCaches` are used by the `AnnotationParserImpl`.

Synchronized annotation cache The `SynchronizedAnnotationCache` is a thread-safe annotation cache. Each call that writes or reads data of the cache gets synchronized.

Example 19 shows how to wire the synchronized annotation cache of the Utilities module.

```
<bean id="annotationCache"
class="org.herasaf.pep.annotation.SynchronizedAnnotationCache" />
```

Example 19: How to wire the synchronized annotation cache.

2.4 Decision Request Generation

Overview Decision Request Generation is used to generate XACML decision requests based on request information. There are different XACML implementations that can be used for this task, for example HERAS^{AF} XACML [DOH07DADev]. HERAS^{AF} PEP currently contains an integration module for HERAS^{AF} XACML.

2.4.1 HERAS^{AF} XACML

Overview The Herasafxacml module provides the integration of HERAS^{AF} XACML into the HERAS^{AF} PEP for the generation of decision requests. It contains a factory to create request contexts that encapsulate a decision request. See the [DOH07DADev] chapter 6, for more information about the HERAS^{AF} XACML implementation.

Wiring Example 20 shows how to wire the Herasafxacml module into the HERAS^{AF} PEP.

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

<bean id="requestFactory"
class="org.herasaf.pep.integration.herasafxacml.HerasRequestContextFactory" />
</beans>
```

Example 20: How to wire the HERAS^{AF} XACML integration.

The imported configuration file for the context and policy configuration is used by HERAS^{AF} XACML.

3 PDP Communication

Overview The PDP Communication is used to evaluate a decision request by a PDP. It consists of the following two aspects:

- Locating a PDP
- Communicating with a PDP

3.1 Locating a PDP

3.1.1 DefaultPdpLocator

Overview The `DefaultPdpLocator` can be used to always locate the same PDP implementation.

Wiring Example 21 shows how to wire the default pdp locator into the HERAS^{AF} PEP.

```
<beans>
<bean id="pdpLocator"
class="org.herasaf.pep.integration.DefaultPdpLocator">
  <property name="pdp">
    <ref bean="pdp" />
  </property>
</bean>
...
</beans>
```

Example 21: How to wire the default PDP locator.

The `DefaultPdpLocator` uses a `Pdp` which is defined in chapter 2.4.

3.2 Communicate with a PDP

Overview There are many different ways to communicate with a PDP, depending on its location. Either the PDP is directly in the local JVM or it is remote and can only be reached by some sort of client, for example a Web Service client.

HERAS^{AF} PEP currently contains a module for the communication with a remote PDP over a Web Service. See the [RW08PEPPDPUsr] for more information about the usage of this module.

4 Decision Handling

Overview The Decision Handling is used to handle authorization decision returned by a PDP. Depending on the application the HERAS^{AF} PEP is used with, the handling of decisions can be different.

HERAS^{AF} PEP contains a default decision handler that handles decision in a standard-compliant way.

4.1 DefaultDecisionHandler

Overview The `DefaultDecisionHandler` handles decisions in standard-compliant way. It is ensured that obligations attached to the decision are processed properly. The handler throws `HerasBusinessExceptions`, if a business case occurs where access to a resource is not permitted,

When using this decision handler the HERAS^{AF} PEP can throw the following `HerasBusinessExceptions`:

- `HerasDecisionDenyException`
- `HerasDecisionNotApplicableException`
- `HerasDecisionIndeterminateException`
- `HerasDecisionObligationFailedException`

Wiring Example 22 shows how to wire the default decision handler into the HERAS^{AF} PEP.

```
<beans>
<bean id="decisionHandler"
class="org.herasaf.pep.core.DefaultDecisionHandler">
  <property name="obligationProcessor"
ref="obligationProcessor" />
</bean>
...
</beans>
```

Example 22: How to wire the default decision handler.

The `DefaultDecisionHandler` uses an `ObligationProcessor`.

Obligation Processing Example 23 shows how to wire the obligation processor for the default decision handler.

```
<beans>
<bean id="obligationProcessor"
class="org.herasaf.pep.obligation.ObligationProcessorImpl">
  <property name="obligationHandlers">
    <util:map>
      <entry key="urn:medinfo:obligation:1"
value-ref="obligationHandler1" />
    </util:map>
  </property>
</bean>
```

```
        </property>  
    </bean>  
</beans>
```

Example 23: How to wire the obligation processor.

The `ObligationProcessorImpl` uses a map of `ObligationHandlers`.

Notice: The property `obligationHandlers` is optional and doesn't have to be set. An obligation processor with no obligation handlers will throw an exception as soon as there is an obligation in the authorization decision to be processed.

5 Exception Handling

Overview

The Exception Handling is used to handle all system exception that might occur within the HERAS^{AF} PEP. Depending on the application the HERAS^{AF} PEP is used with, the handling of exceptions can be different.

HERAS^{AF} PEP contains a default exception handler that handles exceptions in a standard-compliant way.

5.1 DefaultExceptionHandler

Overview

The `DefaultExceptionHandler` handles exceptions by simply throw them up into the application. When using this exception handler the HERAS^{AF} PEP can throw the following `HerasSystemExceptions`:

- `HerasPdpCommunicationException`
- `HerasConfigurationException`

Wiring

Example 24 shows how to wire the default decision handler into the HERAS^{AF} PEP.

```
<bean id="exceptionHandler"  
      class="org.herasaf.pep.core.DefaultExceptionHandler">  
</bean>
```

Example 24: How to wire the default exception handler.

Appendix A: General

1 Glossary

<i>Action</i>	An operation on a resource
<i>Attribute</i>	Characteristic of a subject, resource, action or environment that may be referenced in a predicate or target.
<i>Authorization decision</i>	Returned by the PDP to the PEP as a result of evaluating a decision request.
<i>Bag</i>	An unordered collection of values, in which there may be duplicate values
<i>Decision</i>	The result of evaluating a rule, policy or policy set
<i>Decision context</i>	Context containing request information required to create a decision request.
<i>Decision request</i>	The request from a PEP to a PDP to render an authorization decision.
<i>Effect</i>	The intended consequence of a satisfied rule (either "Permit" or "Deny")
<i>Environment</i>	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 <small>Architecture Framework</small>
<i>Mock object</i>	Mock objects are simulated objects that mimic the behavior of real objects in controlled ways
<i>OASIS</i>	Organization for the Advancement of Structured Information Standards
<i>Obligation</i>	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
<i>PAP</i>	Policy Administration Point
<i>PDP</i>	Policy Decision Point
<i>PEP</i>	Policy Enforcement Point
<i>PIP</i>	Policy Information Point

<i>Pluggability</i>	Applications which were designed to provide default functionality that should be suitable for most any application. However in order to provide complete flexibility pluggability applications allow its foundational components to be over-written with custom implementations.
<i>Policy</i>	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
<i>Prototype</i>	A prototype is often software in a development stage, focusing on a subset of the total requirements for a product.
<i>Request context</i>	Context containing a decision request.
<i>Request information</i>	Information about the Subjects, Resources, Action and Environment that is used to generate the content of an XACML request.
<i>Resolver</i>	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.
<i>Resource</i>	Data, service or system component.
<i>Response context</i>	Context containing an authorization decision.
<i>SAML</i>	Security Assertion Markup Language
<i>SOAP</i>	Simple Object Access Protocol
<i>SSL</i>	Secure Socket Layer
<i>Subject</i>	An actor whose attributes may be referenced by a predicate.
<i>TLS</i>	Transport Layer Security
<i>UML</i>	Unified Modeling Language is a standardized visual specification language for object modeling.
<i>XACML</i>	eXtensible Access Control Markup Language
<i>XML</i>	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]* OASIS
[SAMLBind] 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)

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
June 2008

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

[RW08PEPUsr]

Daniel Regli, Yannick Winiger:

HERAS^{AF} PEP
User's Guide

June 2008

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

*[RW08PEPPDP
Usr]*

Daniel Regli, Yannick Winiger:

HERAS^{AF} PEP-PDP Communication
User's Guide

June 2008

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

[NZ08PAP]

Patrick Neyer, Christoph Zellweger:

HERAS^{AF} Policy Deployment Modul
Bachelor Thesis

June 2008

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

[NZ08PAPDev]

Patrick Neyer, Christoph Zellweger:

HERAS^{AF} PAP
Developer's Guide

June 2008

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

[NZ08PAPUsr]

Patrick Neyer, Christoph Zellweger:

HERAS^{AF} PAP
User's Guide

June 2008

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

[CerStr07PAP]

Massimo Cerqui, Sandro Strebhel:

HERAS^{AF}: Policy Administration Point

November 2007

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

[CerStr07PEP]

Massimo Cerqui, Sandro Strebhel:

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

[JavaAnno] <http://java.sun.com/docs/books/tutorial/java/javaOO/annotations.html>

[Spring] <http://www.springframework.org/>

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

[SpringSecurity]

<http://static.springframework.org/spring-security/site/>