



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

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
2.2 HERAS ^{AF}	7
2.3 REQUIREMENTS OF THE PEP-PDP COMMUNICATION	9
2.4 MODULE DESIGN	10
PART II: CONFIGURATION	12
1 OVERVIEW	13
2 HERAS^{AF} PEP WS CLIENT CONFIGURATION AND INTEGRATION	13
2.1 CONFIGURATION FILES	13
2.2 HERAS ^{AF} PEP WS CLIENT.....	14
3 HERAS^{AF} PDP CONTEXT-WS ENDPOINT CONFIGURATION	17
3.1 WEB APPLICATION DEPLOYMENT DESCRIPTOR.....	17
3.2 HERAS ^{AF} PDP CONTEXT-WS ENDPOINT	17
PART III: ARCHITECTURE & DESIGN	20
1 OVERVIEW	21
2 HERAS^{AF} PDP CONTEXT-WS	22
2.1 SCENARIO.....	23
2.2 IMPORTANT CLASSES AND INTERFACES.....	24
2.2.1 ServerSamlHandler.....	24
2.2.2 AbstractServerSamlHandler	24
2.2.3 SamlPdpEndpoint.....	25
2.3 DESIGN DECISIONS.....	25
2.3.1 WSDL exposure	25
3 HERAS^{AF} PEP WEB SERVICE	27
3.1 SZENARIO	28
3.2 IMPORTANT CLASSES AND INTERFACES.....	29
3.2.1 ClientSamlHandler.....	29
3.2.2 AbstractClientSamlHandler	30
3.2.3 SamlRemotePdpClient.....	30
3.2.4 SamlPdp.....	30
4 HERAS^{AF} SAML	32
4.1 IMPORTANT CLASSES AND INTERFACES.....	33
4.1.1 DecisionQuery-, Response- and AssertionFactory.....	34
4.1.2 SamlException	35
4.1.3 SamlExceptionHandler	35
4.2 DESIGN DECISIONS.....	35
4.2.1 Marshalling/Unmarshalling the XML	35
4.2.2 SAML standard-compliance	37

PART IV: TESTING 39

1 OVERVIEW	40
2 MODULE AND UNIT TESTS.....	40
2.1 HERAS ^{AF} PDP CONTEXT-WS.....	40
2.1.1 TestAbstractServerSamlHandler	40
2.2 HERAS ^{AF} PEP WS.....	41
2.2.1 TestAbstractClientSamlHandler	41
2.2.2 TestSamlPdp.....	41
2.3 HERAS ^{AF} SAML	42
2.3.1 TestSamlExceptionHandler	42
2.3.2 TestFactories	42
2.3.3 TestJaxb2NamespaceMarshaller.....	42
3 INTEGRATION TESTS	43
3.1 HERASAF-PDP-WS-CONTEXT-INTEGRATIONTESTS	43
4 TESTING HINTS	44
4.1 CODE COVERAGE - ECLÉMMMA	44
4.2 DO NOT RUN TESTS IN ECLIPSE	44

PART V: DEPLOYMENT..... 45

1 OVERVIEW	46
2 APACHE MAVEN 2	46
2.1 PLUGINS.....	46
3 HERASAF-PEP	49
4 HERASAF-PDP.....	50
5 HERASAF-SAML.....	50
6 HINTS	50
6.1 ECLIPSE AND MAVEN.....	50

APPENDIX A: GENERAL 52

1 GLOSSARY.....	53
2 BIBLIOGRAPHY	55
2.1 SPECIFICATIONS AND STANDARDS.....	55
2.2 HERAS ^{AF} DOCUMENTS	55
2.3 OTHER RESOURCES	57

APPENDIX B: ISSUES, EXTENSIONS AND REFACTORINGS 59

1 OVERVIEW	60
2 OPEN ISSUES.....	60
3 EXTENSION POINTS	60
4 REFACTORINGS.....	61

Part I: Introduction

1 Intended audience

General

This Developer's Guide provides information on the architecture and design of the HERAS^{AF} PEP-PDP Communication 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-PDP Communication Implementation.

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

The developers should bring solid 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

<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} PEP Web Service</i>	<p>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.</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. 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} PDP Context-WS</i>	<p>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.</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>HERAS^{AF} PIP</i>	<p>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.</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</p>

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.

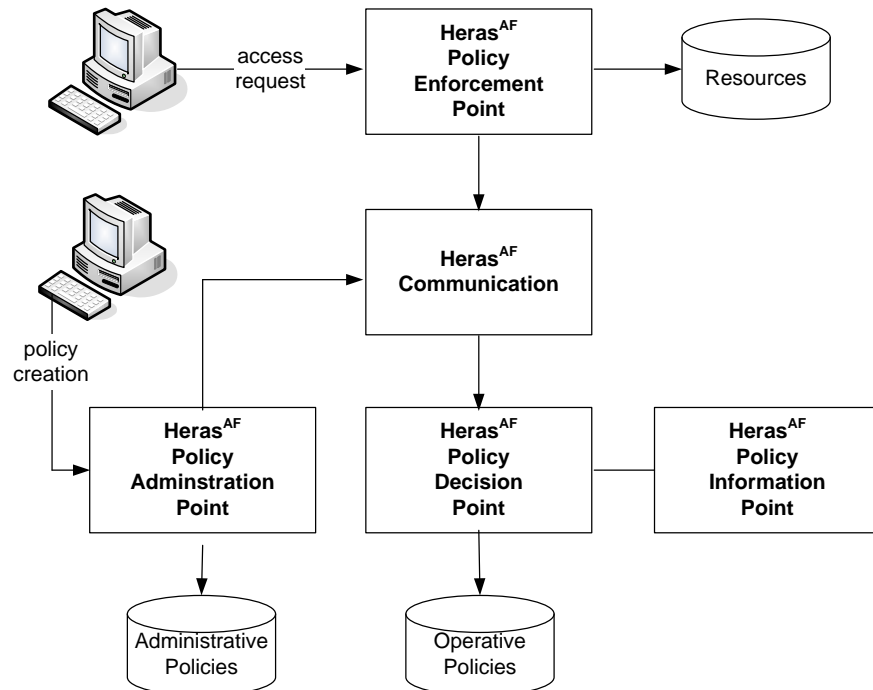
A proof of concept was made by Yan Graf [Graf06] and René Eggenschwiler [Egg06] in mid 2006. They built nearly all the components as prototypes for a working HERAS^{AF}, based on Sun's XACML implementation.

In 2007 Massimo Cerqui and Sandro Strelbel 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: HERAS^{AF} 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-PDP Communication

Introduction

There are several requirements to an implementation of the communication between PEPs and PDPs. This chapter lists and explains the most important requirements.

Functional requirements

Requirements

SOAP, SAML & XACML	The Web Service must be able to send and receive standard-compliant XML messages. These messages must contain SOAP 1.1 data that encapsulates XACML 2.0 data by using the SAML 2.0 Profile of XACML. See Part 3 chapter 4.2.2 for more information.
SAML elements	The Web Service server and client must provide a method to include the handling of SAML elements. See Part 3 chapter 4.2.2 and continues chapters for more information.
PEP integration	The Web Service client must be able to be integrated into the PEP architecture to evaluate a given XACML request by the remote Web Service server and then return an XACML response.
WS Security	It must be possible to add Web Service Security aspects to the Web Service server and client.

Non-functional requirements

Usability

Understandability	The API of the HERAS ^{AF} PEP-PDP Communication must be documented with JavaDoc in English. The documentation should be understandable for any developer. 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 where and how it can be extended. The user's guide will show how to use and configure the HERAS ^{AF} PEP-PDP Communication 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-PDP Communication 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	<p>The PEP-PDP Communication cannot tolerate any faults. The security depends on the PEP-PDP Communication as much as where the policies are evaluated.</p> <p>If a system exception inside the PEP-PDP Communication arises it must be thrown to the PEP and access to the resource must always be denied.</p>
-----------------	---

2.4 Module Design

Overview The implementation of the PEP-PDP communication is divided up in three modules:

- HERAS^{AF} PDP Context-WS
- HERAS^{AF} PEP Web Service (WS)
- HERAS^{AF} SAML, a common module used by both of the above.

This chapter gives a short overview of these three modules and their purpose.

Module overview

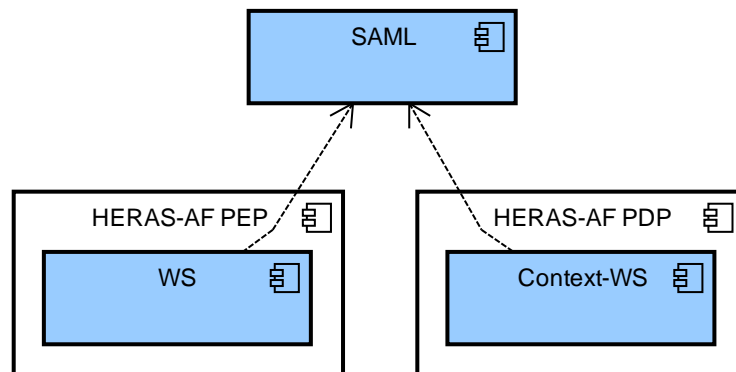


Figure 2: Modules of HERAS^{AF} PEP-PDP Communication

Modules

Context-WS (HERAS^{AF} PDP)

This module is part of the HERAS^{AF} PDP and contains the Web Service endpoint. This endpoint can be mapped to a Web Service interface, where PEPs can send decision request to.

WS (HERAS^{AF} PEP)

This module is an optional component of the HERAS^{AF} PEP and contains a Web Service client and its integration into the HERAS^{AF} PEP.

SAML

This module contains JAXB classes to marshal and unmarshal SAML data. In



In addition, there are classes that are useful when the handling of SAML must be compliant with the standard. Both other Web Service modules depend on this one.

Part II: Configuration

1 Overview

Introduction

The HERAS^{AF} PDP Context-WS endpoint and the HERAS^{AF} PEP WS client are configurable through the Spring Framework [Spring] application context.

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 different configuration topics:

HERAS^{AF} PEP WS client configuration and integration

The configuration of the WS client and its integration into the HERAS^{AF} PEP.

HERAS^{AF} PDP Context-WS endpoint configuration

The configuration of the WS endpoint and its mapping to a Spring Web Service.

2 HERAS^{AF} PEP WS client configuration and integration

Overview

Just like the HERAS^{AF} PEP, the HERAS^{AF} PEP WS client uses the Spring Framework to wire and configure its modules.

This chapter only describes the HERAS^{AF} PEP WS client configuration and its integration into the HERAS^{AF} PEP. All other parts of the HERAS^{AF} PEP configuration are described in the HERAS^{AF} PEP Developer's Guide. [RW08PEPDev]

Common elements

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

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 1: Bean namespaces

2.1 Configuration files

Overview

This chapter shows how to embed the configuration file for the WS client (Pdp.xml) into the application context of the Spring Framework.

code structure

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

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 the configuration file.
--------	--

2.2 HERAS^{AF} PEP WS client

Overview This chapter shows an example WS client configuration and how the Pdp implementation is integrated into the HERAS^{AF} PEP.

code structure

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

<bean id="pdp"
class="org.herasaf.pep.integration.ws.SamlPdp">
  <property name="clientSamlHandler">
    <ref bean="clientSamlHandler" />
  </property>
</bean>
<bean id="clientSamlHandler"
class="org.herasaf.pep.integration.ws.saml.
DefaultClientSamlHandler">
  <property name="remotePdpClient"
ref="springWsClient" />
</bean>

<bean id="springWsClient"
class="org.herasaf.pep.integration.ws.springws.
SpringWsClient">
  <property name="marshaller" ref="marshaller" />
  <property name="unmarshaller" ref="marshaller" />
  <property name="defaultUri"
value="http://example.com/pdpservice" />
  <property name="actionUri"
value="http://example.com/pdpservice/evaluate" />
  <property name="messageSenders">
    <bean
class="org.springframework.ws.transport.http
```

```

CommonsHttpMessageSender">
    <property name="httpClient">
        <bean
            class="org.apache.commons.httpclient.HttpClient">
            <property name="params"
                ref="soapSamlHttpClientParams" />
            </bean>
        </property>
    </bean>
</property>
<property name="interceptors">
    <bean
        class="org.herasaf.saml.springws.SoapActionClientIn
terceptor" />
    </property>
</bean>

<bean id="marshaller"
    class="org.herasaf.saml.marshaller.Jaxb2NamespaceMa
rshaller">
    <property name="contextPath"
        value="org.herasaf.xacml.core.context.impl:org.hera
saf.saml.jaxb.saml.assertion:org.herasaf.saml.jaxb.saml.p
rotocol:org.herasaf.saml.jaxb.w3.xmlsig:org.herasaf.saml
.jaxb.w3.xmlenc:org.herasaf.saml.jaxb.xacml.assertion:org
.herasaf.saml.jaxb.xacml.protocol" />
</bean>

<bean id="soapSamlHttpClientParams"
    class="org.herasaf.saml.springws.SoapSamlHttpClient
Params" />

```

Example 3: Example of a HERAS^{AF} PEP WS configuration

Element description

Element description	
Bean with id <code>pdpLocator</code>	Defines which implementation of a <code>PdpLocator</code> is used. The <code>DefaultPdpLocator</code> has the following properties: <ul style="list-style-type: none"> • <code>pdp</code> to inject a <code>Pdp</code> implementation See [RW08PEPDev] for more details.
Bean with id <code>pdp</code>	Defines which implementation of a <code>Pdp</code> is used. The <code>SamlPdp</code> of the WS client has the following properties: <ul style="list-style-type: none"> • <code>clientSamlHandler</code> to inject a <code>ClientSamlHandler</code> implementation
Bean with id <code>clientSamlHandler</code>	Defines which implementation of a <code>ClientSamlHandler</code> is used. The <code>DefaultClientSamlHandler</code> has the following properties: <ul style="list-style-type: none"> • <code>remotePdpClient</code> to inject a

`SamlRemotePdpClient`

Bean with
`springWsClient`

Defines which implementation of a `SamlRemotePdpClient` is used. The `SpringWsClient` extends `WebServiceGatewaySupport` of the Spring WS framework and has the following properties:

- `defaultUri` to set the URI of the remote PDP's Web Service
- `actionUri` to set the URI of the remote PDP's WS-Addressing Action
- `marshaller` to set the marshaller for XML
- `unmarshaller` to set the unmarshaller for XML
- `messageSenders` to set the `MessageSender`
- `interceptors` to set an Interceptor

See [SpringWs] for details about all the other properties of `WebServiceGatewaySupport`.

Bean with
`marshaller`

Defines which implementation of a `Marshaller` is used. The `Jax2NamespaceMarshaller` has a property:

- `contextPath` to set the path to the JAXB2 classes.

Bean with
`soapSamlHttpClientParams`

Defines which implementation of a `HttpClientParams` is used.

3 HERAS^{AF} PDP Context-WS endpoint configuration

Overview The HERAS^{AF} PDP Context-WS endpoint uses the Spring Framework to wire and configure its modules.

This chapter only describes the WS endpoint configuration and its mapping to a Spring Web Service as well as the WSDL definition. See [SpringWs] for all details on how to configure a Spring-WS.

3.1 Web Application Deployment Descriptor

Overview This chapter shows an example of a web application deployment descriptor (web.xml). To define a Spring-WS, this descriptor must redirect all incoming requests to the `MessageDispatcherServlet`.

code structure

```
<?xml version="1.0" encoding="UTF-8"?>

<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
  http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
  version="2.4">

  <display-name>HERAS-AF PDP Context-WS</display-
  name>
  <servlet>
    <servlet-name>spring-ws</servlet-name>
    <servlet-class>
      org.springframework.ws.transport.http.
      MessageDispatcherServlet
    </servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>spring-ws</servlet-name>
    <url-pattern>/*</url-pattern>
  </servlet-mapping>

</web-app>
```

Example 4: Deployment Descriptor (web.xml)

3.2 HERAS^{AF} PDP Context-WS endpoint

Overview This chapter shows an example of the Spring-WS configuration file and how to configure and map the HERAS^{AF} PDP Context-WS endpoint to the Spring-WS.

code structure

```
<context:annotation-config />

<bean id="mapping"
```

```

class="org.springframework.ws.soap.addressing.
server.SimpleActionEndpointMapping">
  <property name="mappings">
    <util:map>
      <entry key="http://localhost:8080/herasaf-
pdp-ws-context/pdp-service/evaluate" value=" pdpEndpoint "
/>
    </util:map>
  </property>
</bean>

<bean id="pdpEndpoint"
class="org.herasaf.pdp.ws.context.saml.SamlPdpEndpoint">
  <property name="samlHandler"
ref="serverSamlHandler" />
</bean>

<bean id="serverSamlHandler"
class="org.herasaf.pdp.ws.context.saml.DefaultServerSamlH
andler" depends-on="initDb">
  <property name="pdp" ref="pdpImpl" />
  <property name="issuer">
    <bean
class="org.herasaf.saml.jaxb.saml.assertion.NameIDType">
      <property name="value" value="HERAS-AF
Policy Deployment Point" />
    </bean>
  </property>
</bean>

<bean id="contextWs"
class="org.springframework.ws.wsdl.wsdl111.SimpleWsdl111Def
inition">
  <constructor-arg value="/WEB-INF/wsdl/herasaf-
ws.wsdl" />
</bean>

```

Figure 3: Example of a HERAS^{AF} PDP Context-WS endpoint configuration

Element description

<code><context:annotation-config /></code>	<p>Automatically register all of Spring's standard post-processors for annotation-based configuration.</p> <p>This element must be set to ensure that Spring will throw an exception if a mandatory property of a bean is not set.</p>
Bean with id <code>mapping</code>	<p>Defines the implementation used for the mapping of endpoints. The <code>SimpleActionEndpointMapping</code> has the following property:</p> <ul style="list-style-type: none"> <code>mappings</code> to set a map where the URI is the key and the value the endpoint. <p>The URI represents the action address of WS-Addressing. See [SpringWs] for details about all the other</p>

Bean with id
`pdpEndpoint`

properties of `SimpleActionEndpointMapping`.
Defines our WS endpoint. The `SamlPdpEndpoint`
has the following property:

- `serverSamlHandler` to inject a
`ServerSamlHandler`

Bean with id
`serverSamlHandler`

Defines which implementation of a
`ServerSamlHandler` is used. The
`DefaultServerSamlHandler` has the following
properties:

- `pdp` to inject a `PDP`
- `issuer` to inject a `NameIDType`

Bean with id
`contextWs`

Defines the service contract for the Spring-WS. The
`SimpleWsdll11Definition` has a constructor
with the following parameters:

- `wsdlResource` to inject a `Resource` that
defines the WSDL

See [SpringWs] for more information how to define
a Spring-WS service contract.

Part III: Architecture & Design

1 Overview

Introduction

This part describes the architecture and design of HERAS^{AF} PEP-PDP Communication. Each of the following chapters will discuss one of the three modules.

Module overview

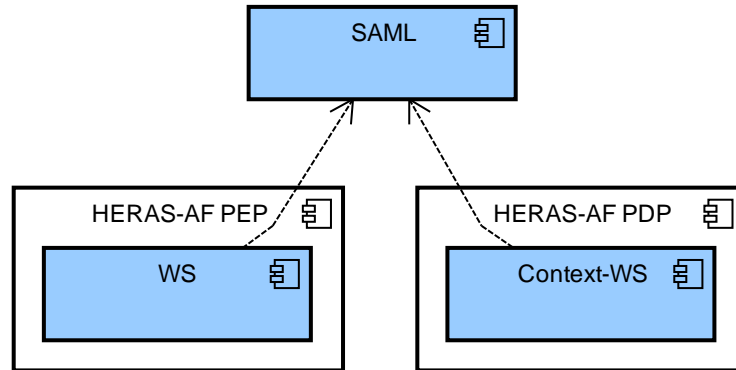


Figure 4: Modules of HERAS^{AF} PEP-PDP Communication

2 HERAS^{AF} PDP Context-WS

Overview

This chapter describes the HERAS^{AF} PDP Context-WS module.

This module is part of the HERAS^{AF} PDP and contains the Web Service endpoint. A Web Service interface mapped to this endpoint enables a PDP to receive decision requests from remote PEP.

Most parts of the Web Service server logic is implemented with components of Spring Web Services 1.5. It is therefore recommended to read the Spring WS Reference [SpringWs] to fully understand the purpose and functionality of each component.

Package and class diagram

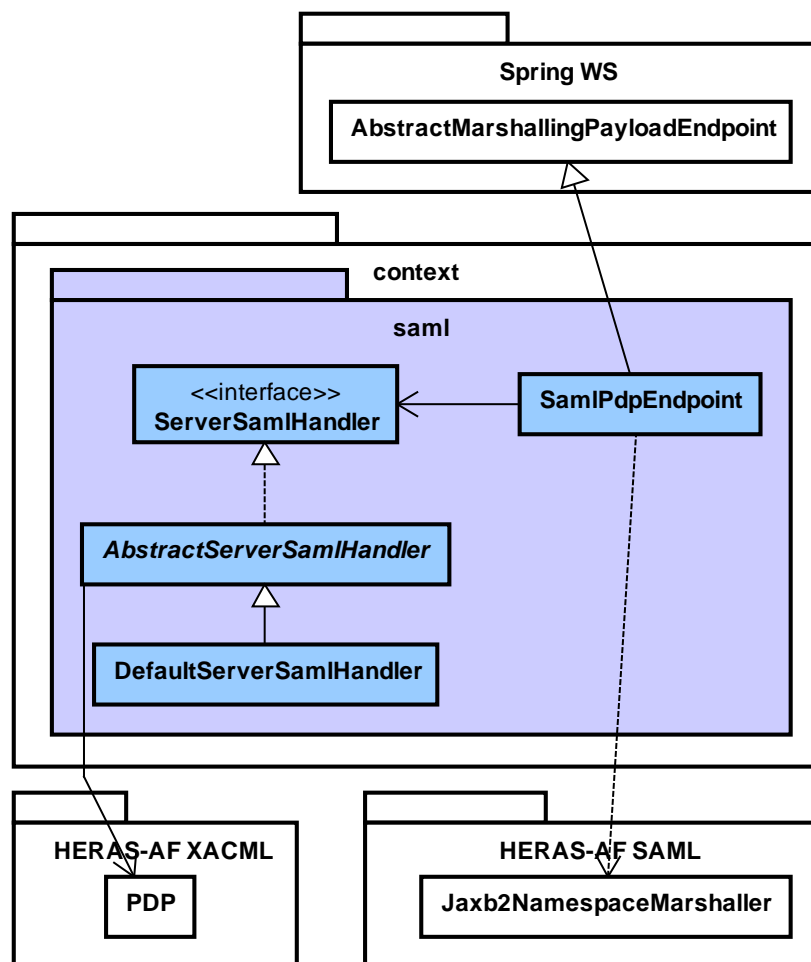


Figure 5: Packages and classes of the PDP Context-WS module

Package description

saml

This package contains all classes related to the SAML aspects of the PDP's communication with a remote PDP.

2.1 Scenario

This scenario shows the role of each PDP Context-WS component in the PEP-PDP Communication.

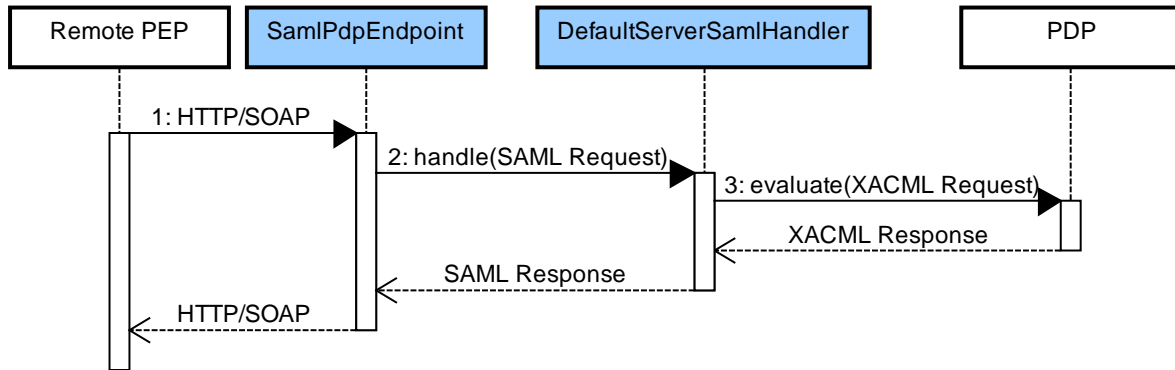


Figure 6: Flow of the PDP Context-WS module

Description

Call 1:

A remote PEP sends a SOAP message over HTTP to the Web Service interface of a PDP. The Spring-WS redirects the transmitted SOAP to the `SamlPdpEndpoint` who unmarshalls and processes the SAML contained in the SOAP envelope.

Call 2:

The `SamlPdpEndpoint` delegates the handling of the SAML request to the `DefaultServerSamlHandler`. This handler validates the SAML request and extracts the XACML request contained in it.

Call 3:

The `DefaultServerSamlHandler` delegates the extracted XACML requests to a PDP implementation for evaluation.

Return 3:

The PDP returns the XACML response to the `DefaultServerSamlHandler`.

Return 2:

The `DefaultServerSamlHandler` returns the standard-compliant SAML response, containing the XACML response, to the `SamlPdpEndpoint`.

Return 1:

The `SamlPdpEndpoint` returns the XML data of the SAML response to the Spring-WS. The Spring-WS wraps the XML data into a SOAP message and sends it to the remote PEP.

2.2 Important classes and interfaces

Overview This chapter explains all important classes and interfaces inside the PDP Context-WS module.

Only some of classes and methods are described, more descriptions can be found in the JavaDoc.

2.2.1 ServerSamlHandler

<<interface>> ServerSamlHandler
+ <i>handle(samlRequest : XACMLAuthzDecisionQueryType) : ResponseType</i>

Description The `ServerSamlHandler` interface defines handler for the SAML aspects on the PDP side of the PEP-PDP Communication. Implementations of this interface can handle a SAML request and return a SAML response.

This interface is used by the `SamlPdpEndpoint` to delegate the handling of incoming SAML requests and to render a corresponding SAML response that is to be returned to the PEP.

2.2.2 AbstractServerSamlHandler

AbstractServerSamlHandler
+ <i>handle(samlRequest : XACMLAuthzDecisionQueryType) : ResponseType</i> # <i>getSamlStatus(decision : XACMLAuthzDecisionStatementType) : StatusType</i> # <i>processSamlRequest(samlRequest : XACMLAuthzDecisionQueryType) : void</i> # <i>processSamlResponse(samlResponseType : ResponseType) : void</i> + <i>setPdp(pdp : PDP) : void</i> + <i>setIssuer(issuer : NameIDType) : void</i>

Description The `AbstractServerSamlHandler` class provides a template implementation to handle all SAML aspects in a standard-compliant way. The passed SAML request is validated and the XACML request is extracted. Then the `PDP` property is used to render an XACML response that can be wrapped into a SAML response and returned to the caller.

Implementations of this class must implement the two template methods `processSamlRequest()` and `processSamlResponse()`. They allow it to process the SAML request after it has been received and validated, and the SAML response before it is returned. This provides the possibility to handle additional optional features of SAML, like encryption or signatures.

2.2.3 SamlPdpEndpoint

SamlPdpEndpoint
<pre>+ SamlPdpEndpoint() # invokeInternal(request : Object) : Object + setSamlHandler(samlHandler : ServerSamlHandler) : void</pre>

Description

The `SamlPdpEndpoint` class extends the `AbstractMarshallingPayloadEndpoint` of the Spring WS framework and can be mapped to a Spring-WS to handle incoming messages. XML data passed to this endpoint is automatically unmarshalled before `invokeInternal()` is called.

On default this endpoint is initialized with a `Jaxb2NamespaceMarshaller` of the HERAS^{AF} SAML module.

The further handling of incoming SAML requests is simply delegated to a `ServerSamlHandler`.

2.3 Design Decisions

2.3.1 WSDL exposure

Overview

The WSDL describes the Web Service. A client who wants to use the Web Service knows with this description, which elements and operations can be used.

Spring WS provides two alternatives to create and expose the WSDL description for a Web Service.

Alternatives

Alternative 1: WSDL generation on startup and exposure through Spring

Spring WS provides a class where an XML schema or multiple schemas can be processed to generate a WSDL description. Additional information like the description of the operations or extended WSDL description must be set through setter-methods (e.g. publishing path). This means that the XSD schemas do not provide enough information.

The class exposes the WSDL on a particular URL.

The WSDL generation will happen in the application context creation phase of Spring.

This alternative would imply that the OASIS XACML, SAML and SAML Profile would be used without any changes.

Advantage

- Less costs (no need to learn how to write WSDL)

Disadvantage

- WSDL is huge and unmanageable
- Application context startup

takes longer

- WSDL would be generated more than once

Alternative 2: WSDL creation by user and exposure through Spring

Spring WS provides a class where a complete WSDL can be set in a property. Some additional information can be set with the setter-methods.

The class exposes the WSDL on a particular URL.

The WSDL must be written by hand or generated with other tools. The WSDL contains all the needed elements and there is no need to set some information via the Application Context.

Advantage

- WSDL is manageable and just contains the needed parts.
- WSDL defines everything in its own file
- Application Context startup is faster than the one with alternative 1

Disadvantage

- Higher costs (need to learn how to write WSDL)
- Configuration of the WSDL is distributed in two files (WSDL itself and Application Context file)

Decision

The chosen solution is to create the WSDL by hand (alternative 2). The description should be manageable and should just contain the needed parts. This solution also makes sure, that the configuration of the Web Service is kept easy (application context configuration is less complex). Changing default settings by the user must be made directly in the WSDL file.

3 HERAS^{AF} PEP Web Service

Overview

This chapter describes the HERAS^{AF} PEP WS module.

This module is an optional part of the HERAS^{AF} PEP and contains the Web Service client. Integrating this module into a PEP, enables the PEP to send decision requests to a remote PDP for evaluation.

Parts of the Web Service client logic are implemented with components of Spring Web Services. It is therefore recommended to read the Spring WS Reference [SpringWs] to fully understand the purpose and functionality of each component.

Package and class diagram

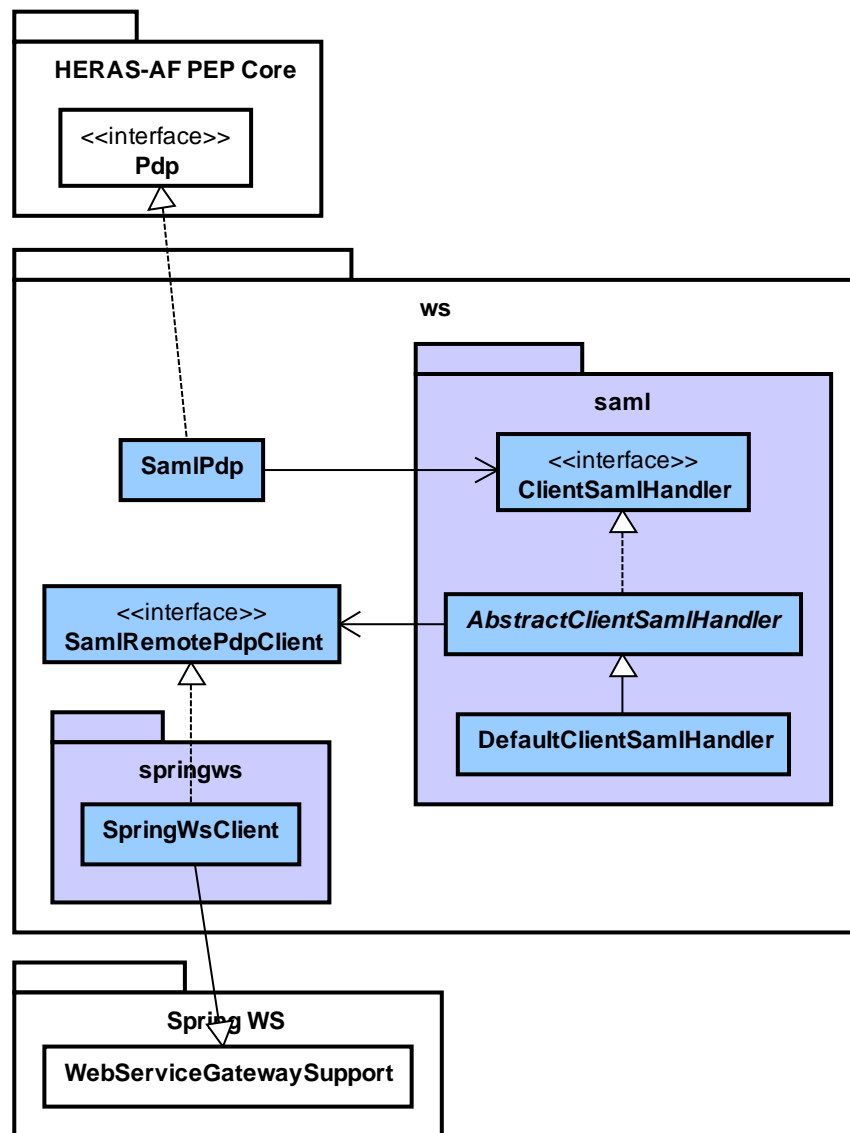


Figure 7: Packages and classes of the PEP WS module

Package description

Package: ws

This package contains the interfaces and implementations for the integration of the Web Service client into the HERAS^{AF} PEP.

Package: saml

This package contains all classes related to the SAML aspects of the PEP's communication with a remote PDP.

Package: springws

This package contains all classes of the Web Service client related to Spring WS.

3.1 Szenario

This scenario shows the role of each PEP WS component in the PEP-PDP Communication when plugged into a HERAS^{AF} PEP.

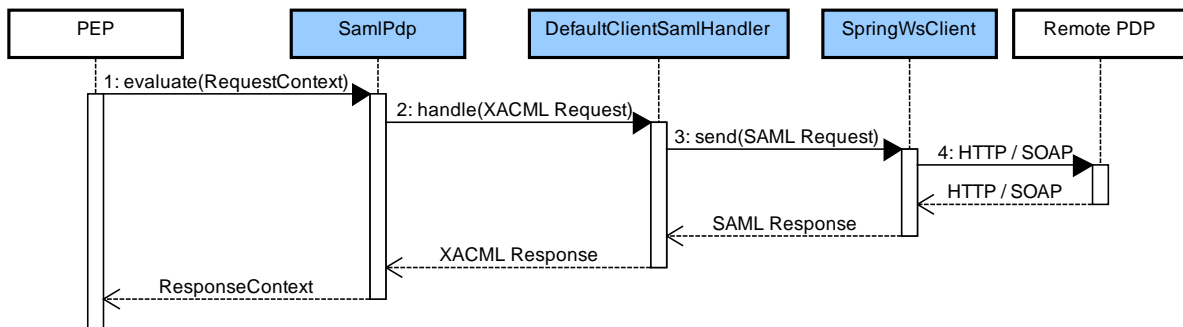


Figure 8: Flow of the PEP WS module

Description

Call 1:

The PEP passes a `RequestContext` to the `SamlPdp` for evaluation. The `SamlPdp` then extracts the XACML request from the `RequestContext`.

Call 2:

The `SamlPdp` passes the XACML request to the `DefaultClientSamlHandler` for the handling of all SAML aspects. This `DefaultClientSamlHandler` wraps the XACML request into a standard-compliant SAML request.

Call 3:

The `DefaultClientSamlHandler` passes the SAML request to the `SpringWsClient` for the marshalling into XML data and the creation of a SOAP message.

Call 4:

The `SpringWsClient` sends the XML data of the SAML request in a SOAP envelope to the evaluate Web Service of the remote PDP.

Return 4:

The remote PDP returns the SOAP message, containing the XML data of the SAML response, to the `SpringWsClient`.

Return 3:

The `SpringWsClient` returns a SAML response, containing the XACML response from the PDP, to the `DefaultClientSamlHandler`.

Return 2:

The `DefaultClientSamlHandler` returns an XACML response, created by a remote PDP, to the `SamlPdp`.

Return 1:

The `SamlPdp` returns the `ResponseContext`, with all data from the XACML response, to the PEP.

3.2 Important classes and interfaces

Overview

This chapter explains all important classes and interfaces inside the PEP Ws module.

Only some of classes and methods are described, further descriptions can be found in the JavaDoc.

3.2.1 ClientSamlHandler

*Description*

The `ClientSamlHandler` interface defines handler for the SAML aspects on the PEP side of the PEP-PDP Communication. Implementations of this interface can handle a XACML request and return a XACML response.

This interface is used by the `SamlPdp` to delegate the handling of SAML aspects like wrapping an XACML request into a SAML request for the transmission to a PDP.

3.2.2 AbstractClientSamlHandler

<i>AbstractClientSamlHandler</i>
+ handle(xacmlRequest : RequestType) : ResponseType # processSamlRequest(samlRequest : XACMLAuthzDecisionQueryType) : void # processSamlResponse(samlResponseType : ResponseType) : void + setRemotePdpClient(remotePdpClient : SamlRemotePdpClient) : void

Description

The `AbstractClientSamlHandler` class provides a template implementation to handle all SAML aspects in a standard-compliant way. The passed XACML request is wrapped into a SAML request. Then the `RemotePdpClient` is used to send the SAML request to a remote PDP and render a SAML response. The XACML response is extracted from the previously validated SAML response and returned to the caller.

Implementations of this class have to implement the two template methods `processSamlRequest()` and `processSamlResponse()`. They allow it to process the SAML request before it is sent to the PDP and the SAML response after it has been received and validated. This provides the possibility to handle additional optional features of SAML, like encryption or signatures.

3.2.3 SamlRemotePdpClient

<<interface>> SamlRemotePdpClient
+ sendXACMLAuthzDecisionQuery(query : XACMLAuthzDecisionQueryType) : ResponseType

Description

The `SamlRemotePdpClient` interface defines a class that is able to send a SAML request to a remote PDP and returns the received SAML response.

This interface is used by the `AbstractClientSamlHandler` to send and receive SAML requests/responses without knowing which transportation technology (e.g. Web Services, RMI, etc.) is actually used.

3.2.4 SamlPdp

SamlPdp
+ evaluate(requestContext : RequestContext) : ResponseContext + setClientSamlHandler(samlHandler : ClientSamlHandler) : void

Description

The `Sam1Pdp` class implements the `Pdp` interface of the HERAS^{AF} PEP module. This allows integrating this Web Service client into a PEP to evaluate decision requests by a remote PDP.

Attention: The PEP WS module heavily depends on the JAXB classes of the HERAS^{AF} XACML module. Therefore `evaluate()` only accepts instances of `HerasRequestContext` to be passed.

4 HERAS^{AF} SAML

Overview

This chapter describes the HERAS^{AF} SAML module.

This module contains classes that are useful when interacting with SAML data.

Package and class diagram

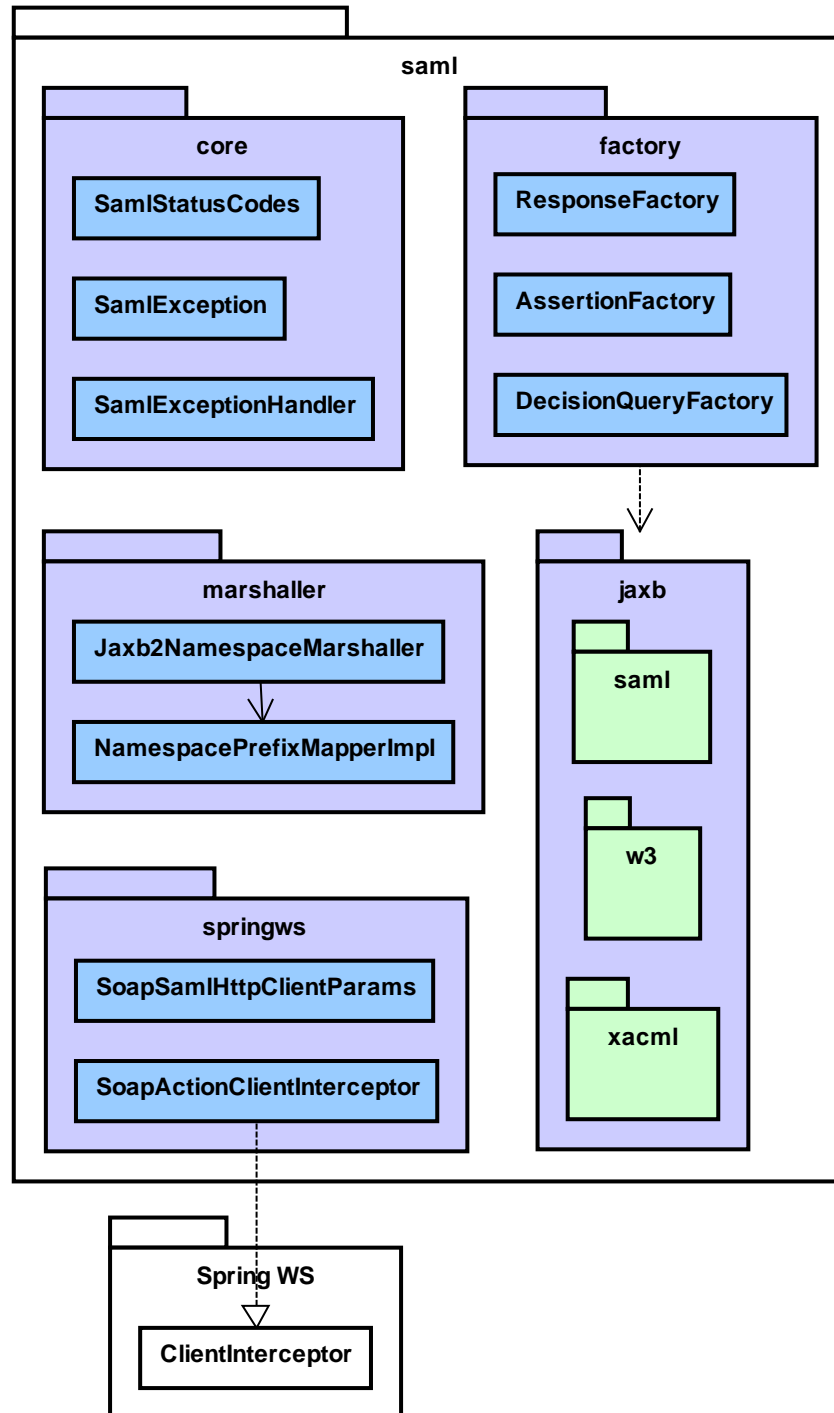


Figure 9: Packages and classes of the SAML module

*Package
description*

Package: core

This package contains the SAML status codes as well as the exception handling.

Package: jaxb

This package contains the generated JAXB 2.0 classes for SAML 2.0 [SAML] (incl. XML digital signatures and encryption) and the SAML 2.0 Profile of XACML [SAMLXACML].

Package: factory

This package contains factories to create and pre initialize certain SAML JAXB objects to guarantee that mandatory fields are set properly. These requirements are specified by the SAML Bindings 2.0 [SAMLBind].

Package: marshaller

This package contains classes for a customized Jaxb2Marshaller. The customization allows to always use a specific prefix for a namespace.

Package: springws

This package contains classes that implement or extend Spring WS classes to fulfill SAML-specific requirements for SAML over SOAP over HTTP. The requirements are specified by the SAML Bindings 2.0 and the SAML 2.0 Profile for XACML 2.0.

See the Spring WS Reference [SpringWs] for more details about the components used or extended by classes of this package.

4.1 Important classes and interfaces

Overview

This chapter explains all important classes and interfaces inside the SAML module.

Only some of classes and methods are described, further descriptions can be found in the JavaDoc.

4.1.1 DecisionQuery-, Response- and AssertionFactory

DecisionQueryFactory
+ create(request : RequestType) : XACMLAuthzDecisionQueryType + create(request : RequestType, issuerInstant : XMLGregorianCalendar) : XACMLAuthzDecisionQueryType

ResponseFactory
+ create(status : StatusType) : ResponseType + create(status : StatusType, issuerInstant : XMLGregorianCalendar) : ResponseType

AssertionFactory
+ create(issuer : NameIDType) : AssertionType + create(issuer : NameIDType, issuerInstant : XMLGregorianCalendar) : AssertionType

Description

The `DecisionQueryFactory`, `ResponseFactory` and `AssertionFactory` classes allow the creation of the following pre initialized JAXB objects:

- `XACMLAuthzDecisionQueryType`
- `ResponseType`
- `AssertionType`

Instances of these classes created by the factories are compliant with SAML 2.0 and the SAML 2.0 Profile of XACML. The following required attributes get initialized on creation:

- ID (a random integer number)
- Version ("2.0")
- IssueInstant (the current time in XML format)
- Request (*only for XACMLAuthzDecisionQueryType*)
- Status (*only for ResponseType*)
- Issuer (*only for AssertionType*)

4.1.2 SamlException

SamlException
+ SamlException(statusCode : String) + SamlException(statusCode : String, message : String) + SamlException(statusCode : String, message : String, cause : Throwable) + SamlException(statusCode : String, cause : Throwable) + getStatusCode() : String

Description The `SamlException` class indicates an unexpected error related to the handling of SAML. The error is described by a SAML status code. All possible status codes that can be returned in a SAML response can be found as constants in the `SamlStatusCodes` class.

4.1.3 SamlExceptionHandler

SamlExceptionHandler
+ createErrorResponse(exception : SamlException) : ResponseType

Description The `SamlExceptionHandler` class is a utility to simplify the handling of a `SamlException`. It creates and returns a SAML response containing the status code of the specified `SamlException`.

4.2 Design Decisions

4.2.1 Marshalling/Unmarshalling the XML

Overview XACML requests and responses are sent in XML format inside a SAML message. These are represented in data structures on server and on client side.

A process of transforming XML into an object as well as from an object into XML is needed. The technical term is un-/marshalling. A marshaller serializes an object to XML, and an unmarshaller deserializes XML data to object.

Alternatives **Alternative 1: Usage of the HERAS^{AF} XACML JAXB classes and some additional JAXB generated classes**

The JAXB class generation from the XML schemas, that define SAML and XACML could be reduced because all the already generated classes for XACML could be referenced from HERAS^{AF} XACML. This would result in faster processing of the XML as well as no duplicated code whenever only

HERAS^{AF} components are used. Figure 10 shows this approach.

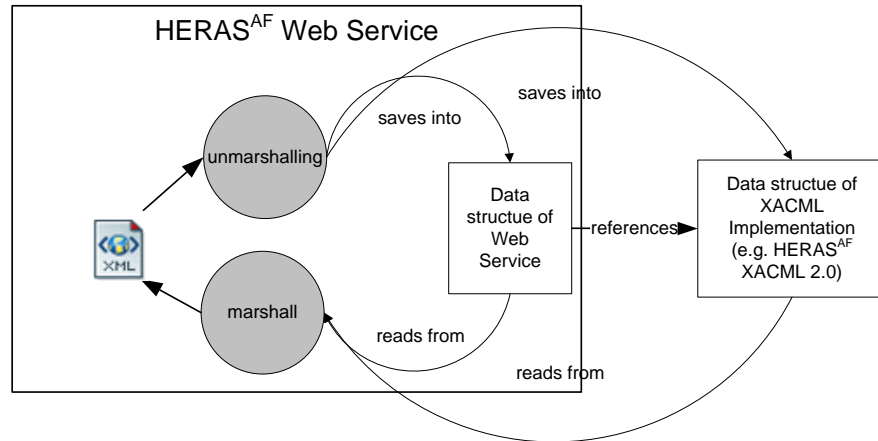


Figure 10: Marshalling alternative 1

Advantage

- Faster processing
- No “duplicated code” in HERAS^{AF}

Disadvantage

- Dependency to HERAS^{AF} XACML
- No abstraction possibility to use the Web Service without HERAS^{AF} XACML. Pluggability for other XACML implementation is gone.

Alternative 2: Regenerate all JAXB classes

All the JAXB classes are generated from the XML Schemas. This would result in no dependency to any XACML implementation. This would enable a further abstraction layer which gives the Web Service the possibility to use any PDP implementation.

The further abstract layer is not for free. Every single un-/marshall process would be slower, as it needed to be transformed into the data structure of the Web Service first and from there into the data structure of the concrete XACML implementation. Figure 11 shows this approach.

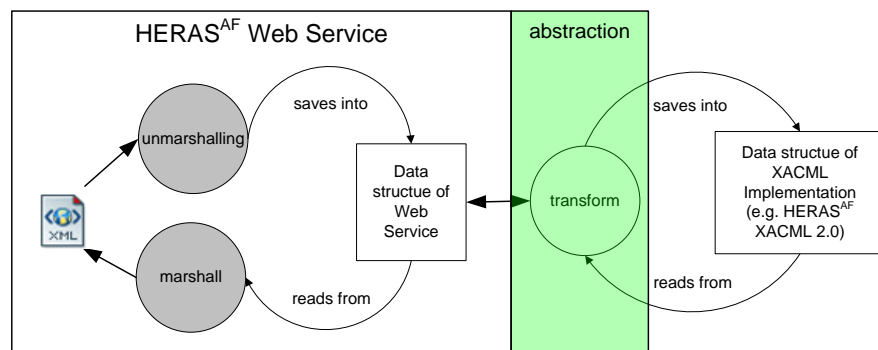


Figure 11: Marshalling alternative 2

Advantage

- No dependency to any XACML

Disadvantage

- Slower processing because the XML processing goes through

implementation	one more abstract layer. (e.g. XACML XML→Web Service data structure→XACML implementation data structure)
<ul style="list-style-type: none"> Abstraction possibility. 	<ul style="list-style-type: none"> Duplicated code inside HERAS^{AF}

Decision The chosen solution is to implement alternative 1 because a dependency to HERAS^{AF} XACML 2.0 doesn't hurt the design (Web Service is concrete for a HERAS^{AF} implementation) and the requirement is that the processing of the XACML is fast.

4.2.2 SAML standard-compliance

Overview Aside from XML schemas, SAML 2.0 and the SAML 2.0 Profile of XACML define many constraints for certain elements and attributes on what values they must contain.

There are two alternatives to assure that newly created JAXB objects are standard-compliant.

Alternatives

Alternative 1: Factories for each JAXB class.

For each JAXB class a factory is created. These factories define methods that only allow the creation of JAXB objects that are compliant with SAML 2.0 and the SAML 2.0 Profile for XACML.

Advantage

- JAXB objects created with the corresponding factory are standard-compliant.

Disadvantage

- JAXB objects created without the corresponding factory are not standard-compliant.

Alternative 2: Adapt the JAXB implementations

The implementation of all JAXB classes and JAXB object factories for elements constrains are changed, so that new instances of JAXB objects can only be created if they are compliant with SAML 2.0 and the SAML 2.0 Profile of XACML.

Advantage

- New instances of JAXB classes are guaranteed to be standard-compliant.

Disadvantage

- Could lead to runtime exceptions when usingmarshallers or unmarshallers who cannot create new JAXB objects with non-default constructors.

Decision The chosen solution is to implement factories for the creation of certain JAXB classes (alternative 1, with the exception that we only implement factories for some root JAXB classes). We did not choose alternative 2 because of the

uncertainty about how Mashallers and Unmarshallers would react to changes in the creation of the JAXB objects.

The factories of alternative 1 should only be seen as utility classes. Classes that handle SAML are still the ones responsible for the proper usage of the JAXB classes, but they can be sure that when using one of the factories, that the created JAXB object is standard-compliant and sets all required attributes and elements with proper values.

Part IV: 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.

2 Module and Unit Tests

Overview This chapter describes the module and unit tests of the HERAS^{AF} PEP-PDP Communication components.

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 do not 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 HERAS^{AF} PDP Context-WS

2.1.1 TestAbstractServerSamlHandler

Test cases

testResponseStatusCode
Tests if the <code>AbstractServerSamlHandler</code> properly sets the status code of the SAML response depending on the status code of the XACML response returned from the injected PDP implementation.
testVersionMismatch

Tests if the `AbstractServerSamlHandler` checks the version of SAML requests and returns a SAML response with a status code indicating a version mismatch.

2.2 HERAS^{AF} PEP WS

2.2.1 TestAbstractClientSamlHandler

Test cases

testDefault

Tests if the `AbstractClientSamlHandler` properly handles XACML requests. The following points are checked:

- The injected `RemotePdpClient` receives a standard-compliant SAML request that contains the specified XACML request.
- The XACML response contained in the SAML response returned by the injected `RemotePdpClient` is returned to the caller.
- The template methods `processSamlRequest` and `processSamlResponse` are called.

testVersionMismatch

Tests if the `AbstractServerSamlHandler` checks the version of the SAML response received from the injected `RemotePdpClient`. The handler must return a `HerasPdpCommunicationException` with a specified message indicating a version mismatch.

testSamlRemotePdpClientException

Tests if the `AbstractServerSamlHandler` throws `HerasPdpCommunicationExceptions` thrown by the injected `RemotePdpClient`.

2.2.2 TestSamlPdp

Test cases

testDefault

Tests if the `SamlPdp` returns a `HerasResponseContext`.

testClientSamlHandlerException

Tests if the `SamlPdp` throws `HerasPdpCommunicationExceptions` thrown by the injected `ClientSamlHandler`.

testWrongRequestContext

Tests if the `AbstractServerSamlHandler` throws a `HerasPdpCommunicationExceptions` with a `HerasConfigurationException` as cause if something different than a `HerasRequestContext` is passed.

2.3 HERAS^{AF} SAML

2.3.1 TestSamlExceptionHandler

Test cases **testCreateErrorResponse**

Tests if the `SamlExceptionHandler` creates a SAML response containing the status code and message of the specified `SamlException`.

2.3.2 TestFactories

Test cases **testAssertion**

Tests if the `AssertionFactory` creates a standard-compliant `AssertionType` instance.

testDecisionQuery

Tests if the `DecisionQueryFactory` creates a standard-compliant `XACMLAuthzDecisionQueryType` instance.

testResponse

Tests if the `ResponseFactory` creates a standard-compliant `ResponseType` instance.

2.3.3 TestJaxb2NamespaceMarshaller

Test cases **testNameSpaceMarshaller**

Tests if the `Jaxb2NamespaceMarshaller` uses the correct prefixes for namespaces as defined by `NamespacePrefixMapperImpl`.

3 Integration Tests

Introduction The functionality of the HERAS^{AF} PEP-PDP Communication as one piece is tested with integration tests.

3.1 herasaf-pdp-ws-context-integrationtests

Overview This module tests the functionality of the HERAS^{AF} PEP-PDP Communication by integrating the HERAS^{AF} PEP WS into a HERAS^{AF} PEP and integrating the HERAS^{AF} PDP Context-WS into a PDP.

The classes in this module test all 4 business cases that can occur when invoking a method protected by HERAS^{AF} PEP.

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 HERAS^{AF} 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 HERAS^{AF} 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.

Part V: Deployment

1 Overview

Introduction To build the whole HERAS^{AF} PEP-PDP Communication modules in combination with the other HERAS^{AF} PEP and HERAS^{AF} PDP modules, 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.

The description of Maven 2 [Maven] in this guide is that detailed because the information for the configuration of Maven 2 and its plugins is spread over several web pages and books.

The following chapters explain the usage of Apache Maven 2 and the responsibilities of important projects.

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 14 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 12: 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 13: maven-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 14: 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>
```

```

    </configuration>
  </plugin>

```

Figure 15: 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 16 shows an example configuration for the above cargo-maven2-plugin configuration. The bold elements are the needed for the configuration in Figure 15.

```

<?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 16: Maven user-specific configuration for tomcat

3 herasaf-pep

Introduction

The herasaf-pep directory is the root project of the HERAS^{AF} PEP implementation. It is used to define configurations and dependencies used by all HERAS^{AF} PEP projects and to perform a Maven build.

Standard build

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

```
mvn clean install
```

4 herasaf-pdp

Introduction The herasaf-pdp directory is the root project of the HERAS^{AF} PDP implementation. It is used to define configurations and dependencies used by all HERAS^{AF} PDP projects and to perform a Maven build.

Standard build To perform a Maven build without integration tests, go to the herasaf-pdp directory in the command console and execute:

```
mvn clean install
```

Integration test build To perform a Maven build with all integration tests, go to the herasaf-pdp directory in the command console and execute:

```
mvn clean install -P integration-test
```

Context-WS intergration test build To perform a Maven build with only the integration tests for the Context-WS project, go to the herasaf-pdp directory in the command console and execute:

```
mvn clean install -P context-integration-test
```

5 herasaf-saml

Introduction The herasaf-saml directory is the project of the HERAS^{AF} SAML implementation. It is used to define configurations and dependencies of the HERAS^{AF} SAML project and to perform a Maven build.

Standard build To perform a Maven build with all the integration tests, go to the herasaf-saml directory in the command console and execute:

```
mvn clean install
```

6 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.

6.1 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 HERAS^{AF} 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

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
OASIS	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)
- [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

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)

[CS07PAP]

Massimo Cerqui, Sandro Strelbel:

HERAS^{AF}: Policy Administration Point

November 2007

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

[CS07PEP]

Massimo Cerqui, Sandro Strelbel:

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/>

<i>[MVNCargo]</i>	http://cargo.codehaus.org/Maven2+plugin
<i>[MVNJavaDoc]</i>	http://maven.apache.org/plugins/maven-javadoc-plugin/
<i>[MVNSettings]</i>	http://maven.apache.org/ref/2.0.8/maven-settings/settings.html
<i>[MVNSource]</i>	http://maven.apache.org/plugins/maven-source-plugin/
<i>[MVNSurefire]</i>	http://maven.apache.org/plugins/maven-surefire-plugin/
<i>[Q4Eclipse]</i>	http://code.google.com/p/q4e/
<i>[Spring]</i>	http://www.springframework.org/
<i>[SpringAOP]</i>	http://static.springframework.org/spring/docs/2.5.x/reference/aop.html
<i>[SpringSecurity]</i>	http://static.springframework.org/spring-security/site/index.html
<i>[SpringWs]</i>	http://static.springframework.org/spring-ws/site/
<i>[TestNG]</i>	http://www.testng.org/doc/

Appendix B: Issues, Extensions and Refactorings

1 Overview

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

2 Open Issues

Overview This chapter contains open issues of the HERAS^{AF} PEP-PDP Communication. There are no open issues for the current release.

3 Extension Points

Overview This chapter contains the possible extension points to this implementation. Listed are capabilities which could make the work with the PEP-PDP Communication more comfortable.

ReturnContext and InputContext Only The SAML 2.0 Profile of XACML defines the Boolean attributes `ReturnContext` and `InputContextOnly` that allow a PEP to request special behavior from a PDP. The `ServerSamlHandler` and `ClientSamlHandler` could be extended to support the usage of these attributes.

See page 23 of [SAMLXACML] for further information.

XML validation The SAML transmitted between PEPs and PDPs should be compliant with SAML 2.0 and the SAML 2.0 Profile of XACML. Currently only a few requirements of the specifications are being checked and there is not check if the template methods of the `ServerSamlHandler` and `ClientSamlHandler` produce valid SAML data.

A component could be introduced for the strict validation for incoming and outgoing SAML.

Support for any XACML implementation Currently the HERAS^{AF} PEP WS module depends on the `Herasafxacml` module, which integrates HERAS^{AF} XACML into the PEP. In addition the HERAS^{AF} SAML module depends on HERAS^{AF} XACML. An additional abstraction layer could be provided to enable the usage of the PEP WS module for any XACML 2.0 implementation.

We recommend an analysis of the impact such an abstraction would have on the performance and architecture of the communication.

4 Refactorings

Overview

This chapter contains refactorings which should be done to improve performance or design of HERAS^{AF} PEP-PDP Communication.

There are no refactorings for the current release.