



HERAS^{AF}

Bachelor Thesis
**HERAS^{AF}: PEP and PDP Web Service
Integration**

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

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

Disclaimer

We ensure with this disclaimer that this thesis was created by ourselves. We did not use any additional resources than the ones mentioned in the bibliography in the appendix.

The figures we used for this thesis were created by ourselves or are attached with a reference to the original author.

Each copied text phrase has an attached reference to the original text.

This thesis was not given, in this or in any similar form, to an examination board.

Acknowledgement

First we thank Wolfgang Giersche who brought us in contact with the beautiful Enterprise Computing and assisted us with this bachelor thesis by words and deeds.

We thank Florian Huonder and René Eggenschwiler for giving us the opportunity to write this bachelor thesis and for their professional advices, expert guidance and support.

We thank in particular the HERAS^{AF} developers which willingly provided information at any time.

In addition, we would like to thank everyone who supported us in writing this bachelor thesis.

Abstract

Overview	<p>Today, most applications have their own way to handle access control. This makes it hard for companies to enforce general security policies among all their applications and leads to additional costs, security risks and inconsistencies.</p> <p>HERAS^{AF} takes this opportunity and provides an Open Source architecture framework for the security aspect of enterprise applications based on the OASIS XACML 2.0 standard.</p> <p>An important component of this framework is the HERAS^{AF} Policy Enforcement Point (PEP). PEPs are responsible to enforce access control in an application and communicate with a Policy Decision Point (PDP) to receive authorization decisions for access requests.</p>
Goals	<p>The goals of this thesis are the implementation of the HERAS^{AF} PEP and the Web Service components (Endpoint & Client) for the communication between PEPs and PDPs. This contains the following points:</p> <ul style="list-style-type: none">• Analysis, design, implementation and testing of the PEP and the PEP-PDP Communication• Creation of an API to build PEPs• Web Service Endpoint for the PDP• Modular and configurable components• Bachelor Thesis, Developer's and User's Guides in English
Solution	<p>We implemented the HERAS^{AF} PEP as a framework that controls access by intercepting method calls, sending decision requests to a PDP and enforcing the returned authorization decision. The framework provides the plugability to configure and adapt all components through the Spring Framework.</p> <p>A set of Java annotations enables the user to choose which methods should be protected by the PEP and what information a request should contain. This information, acquired to create decision requests, is cached to optimize performance. Calls to protected methods are intercepted with Spring AOP.</p> <p>The standardized communication between the PEP and PDP is realized with the Spring Web Services framework. To be standard-compliant, the XACML requests/responses are wrapped into the SAML 2.0 protocol and SOAP envelopes.</p>

Table of Contents

PART I: INTRODUCTION	8
1 MANAGEMENT SUMMARY	9
1.1 INITIAL STATUS	9
1.2 REALIZATION	10
1.3 ACHIEVED GOALS	12
1.4 OUTLOOK	12
2 AIMS AND OBJECTIVES	13
2.1 GOALS OF THIS THESIS	13
2.2 APPOINTMENTS	14
3 HERAS^{AF}	15
3.1 EXTENSIBLE ACCESS CONTROL MARKUP LANGUAGE - XACML	16
3.2 HERAS ^{AF} LOGIC	20
3.2.1 HERAS ^{AF} Policy Enforcement Point	21
3.2.2 HERAS ^{AF} Policy Decision Point	21
3.2.3 HERAS ^{AF} PEP-PDP Communication	21
3.2.4 HERAS ^{AF} Policy Administration Point	22
3.2.5 HERAS ^{AF} Policy Information Point	22
4 HERAS^{AF} POLICY ENFORCEMENT POINT (PEP)	23
4.1 ACCESS INTERCEPTION	24
4.2 REQUEST INFORMATION ACQUISITION	25
4.3 DECISION REQUEST GENERATION	25
4.4 PDP COMMUNICATION	25
4.5 DECISION HANDLING	26
5 HERAS^{AF} PEP-PDP COMMUNICATION	27
5.1 STANDARD-COMPLIANCE	28
5.2 SAML HANDLING	29
5.3 WEB SERVICE HANDLING	30
5.4 SECURITY ASPECTS	30
PART II: PROJECT MANAGEMENT	32
1 PROJECT PLANNING	33
1.1 INTRODUCTION	33
1.2 OVERVIEW	33
1.3 SCHEDULE	34
1.4 RISK MANAGEMENT	37
1.5 QUALITY MANAGEMENT	38
1.6 TECHNOLOGIES AND STANDARDS	38
1.6.1 Naming Conventions	38
1.6.2 Technologies	39
1.6.3 Standards	40
1.7 PROJECT MANAGEMENT FINAL REPORT	41
2 EVALUATION	42
2.1 INTRODUCTION	42

2.2	ARCHITECTURE AND CODE QUALITY	42
2.3	TIME EVALUATION	44
PART III: TECHNICAL REPORT		46
1	ANALYSIS	47
1.1	REQUIREMENTS FOR THE PEP	47
1.2	REQUIREMENTS FOR THE PEP-PDP COMMUNICATION	49
1.3	OLD HERAS ^{AF} PEP	50
1.4	SPRING WEB SERVICE	53
1.5	KEY ASPECTS OF NEW HERAS ^{AF} PEP	55
1.5.1	Interception of a method call	55
1.5.2	Resolving request information	57
1.5.3	Parsing request information	59
1.5.4	Java annotation gathering algorithm	60
1.5.5	Attribute data types	61
1.5.6	Finding/locating a PDP	62
1.5.7	XACML request context creation	63
1.5.8	XACML response	64
1.5.9	Decision handling	64
1.5.10	Obligation handling	66
1.6	KEY ASPECTS OF THE HERAS ^{AF} PEP-PDP COMMUNICATION	68
1.6.1	Marshalling/Unmarshalling the XML	68
1.6.2	WSDL exposure	69
1.6.3	SAML standard-compliance	70
2	ARCHITECTURE	72
2.1	HERAS ^{AF} PEP	72
2.2	HERAS ^{AF} PEP-PDP COMMUNICATION	75
3	DESIGN	77
3.1	HERAS ^{AF} PEP	77
3.1.1	Core module	77
3.1.1.1	Decisions	78
3.1.2	Herasafxacml module	80
3.1.2.1	Decisions	80
3.1.3	Spring AOP Module	81
3.1.3.1	Decisions	82
3.1.4	Utilities Module	82
3.1.4.1	Decisions	83
3.2	HERAS ^{AF} PEP-PDP COMMUNICATION	84
3.2.1	WS Module	84
3.2.2	Context-WS Module	85
3.2.2.1	Decisions	85
3.2.3	Saml Module	86
3.2.3.1	Decisions	88
4	TESTS	90
4.1	UNIT TEST	90
4.1.1	Code coverage	90
4.2	INTEGRATION TESTS	90

4.2.1	Cargo-maven2-plugin	91
5	FINAL REPORT	92
5.1	SUMMARY	92
5.2	OUTLOOK	93
APPENDIX A: GENERAL		96
1	PERSONAL REPORTS	97
1.1	DANIEL REGLI	97
1.2	YANNICK WINIGER	97
2	GLOSSARY	99
3	BIBLIOGRAPHY	101
3.1	SPECIFICATIONS AND STANDARDS	101
3.2	HERAS ^{AF} DOCUMENTS	101
3.3	ARTICLES AND DOCUMENTS	103
3.4	OTHER RESOURCES	104
APPENDIX B: MINUTES OF MEETING		106
1	MEETING INFORMATION 29.02.08	107
1.1	AGENDA ITEMS	107
2	MEETING INFORMATION 07.03.08	108
2.1	AGENDA ITEMS	108
3	MEETING INFORMATION 14.03.08	109
3.1	AGENDA ITEMS	109
4	MEETING INFORMATION 28.03.08	110
4.1	AGENDA ITEMS	110
5	MEETING INFORMATION 04.04.08	111
5.1	AGENDA ITEMS	111
6	MEETING INFORMATION 11.04.08	112
6.1	AGENDA ITEMS	112
7	MEETING INFORMATION 18.04.08	113
7.1	AGENDA ITEMS	113
8	MEETING INFORMATION 25.04.08	114
8.1	AGENDA ITEMS	114
9	MEETING INFORMATION 02.05.08	115
9.1	AGENDA ITEMS	115
10	MEETING INFORMATION 09.05.08	116
10.1	AGENDA ITEMS	116
11	MEETING INFORMATION 16.05.08	117
11.1	AGENDA ITEMS	117
12	MEETING INFORMATION 22.05.08	118
12.1	AGENDA ITEMS	118
13	MEETING INFORMATION 30.05.08	119
13.1	AGENDA ITEMS	119

Part I: Introduction

1 Management Summary

1.1 Initial status

Motivation

Today, most applications have their own way to handle access control. Varying security rights for a resource needs to be handled separately on all applications. General security policies for company are hard to handle consistently on all applications. All in all this leads to additional costs, security risks and inconsistency.

A centralized policy and access control management framework is the foundation stone to reduce risks, costs and of course reduce complexity. HERAS^{AF} takes this opportunity and provides an Open Source [OpenSource] implementation for such a framework.

HERAS^{AF} uses the XACML 2.0 [XACML] standard of OASIS. An implementation of this standard has been developed in an earlier work [DOH07DA]. This implementation is used for most HERAS^{AF} components.

An important component of this framework is the HERAS^{AF} Policy Enforcement Point (PEP). PEPs are responsible to enforce access control in an application and communicate with a Policy Decision Point (PDP) to receive authorization decisions for access requests.

An earlier project realized a basic implementation of such a PEP. Since then the requirements for a HERAS^{AF} PEP changed and a completely new PEP implementation from scratch is needed.

Components for the communication between PEPs and PDPs were also realized in an earlier project. Unfortunately, the result could not be used, because the roundtrip time from the PEP to the PDP took too long. A different technology has to be found for a better implementation of the PEP-PDP Communication.

Existing work

HERAS^{AF} Policy Enforcement Point

In an earlier project [CS07PEP] an implementation of a PEP was realized. It contains the basic functionalities, but did not support the full potential of XACML 2.0. In addition, the old PEP was implemented with Sun XACML [SunXACML] and not with the newer and better HERAS^{AF} XACML 2.0.

A completely new implementation of the HERAS^{AF} PEP will be realized.

HERAS^{AF} PEP-PDP Communication

An implementation of the communication between a PEP and PDP was realized in an earlier project [DOH07SA]. Unfortunately, the result could not be used, because the roundtrip time from the PEP to PDP took too long.

A completely new implementation of the HERAS^{AF} PEP-PDP Communication will be realized.

Goals

The goal of this thesis is the implementation of the HERAS^{AF} PEP and the Web Service components (Endpoint & Client) for the communication between PEPs and PDPs. This contains the following points:

HERAS^{AF} PEP

- Analysis of the old HERAS^{AF} PEP implementation.
- Analysis, design, implementation and testing of the PEP
- Creation of a powerful API to build PEPs.
- Method interceptor with Spring AOP
- Caching strategies
- Plugability for different XACML implementations, interceptor technologies and PDPs.
- Modular and configurable components

HERAS^{AF} PEP-PDP Communication

- Analysis of the Spring Web Service Framework
- Analysis, design, implementation and testing of the PEP-PDP Communication
- Fast and reliable Web Service components (WS-Endpoint for the PDP and a WS-Client for the PEP)
- Standard-compliant transmission of XACML over SAML over SOAP
- Modular and configurable components

Documentation in English

- Bachelor Thesis
- PEP User's and Developer's Guide
- PEP-PDP Communication User's and Developer's Guide

1.2 Realization

Involved persons

Person	Function
Wolfgang Giersche	Responsible tutor
Sacha Dolski	Expert
Florian Huonder	Coach
René Eggenschwiler	Coach
Daniel Regli	Project Manager, Programmer
Yannick Winiger	Lead programmer, Quality manager

The functions project manager, programmer, lead programmer and quality manager are showing the accountability and not the responsibility. Daniel Regli and Yannick Winiger are both responsible for the realization of this bachelor thesis.

Familiarization

To become acquainted with the project, XACML 2.0, HERAS^{AF}, the often used Spring Framework [Spring] and a few new Eclipse plugins had to be understood.

Procedure

This thesis was realized with weekly iteration. The weekly iterations are arranged into five main phases. These phases are listed and described in the part below.

Phase 1: Project familiarization and setup

This phase took two part-time weeks. The setup of the project infrastructure and technology studies (e.g. Spring) was done in the first week.

The second week was the migration of the old Policy Enforcement Point to the new HERAS^{AF} XACML 2.0 implementation.

Phase 2: Analysis and prototyping of the Policy Enforcement Point

This phase took two part-time weeks. The analysis of the old PEP and a new design were done in the first week. The realization of the new PEP prototype took part in the second week. Basic tests were implemented for the prototype as well.

Phase 3: Implementation and documentation of the PEP

This phase took five part-time weeks. It began with documenting the analysis of the old PEP and the experiences with the prototype. After that the implementation of the PEP took place in the next four weeks. Integration tests were implemented at the end of this phase. Module tests were developed and run parallel to the implementation.

With this phase the development of the PEP and the documentation were finished. Just some small code changes, which included migrations to newer Spring versions, were performed.

Phase 4: Development of the PEP-PDP Communication

This phase took five part-time weeks. It contained all the development phases (Analysis, Design, Implementation, Testing and Documentation). The old PDP Web Service was studied as well as a new prototype was created in the Analysis phase. The creation contained the familiarization of new technologies like Spring Web Service and standards like SOAP [SOAP], SAML [SAML] and SAML Profile of XACML [SAMLXACML]. The prototype also proofed standard-compliance, which was very important goal to achieve.

The implementation based on the prototype design took place the next two weeks and contained additional functionality like proper Exception Handling in a Web Service.

The integration tests for the client and server modules were implemented in the end and showed a vertical breakthrough of the whole Bachelor Thesis.

The other HERAS^{AF} team, responsible for the implementation of the Policy Administration Point and the PAP-PDP Communication [NZ08PAP], had to implement a similar Web Service. Therefore, knowledge management was important to share design, knowledge and of course the experience with them.

Phase 5: Finishing

This phase took two full-time weeks. The finishing phase contained a cleanup of the code. For example, the quality of code and JavaDoc has been checked and enhanced. The documentation, the bachelor thesis poster and some other mandatory documents were completed in this phase.

1.3 Achieved goals

Results

HERAS^{AF} PEP

We implemented the HERAS^{AF} PEP as a framework that controls access by intercepting method calls, sending decision requests to a PDP and enforcing the returned authorization decision. The framework provides the plugability to configure and adapt all components through the Spring Framework. This allows building very specific PEPs for all kind of Java applications.

A set of Java annotations enables the user to choose which methods should be protected by the PEP and what information for a request are collected. This information, acquired to create decision requests, is cached to optimize performance. Calls to protected methods are intercepted with Spring AOP. To generate decision requests it is possible to integrate the HERAS^{AF} XACML 2.0 implementation. Finally, authorization decisions returned by a PDP are handled by a default decision handler.

HERAS^{AF} PEP-PDP Communication

The standardized communication between the PEP and PDP is realized with the Spring Web Services Framework. To be standard-compliant, the XACML requests/responses are wrapped into the SAML 2.0 protocol and SOAP envelopes. The Web Service client is implemented as a PEP component and therefore can be plugged into the PEP to evaluate decision requests by a remote PDP.

Personal Experience

- Working with an agile development process.
- Writing a technical report and multiple guides in English.
- Teamwork
- New experiences in software engineering and Java 5.
- New know-how in modern technologies like the Spring Framework, JAXB, TestNG, Maven 2.
- Implementation of complex standards (XACML over SAML over SOAP).

1.4 Outlook

General

Open issues, problems and solution descriptions can be found in part 3 (Technical Report) in chapter 5.2.

2 Aims and objectives

Intention In this part of the documentation the aims and objectives of this bachelor thesis are described.

2.1 Goals of this thesis

Context In the fall semester 2007, the HERAS^{AF} XACML project realized a JAXB implementation of the OASIS XACML 2.0 standard. This implementation is the basis of the development of a PDP Web Service and a PEP.

Goals The goal of this thesis is the implementation of the HERAS^{AF} PEP and the Web Service components (Endpoint & Client) for the communication between PEPs and PDPs. This contains the following points:

HERAS^{AF} PEP

- Analysis of the old HERAS^{AF} PEP implementation.
- Analysis, design, implementation and testing of the PEP
- Creation of a powerful, well documented API to build PEPs.
- Method interceptor with Spring AOP
- Caching strategies
- Plugability for different XACML implementations, interceptor technologies and PDPs.
- Modular and configurable components

HERAS^{AF} PEP-PDP Communication

- Analysis of the Spring Web Service Framework
- Analysis, design, implementation and testing of the PEP-PDP Communication
- Fast and reliable Web Service components (WS-Endpoint for the PDP and a WS-Client for the PEP)
- Standard-compliant transmission of XACML over SAML over SOAP
- Modular and configurable components

Documentation in English

- Bachelor Thesis
- PEP User's and Developer's Guide
- PEP-PDP Communication User's and Developer's Guide

2.2 Appointments

<i>18.02.2008</i>	Start of the bachelor thesis
<i>28.05.2008</i>	Abstract (draft) for examiner
<i>04.06.2008</i>	Abstract (final) for Claudia Furrer
<i>06.06.2008</i>	End of bachelor thesis
<i>23/24.08.2008</i>	Presentations of all bachelor theses

3 HERAS^{AF}

- Intention* In this part of the documentation background information and the structure of HERAS^{AF} will be described.
- Starting with general information about the background, motivation and intentions of HERAS^{AF}. Then there will be sections about application security, enterprise suitability and the declarative access control policy language in XML called XACML. Finally, there is a chapter about the logic of HERAS^{AF}, where each component and the communication between them are described.
- 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 [DOH07DADev].
- Motivation* Today, most applications have their own way to handle access control. This makes it hard for companies to enforce general security policies among all their applications and leads to additional costs, security risks and inconsistencies.
- HERAS^{AF} takes this opportunity and provides an Open Source architecture framework for the security aspect of enterprise applications based on the OASIS XACML 2.0 standard.
- Goals* HERAS^{AF} should gain ground in the Open Source market with providing a solution for the following goals.
- Holistic approach**
- HERAS^{AF} supports the whole authorization process
 - In a technical view this means that any kind of access in distributed agents (PEPs) on secured resources are interrupted and redirected to a PDP, where the evaluation based on policies is made. Whenever the PDP responds with a positive decision, the access on the protected resource will be granted.
 - HERAS^{AF} provides the possibilities for the definition and administration of policies without the need of technical knowledge.
- Enterprise Suitability**
- Integration of HERAS^{AF} in a corporation should be easy and should bring about only a few changes in the current infrastructure.

Authorization solutions used by the corporation are still able to be used or integrated with HERAS^{AF}.

- HERAS^{AF} is designed in terms of expandability and adaptability. In consistent use of the Spring Framework ApplicationContext to wire dependencies it allows the exchangeability of components.
- The API of HERAS^{AF} can be used to integrate company-specific parts, which can be hooked up into extension points.
- HERAS^{AF} uses established standards. For this reason the framework is broad-based and open for future extensions. This increases the interoperability and helps the integration of HERAS^{AF} in to an existing and future infrastructure.
- The PAP offers a separation of business concerns with its business views and language-constructs so the policy-administrator doesn't need to know anything about an URL or a class method. Templates provide the functionality which allows an administrator to create technical policies in a corporate-syntax (technical view). Whereas as the responsible person for authorization can define new policies based on the templates. He is able to define these policies in businesslike language (business view) where he doesn't need any technical knowledge.

3.1 Extensible Access Control Markup Language - XACML

General

XACML [XACML] is a standard of the organization OASIS. It defines a ubiquitous description of a policy-based access control language based on XML. XACML defines the syntax of a description language for:

- policies
- semantics for handling policies
- and request/response handling between PEP and PDP (communication)

XACML describes a solid base on which solution can be build on.

Additionally many other standards can be included into XACML:

- Security Assertion Markup Language (SAML) is an XML-based standard for exchanging authentication and authorization data between security domains, that is, between an *identity provider* (a producer of assertions) and a *service provider* (a consumer of assertions). [wikiSAML]
- Web Service Security (WSS) is a standard for securing integrity and confidentiality of messages in combination of Web Services.
- SPML (Service Provisioning Markup Language) is an XML-based framework, being developed by OASIS, for exchanging user, resource and service provisioning information between cooperating organizations. [wikiSPML]
- The Digital Signature Algorithm (DSA) is a United States Federal Government standard or FIPS for digital signatures. [wikiDSS]
- Public key infrastructure (PKI) is an arrangement that binds public keys with respective user identities by means of a certificate authority (CA). [wikiPKI]

Background

XACML 2.0 was ratified in february 2005 from the OASIS-Consortium. Members of the OASI-Group like Entrust, IBM, Hewlett-Packard, Sun Microsystems and others helped on the specification of the XACML.

Sun Microsystems, headed by Seth Proctor, did an Open Source reference-implementation of XACML 1.0 and the development of the XACML 2.0 implementation is stopped. The latest news entry about the progress was added on 21 June 2006. [SunXACML]

By the end of 2007 a team of HERAS^{AF} engineers (Sacha Dolski, Stefan Oberholzer and Florian Huonder) implemented the XACML 2.0 standard under the name HERAS^{AF} XACML. [DOH07DADev]

The reputation of XACML is growing continuous. In a press release of OASIS [PressInterop] was announced that eight new well-known companies such as BEA Systems, IBM, Oracle, Red Hat and some more will work together to demonstrate the interoperability of XACML.

JBoss Security is providing a standards based, robust Policy Infrastructure library based on the Oasis XACML 2.0 standard. It defines an API to read one or more PolicySets via simple configuration files, defines an API to parse XACML request files or streams and provides a JAXB 2.0 compatible object model that can be used to construct policies and requests in XACML. [JBossXACML]

*Advantage /
Disadvantages*

Sun Microsystems describes the most important advantages as followed. [SunXACML05]

XACML is a standard

Because XACML is a ratified, standard language, developers who use XACML can take advantage of the collective experience of a large community of experts and users, so they don't need to roll their own system each time, and they don't need to think about all the tricky issues involved in designing a new language. Moreover, as XACML becomes more widely deployed, it will be easier to interoperate with other applications using the same standard language.

XACML is generic

This means that rather than having to provide access control for each particular environment or each specific kind of resource separately, developers can use XACML in any environment. One policy can be written that can then be used by many different kinds of applications, and when one common language is used, policy management becomes much easier.

XACML is distributed

This means that a policy can be written in a way that in turn refers to other policies kept in arbitrary locations. The result is that rather than having to manage a single monolithic policy, different people or groups can manage separate sub-policies as appropriate, and XACML knows how to correctly combine the results from these different policies into one decision.

XACML is powerful

While there are many ways the base language can be extended, many environments will not need to do so. The standard language already supports a wide variety of data types, functions, and rules about combining the results of different policies. In addition to this, there are already standards groups working on extensions and profiles that will hook XACML into other standards such as SAML and XML Digital Signature, which will increase the number of ways that XACML can be used.

A disadvantage of a flexible and powerful language like XACML is the amount of additional metadata which is used for a message. For example if XACML data is

sent via SAML and SOAP the message containing the reference data is comparatively huge. [Egg06]

Model

The following logic components are described by the XACML 2.0 specification.

Policy Administration Point (PAP)

The PAP is an administration component. The PAP allows administrating policies as well as un-/deploying these on a PDP.

Policy Enforcement Point (PEP)

The PEP is an enforcement component (performer). The PEP interrupts accesses on secured resources and creates an authorization-request which is forwarded to a Context Handler. Depending on the authorization-response (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. The PDP 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. The PIP provides additional information about subjects, resources, environment or actions. This information can be referenced by policies.

Context Handler (CH)

The context handler has task to fulfill. The handler is responsible to translate authorization-request not yet expressed in XACML into an equal authorization-request in XACML. As well as to provide additional information for a PDP.

Data flow

The model operates by the following steps. See Figure 1.

1. PAPs write policies and policy sets and make them available to the PDP. These policies or policy sets represent the complete policy for a specified target.
2. The access requester sends a request for access to the PEP.
3. The PEP sends the request for access to the context handler in its native request format, optionally including attributes of the subjects, resource, action and environment.
4. The context handler constructs an XACML request context and sends it to the PDP.
5. The PDP requests any additional subject, resource, action and environment attributes from the context handler.
6. The context handler requests the attributes from a PIP.
7. The PIP obtains the requested attributes.
8. The PIP returns the requested attributes to the context handler.
9. Optionally, the context handler includes the resource in the context.
10. The context handler sends the requested attributes and (optionally) the resource to the PDP. The PDP evaluates the policy.
11. The PDP returns the response context (including the authorization decision) to the context handler.
12. The context handler translates the response context to the native response format of the PEP. The context handler returns the response to the PEP.
13. The PEP fulfills the obligations.

14. (Not shown) If access is permitted, then the PEP permits access to the resource; otherwise, it denies access.

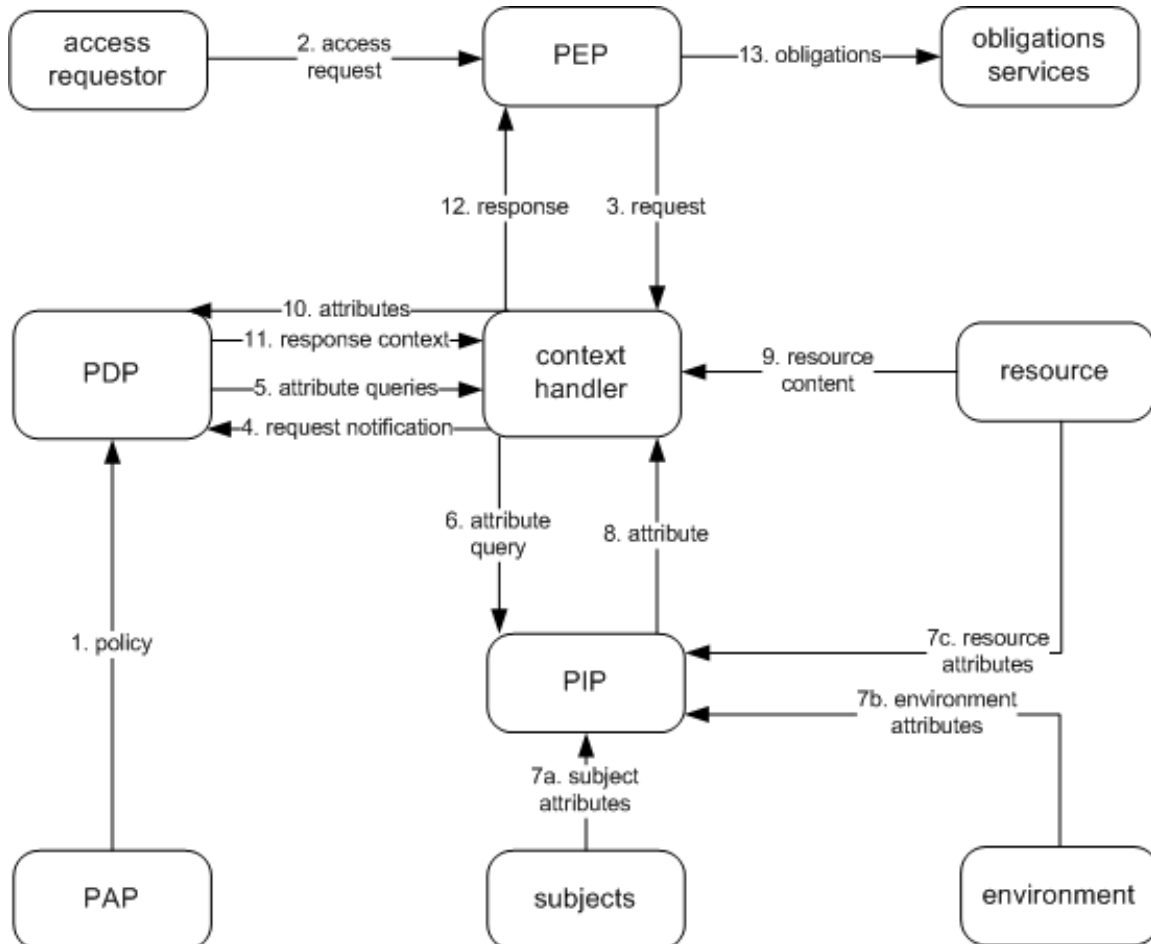


Figure 1: Data flow diagram model of the XACML 2.0 specification

3.2 HERAS^{AF} Logic

Intention This chapter describes the logical structure of HERAS^{AF} with all its components and how these components communicate with each other.

Overview HERAS^{AF} implements all components needed for a comprehensive authorization process. Each component can be deployed and operated independently from each other by the use of common standard technologies.

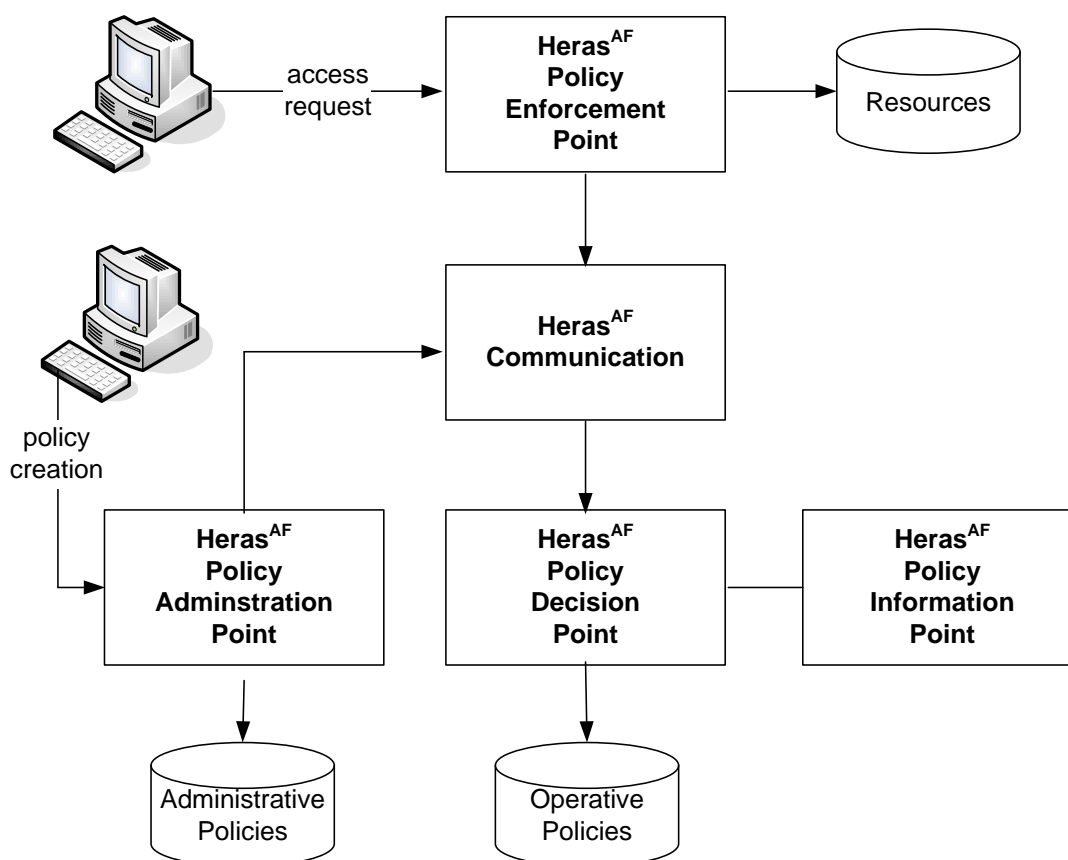


Figure 2: HERAS^{AF} components

Special Characteristic

Compared to the XACML 2.0 specification, HERAS^{AF} has two major differences: There is no ContextHandler module and only one resource per decision request can be handled.

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

In the current state of HERAS^{AF}, handling of multiple resources as specified by XACML 2.0 is not yet implemented. The HERAS^{AF} PDP can only handle one resource per decision request from a PEP.

3.2.1 HERAS^{AF} Policy Enforcement Point

General The HERAS^{AF} PEP provides the structures and mechanisms to build agents that perform access control, by making decision requests and enforcing authorization decisions in a specific application.

These application-specific agents can intercept and control access in many different ways, depending on how the structure of HERAS^{AF} PEP is adapted.

Adaptation Many logical aspects of HERAS^{AF} PEP can be adapted to fit the application in need of protection. These aspects are mainly the generation of decision requests, the communication to the PDP and the handling of authorization decisions.

Additionally, agents can use different approaches to intercept requests to resources and enforce security within the application. Common examples are filter technologies or aspect-oriented programming.

3.2.2 HERAS^{AF} Policy Decision Point

General The HERAS^{AF} PDP implements the complete logic of access decision making. This includes locating applicable policies, evaluating decision requests and combining the evaluation results.

All requirements for a PDP are described by the XACML 2.0 specification and the paper [Graf06].

Performance Because the main focus of the PDP is performance, many performance-enhancing mechanisms, like configurable indexing or fast comparing-algorithms, have been included.

Additionally, the PDP only accesses the data storage on initialization. After that all policies are held in the RAM, which leads to an acceleration of the decision making process. [DOH07DA]

Distributed PDP A PDP is normally used as a central entity serving multiple PEPs. On a physical layer the HERAS^{AF} PDP can be divided onto different virtual machines and computers, thus allowing applying load sharing or rules for failure. [DOH07DA]

3.2.3 HERAS^{AF} PEP-PDP Communication

General The communication between a HERAS^{AF} PEP and HERAS^{AF} PDP is based on Web Services. To allow that HERAS^{AF} components to be used with components of other vendors, a standard-compliant communication is essential.

Standard compliance The standard-compliance is assured by transmitting SAML [SAML] inside a SOAP envelope [SOAP]. SAML defines a profile [SAMLXACML] which allows encapsulating the XACML requests/responses.

Web Service The WSDL describes the Web Service contract. It defines a simple method to evaluate an XACML request and limits the XML element types that can be sent inside the SOAP Envelope.

3.2.4 HERAS^{AF} Policy Administration Point

General The HERAS^{AF} PAP allows creating and managing security policies and is responsible to inform the PDPs about these policies.

The PAP provides a user-friendly interface to support the creation of even very complex policies for different types of users. Users with no technical background can easily manage policies on a business view using templates, created by engineers. Engineers on the other hand can use a technical view to create templates, which allows them to consider all the technical aspects of security policies.

More information about this module can be found in the [NZ08PAP] document.

3.2.5 HERAS^{AF} Policy Information Point

General If a request doesn't contain all required information, the PDP can contact a specific PIP to gather this information.

The HERAS^{AF} PIP provides the logic and structure to build a specific adapter or connector between the authorization solution and the existing infrastructure. Thus it is possible to add data from legacy systems into the decision making. [DOH07DA]

The requirements to the HERAS^{AF} PIP and how a PIP is integrated into HERAS^{AF} has not yet been determined.

4 HERAS^{AF} Policy Enforcement Point (PEP)

Intention This chapter describes the detailed structure of the HERAS^{AF} PEP and all of its adaption points.

Overview The HERAS^{AF} PEP is a framework which provides the pluggability to build agents that perform access control, by making decision requests and enforcing authorization decisions in a specific application.

These application-specific agents can intercept and control access in many different ways, depending on how the structure of HERAS^{AF} PEP is adapted.

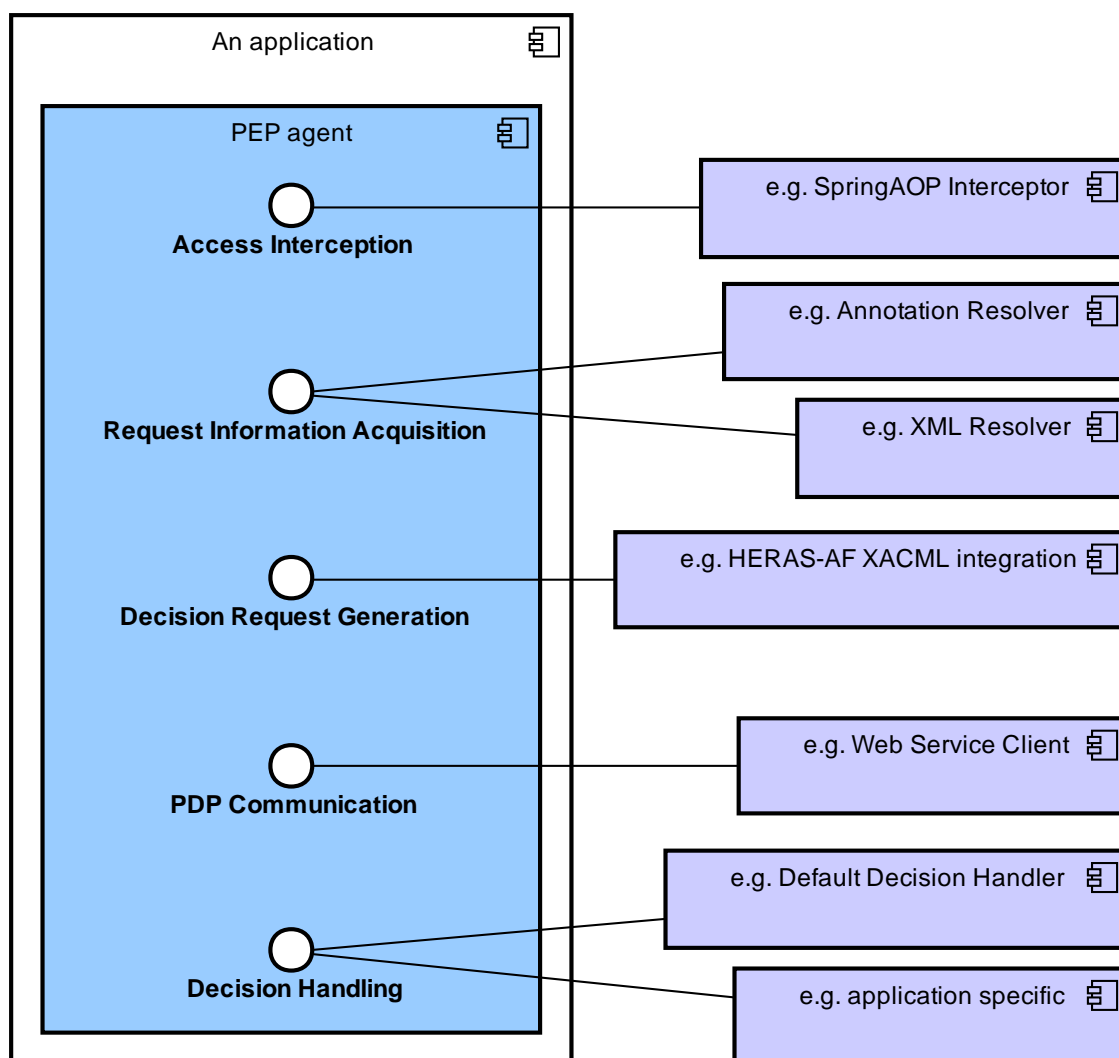


Figure 3: Logical components and adaption points of a PEP

Adaptation Many logical aspects of HERAS^{AF} PEP can be adapted to fit the application in need of protection. These aspects are mainly:

- the interception of access
- the acquisition of request information
- the generation of decision requests
- the communication to the PDP
- the handling of authorization decisions

4.1 Access Interception

<i>Overview</i>	HERAS ^{AF} PEP agents can use different approaches to intercept requests to resources and enforce security within the application. Common examples are filter technologies or aspect-oriented approaches.
<i>Filter technologies</i>	<p>Filter technologies, like Servlet filters or Apache modules, can often be found in web environments where web applications are executed in an application container (e.g. Java EE or Dependency Injection container).</p> <p>User requests are filtered by the requested resource and delegated to a filter. The filter can then add additional logic and initiate the authorization process.</p> <p>After the decision request has been evaluated the filter can continue its work, depending on the returned authorization decision.</p>
<i>Aspect-oriented programming</i>	<p>Aspect-oriented programming (AOP) is a programming paradigm that attempts to aid programmers in the separation of concerns. To accomplish this, additional code in separate files is automatically added to the program code on runtime.</p> <p>With this method an agent could be integrated into an existing application without the need of recompiling it. This allows programmers to ignore the aspect of security in most parts of their application, because they know that an access controlling agent can be integrated later on. This approach of software security development is known as security last.</p> <p>In addition, using AOP allows a more explicit authorization logic that can control access to an object, its methods or even fields.</p>
<i>Specific programming</i>	<p>The enforcing of security can also be done directly in the application without the use of an interceptor technology.</p> <p>This allows developing applications like before by using the HERAS^{AF} PEP agent manually whenever security concerns need to be addressed.</p>

4.2 Request Information Acquisition

<i>Overview</i>	Information about context of a decision request (subjects, resources, the action and the environment) build the basis for the generation of a decision request. HERAS ^{AF} PEP agents can use different so called Resolvers to acquire this information from any kind of dynamic or static data source. Common examples are Java annotations, XML files or databases. All the data sources are in connection with required runtime information.
<i>Java annotations</i>	The HERAS ^{AF} PEP provides a number of Java annotations [wikiJavaAnnotation] that can be used to annotated Java classes with request information. By using special Resolvers for annotations the HERAS ^{AF} PEP is then able to acquire the request information from annotations attached to the invoked method, its parameters and the class it is part of.
<i>Other data sources</i>	Any other request information source, like an XML file or a database, can be supported by the HERAS ^{AF} PEP by implementing a specific Resolver. This Resolver has to acquire the request information from the data source and parse it into a data structure known by the HERAS ^{AF} PEP.

4.3 Decision Request Generation

<i>Overview</i>	PEP agents are free to choose how standard-compliant XACML decision requests are generated. In most cases the generation is done by using an existing XACML implementation like HERAS ^{AF} XACML.
<i>Using the HERAS^{AF} XACML integration</i>	In the integration layer for HERAS ^{AF} XACML so called Transformers transform the request information of the HERAS ^{AF} PEP into the JAXB data structure [wikiJAXB] of HERAS ^{AF} XACML. Once the transformation is completed JAXB marshallers are used to generate the actual XACML decision request in XML form. Parts of this functionality are already supported by HERAS ^{AF} XACML.
<i>Other XACML implementations</i>	Any other XACML implementations, compliant with XACML 2.0, can be supported by the HERAS ^{AF} PEP by implementing an integration layer that implements interfaces used by the PEP and uses a specific XACML implementation.

4.4 PDP Communication

<i>Overview</i>	To receive a decision on a decision request, HERAS ^{AF} PEP agents can use any strategy to locate and communicate with a PDP.
<i>Locating a PDP</i>	Locating a PDP is supported by the HERAS ^{AF} PEP by implementing a so called PDP Locator that locates and returns the PDP to evaluate decision requests.

The HERAS^{AF} PEP asks the PDP Locator for a PDP and the PDP Locator returns a PDP. With this approach the PEP does not need to know what kind of PDP he uses or where the PDP is located.

This allows HERAS^{AF} PEP agents to use complex strategies that introduce load balancing between multiple PDPs or return a backup PDP, if a primary PDP is not responding.

Communicating with a PDP

Communicating with a PDP is supported by the HERAS^{AF} PEP by implementing a HERAS^{AF} PEP specific PDP interface. This allows HERAS^{AF} PEP agents to communicate with local as well as remote PDPs.

For example an agent could use the HERAS^{AF} PEP Web Service client to communicate through the Web Service interface of a remote PDP.

4.5 Decision Handling

Overview

After a decision request has been evaluated by a PDP the HERAS^{AF} PEP delegates the handling of the returned authorization decision to a so called Decision Handler. The PEP provides a default decision handler but also supports any custom decision handlers.

Default decision handler

HERAS^{AF} PEP agents can use the default decision handler that ensures a standard-complaint handling of the XACML response. The behavior of the HERAS^{AF} PEP is affected by the XACML response elements obligations, decision and status code.

The handling of all obligations, the correct interpretation of the decision and the proper continuation or interruption of application flow is done by a decision handler. The following diagram shows the flow of the default decision handler.

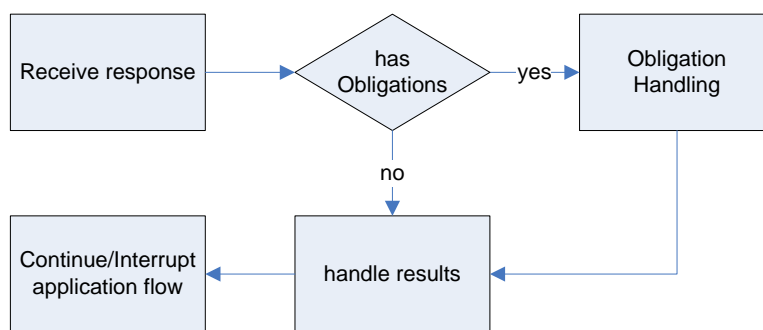


Figure 4: High level decision handling diagram

Application-specific handlers

It is also possible to implement and use custom decision handlers that have an application-specific behavior that might even violate the XACML specification.

For example a custom decision handler could abandon the handling of any obligations in the XACML response and just ignore it. Even if a PEP must fulfill any obligations before it continues.

5 HERAS^{AF} PEP-PDP Communication

Intention This chapter describes the detailed structure of the HERAS^{AF} PEP-PDP Communication and all of its adaption points.

Overview The HERAS^{AF} PEP-PDP Communication is a framework that provides the ability to build Web Service Server and Client components based on the Spring Web Service framework [SpringWS].

Web Service Server Web Service server interfaces allow a HERAS^{AF} PDP to receive decision requests from PEPs.

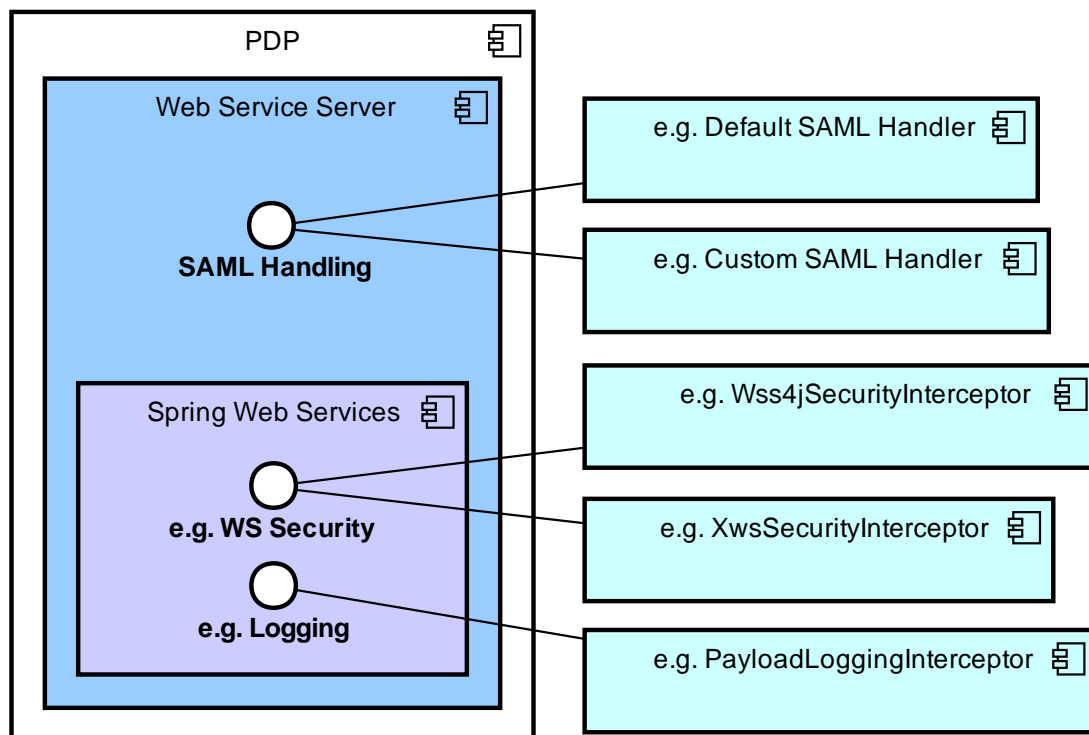


Figure 5: Logical components and adaption points of a PDP WS server

Web Service Client The Web Service client can be integrated into a HERAS^{AF} PEP and allows the evaluation of decision requests by a remote PDP.

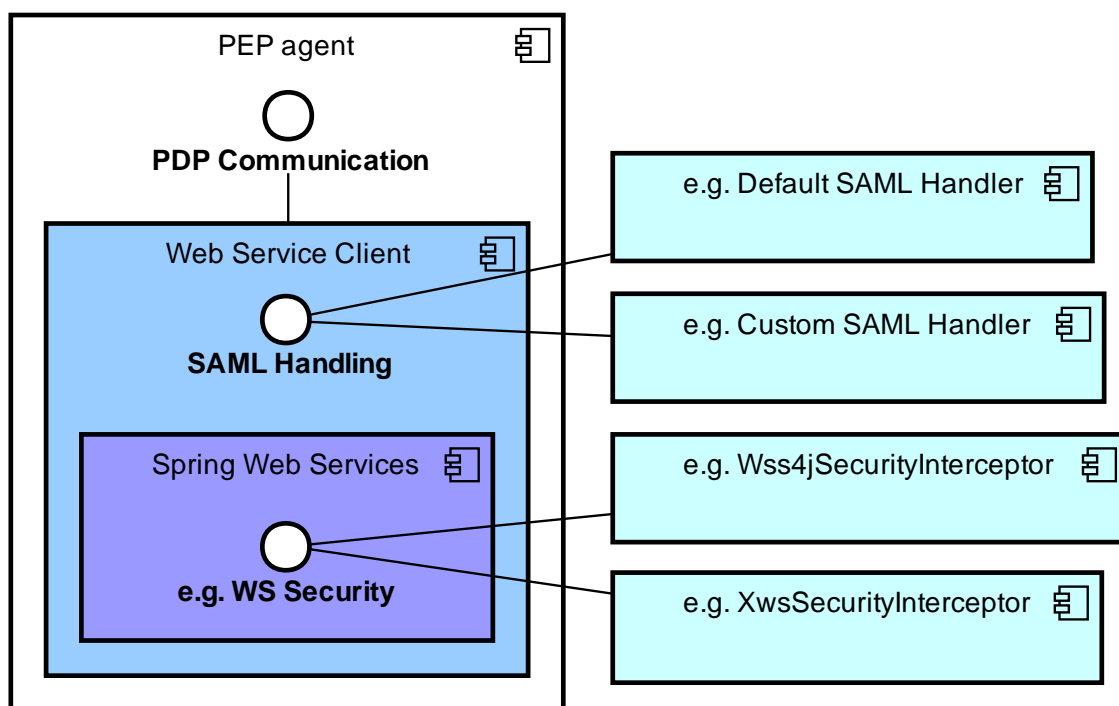


Figure 6: Logical components and adaption points of a PEP WS client

Adaption

These Web Service clients and servers are highly adapt- and extendible to meet various security or other requirements for a specific environment or application.

Many logical aspects of the HERAS^{AF} PEP-PDP Communication can be adapted to meet specific requirements of the system. These aspects are mainly:

- the handling of SAML (requests/responses)
- the handling of the Web Service
- security aspects

5.1 Standard-compliance

Overview

It is very important that the communication between a PEP and PDP is compliant with SAML 2.0 [SAML] and the SAML 2.0 Profile of XACML [SAMLXACML].

HERAS^{AF} components are open for other implementations only if all the components are standard-compliant.

The interfaces to a PDP or the client of PEP are typical points where, for example, other implementations of a PDP server or PEP client from different vendors could be used.

HTTP Message

The communication between PEPs and PDPs works via HTTP, SOAP, SAML, SAML Profile of XACML and XACML. Figure 7 shows the encapsulations of

such a message.

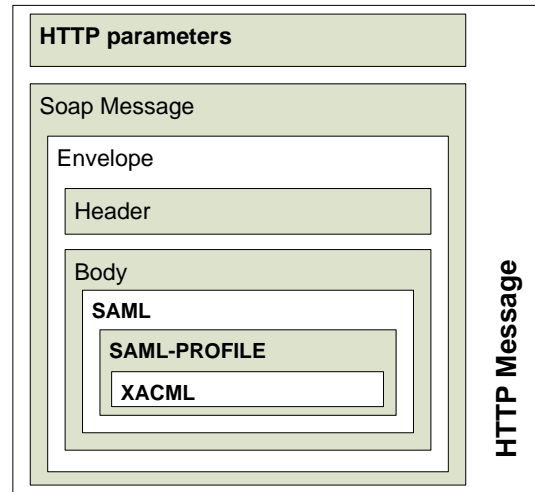


Figure 7: HTTP, SOAP, SAML, SAML Profile of XACML, XACML message

5.2 SAML Handling

Overview

One XML standard that provides the assertion and protocol mechanism needed by XACML is the SAML 2.0 OASIS standard extended by the SAML 2.0 Profile of XACML.

The HERAS^{AF} PEP-PDP Communication uses this standard to encapsulate XACML data for the transmission between PEPs and PDPs. All aspects of SAML are handled by a so called SAML Handler. Both Web Service server and client provide a default SAML handler for a standard-compliant handling of SAML aspects.

Mandatory and optional attributes

Multiple SAML elements define required attributes. The default SAML handler ensures that these mandatory attributes are always set, to be compliant with SAML 2.0.

Other optional attributes in SAML elements allow the usage of additional SAML features, like encryption or signatures. The default SAML handler provides the extensibility to allow easy implementations of standard-compliant handlers that can use these additional SAML features.

SAML status code

The SAML 2.0 Profile of XACML defines that the status code of a SAML response depends on the status code of the XACML response encapsulated in SAML.

The default SAML handler of HERAS^{AF} PEP-PDP Communication fulfills this requirement.

5.3 Web Service Handling

Overview The HERAS^{AF} PEP-PDP Communication uses the Spring Web Services framework to build Web Services for the communication between a PEP and a PDP.

Most aspects of these Web Services and the transmission of SOAP messages over HTTP are handled by components of Spring WS. But the transmission of XACML over SAML over SOAP over HTTP, requires the additional handling of some special requirements.

Special requirements Some special requirements for the transmission of SAML data over SOAP/HTTP are specified by the SAML Bindings 2.0 [SAMLBind]:

- HTTP header fields defining the specified caching behavior.
- HTTP header field `SOAPAction` with a specified value. (optional)

5.4 Security aspects

Overview The communication between two components of HERAS^{AF} must be secured, especially if the communication is over a network.

Figure 8 shows the possible attacking points. The four security values are shortly explained:

- **Integrity** (forge, modify, spoof, fake): attack to change or forge the information residing on or passing through a system
- **Confidentiality** (read, sniff, eavesdrop): attack to read (or steal) secret information from a system
- **Authenticity** (replay, hijack, masquerade): attack to forge the originator of the message (Identity Theft)
- **Availability** (Flooding, denial of service): attack that directly inhibits a user (human or machine) from accessing a particular system resource

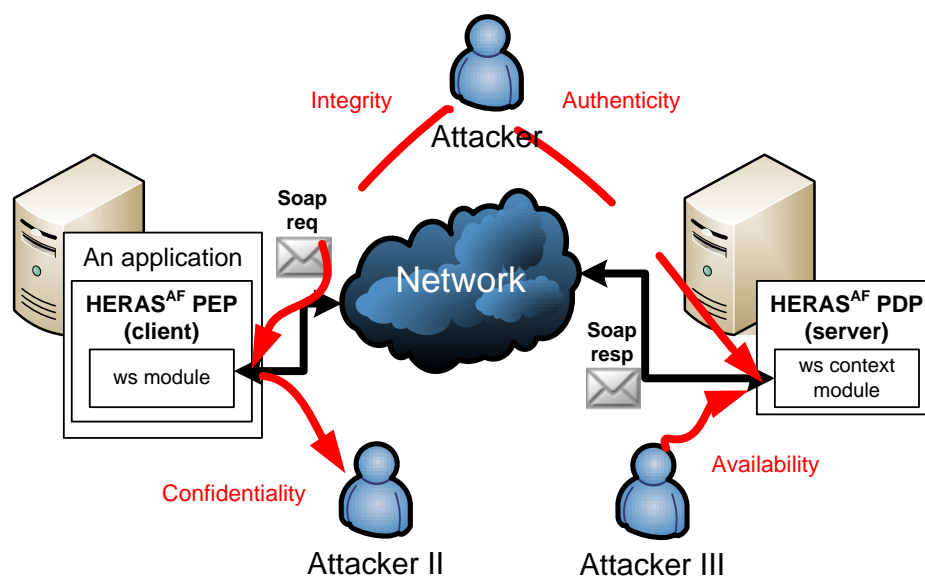


Figure 8: Attacking points

There are basically two approaches to add security to a Web Service. Provide Web Service Security and/or handle security issues on the transport layer (e.g. TLS/SSL).

Web Service Security

Web Service Security is a communication protocol to apply security to Web Services. For example it describes how attach signatures and encryption headers to SOAP messages.

WSS4J [WSS4J] is a Java framework which can handle Web Service Security.

Security on transport layer

The security on transport layer is based on Secure Socket Layer (SSL) or Transport Layer Security (TLS).

SSL and TLS provide Authentication, Privacy Protection as well as decryption-tokes to secure HTTP-connections.

Part II: Project Management

1 Project Planning

1.1 Introduction

Intention This planning is a guide for the project HERAS^{AF} PEP and PDP Web Service Integration. It contains a project schedule, quality/risk management, used standards and a final project report.

1.2 Overview

Goals The goals of the project are defined in the first part **Aims and objective**. See chapter 2 of part 1.

<i>Organizational structure</i>	Daniel Regli (RED)	Responsibilities
	Project manager	<ul style="list-style-type: none"> • Project controlling • Communication in team
	Programmer	<ul style="list-style-type: none"> • Design, Tests
		Email: daniel.regli@herasaf.org
	Yannick Winiger (WIY)	Responsibilities
	Lead programmer	<ul style="list-style-type: none"> • Analysis, Architecture • Coordination in refactoring
	Quality manager	<ul style="list-style-type: none"> • Code Quality • Documents Quality
		Email: yannick.winiger@herasaf.org
<i>External Contacts</i>	Wolfgang Giersche (GIW)	Contact
	Responsible tutor	Email: wolfgang.giersche@herasaf.org
	Responsible tutor for the bachelor thesis and contact person for architectural questions and other concerns.	
	Sacha Dolski(DOS)	Contact
	Expert	sd&m Schweiz AG software design & management World Trade Center Zürich Leutschenbachstrasse 95 8050 Zürich
		Email:

sacha.dolski@herasaf.org

External expert for this bachelor thesis. He also will be present for the oral examination at the end of the thesis.

Florian Huonder (HUF)

Contact

Coach

Email:
florian.huonder@herasaf.org

Coaches the project team and can be contacted for anything related to the project.

René Eggenschwiler (EGR)

Contact

Coach

Email:
rene.eggenschwiler@herasaf.org

Coaches the project team and can be contacted for anything related to the project.

Appointments

The following appointments are given from the HSR:

- Project start: 18.02.2008
- Abstract (draft) for examiner: 28.05.2008
- Abstract (final) for Claudia Furrer: 04.06.2008
- Project end: 06.06.2008 (12:00)

Meetings

During the project there will weekly meetings with the responsible tutor and both coaches.

At these weekly meetings the status of the project and current iteration will be presented and the next steps discussed.

1.3 Schedule

Iteration Planning

Iteration 1	Start	End
Setup of the project infrastructure, studies of new technology.	18.02.2008	22.02.2008
Iteration 2	Start	End
Integration of HERAS ^{AF} XACML 2.0 into the existing PEP.	23.02.2008	29.02.2008
Documentation of the project management.		



Iteration 3	Start	End
Analysis of the old PEP.	01.03.2008	07.03.2008
Design and implementation of the PEP.		
Iteration 4	Start	End
Design and implementation of the PEP.	08.03.2008	14.03.2008
Test of the PEP		
Iteration 5	Start	End
Documentation of the project and analysis.	15.03.2008	21.03.2008
Implementation of the PEP.		
Iteration 6	Start	End
Documentation of the analysis and design.	22.03.2008	28.03.2008
Implementation and test of the PEP.		
Iteration 7	Start	End
Implementation of the PEP.	29.03.2008	04.04.2008
Migration to Spring 2.5.		
Iteration 8	Start	End
Implementation of the PEP and Integration tests.	05.04.2008	11.04.2008
Iteration 9	Start	End
Finish Implementation of the PEP with documentation.	12.04.2008	18.04.2008
Extend test coverage.		
Iteration 10	Start	End
Studies of the old PDP Web Service.	19.04.2008	25.04.2008
Prototype with Spring WS. Familiarization with SOAP, SAML and the SAML profile for XACML.		
Iteration 11	Start	End
Finish the prototype so that SOAP/SAML/XACML request/response is sent standard-compliant.	26.04.2008	02.05.2008

Iteration 12	Start	End
Implementation of the PDP-PEP Web Service based on the prototype with Spring WS.	03.05.2008	09.05.2008
Iteration 13	Start	End
Finish the implementation of the PDP-PEP Web Service (with Exception / SAML handling). Extend test coverage	10.05.2008	16.05.2008
Iteration 14	Start	End
Code and JavaDoc clean up. Extend test coverage.	17.05.2008	23.05.2008
Iteration 15	Start	End
Completion of the documentation.	24.05.2008	30.05.2008
Iteration 16	Start	End
Completion of the documentation.	31.05.2008	06.06.2008

Milestones

M1: A fully functional PEP Implementation	Date
The HERAS ^{AF} PEP implementation is completed and tested. All the given requirements are implemented and work.	18.04.2008
M2: A standard-compliant PEP-PDP Communication implementation	Date
The HERAS ^{AF} PEP-PDP Communication implementation is completed and tested. All the given requirements are implemented and work.	14.05.2008
M3: Finish the bachelor thesis	Date
The bachelor thesis is documented and given to the experts.	06.06.2008

1.4 Risk management

Introduction

The risk management will be updated through the whole project and each risk reevaluated.

risk	consequence	actions	Invest in action in work-hours	Maximum damage in work-hours	occurrence of event in percent	Emphasis of damage in work-hours	Priority (1-low, 2-medium, 3-high)
R01: complexity of XACML standard	Time schedule danger in delay and investment of more time	Asking/communicate with tutor and coaches when questions arise, weekly meetings with status reports	10	15	10 %	1.5	3
R02: Bad medical conditions of team members	Delay of project	-	0	16	15 %	2.5	1
R03: Loss of SVN server infrastructure	Current status of work will be lost	Local copies or / and re-programming the code	10	20	20 %	4	2
R04: Team-problems / personal problems	Less useful communication, inefficient work-hours of team member	Separating work and leisure time, respect the each individual	8	40	5 %	2	1
R05: Complexity of new technologies	Time schedule danger in delay and investment of more time	reading literature and coding concepts of earlier projects	25	35	15 %	5.5	2
Total work hours contained in work package			53				
Total accruals						15.5	

1.5 Quality management

Documentation All project relevant information will be written in down in any of the following documents:

- HERAS-AF PEP-PDP - Bachelor Thesis
- HERAS-AF PEP-PDP - Minutes of meeting
- HERAS-AF PEP - Developers Guide
- HERAS-AF PEP - User Guide
- HERAS-AF PEP-PDP - Developers Guide
- HERAS-AF PEP-PDP - User Guide

All the documents will be updated during the project duration.
The language used in all documents is English.

Tests Complete tests for each module and feature will be written. Mock objects will be used to test each part individually.

Reviews / Communication Every document and architectural decision is made by the whole team. If the team cannot agree on a point as whole team, it will be discussed with the tutor or coach.

All the documents will be read from both team members to be able to find all the obscurities and mistakes.

The document HERAS-AF PEP-PDP - Minutes of meeting will contain all decisions and meeting topics.

Version System Project and source code files will be managed by a version control system (SVN). This will help that every team member has the same view to the source code and project files.

Build Automation Maven2 will be used to build HERAS^{AF} PEP and HERAS^{AF} PEP-PDP Communication modules. The HERAS^{AF} PEP will contain a root module which will be used to build the all modules of the HERAS^{AF} PEP.

1.6 Technologies and Standards

1.6.1 Naming Conventions

Artifact Filenames The filenames are build from the following structure:

HERAS-AF [PEP|PEP-PDP] - <artifact-name>

Examples:

- HERAS-AF PEP-PDP - Bachelor Thesis.doc

- HERAS-AF PEP - Developer Guide.doc
- HERAS-AF PDP - User Guide.doc

1.6.2 Technologies

<i>Spring Security 2.x.x</i>	Spring Security is a powerful, flexible security solution for enterprise software, with a particular emphasis on applications that use Spring. Using Spring Security provides your applications with comprehensive authentication, authorization, instance-based access control, channel security and human user detection capabilities. [SpringSecurity].
<i>CgLib 2.x.x</i>	Cglib is a powerful, high performance and quality Code Generation Library, It is used to extend JAVA classes and implements interfaces at runtime. [CgLib]
<i>Java 1.5 annotations</i>	<p>An annotation, in the Java computer programming language, is a special form of syntactic metadata that can be added Java source code. Classes, methods, variables, parameters and packages may be annotated.</p> <p>Unlike JavaDoc tags, Java annotations are reflective in that they are embedded in class files generated by the compiler and may be retained by the Java Virtual Machine to be made retrievable at run-time.[wikiJavaAnnotation]</p>
<i>JAXB 2.x</i>	Java Architecture for XML Binding (JAXB) allows Java developers to map Java classes to XML representations. JAXB provides two main features: the ability to marshal Java objects into XML and to unmarshal XML back into Java objects. In other words, JAXB allows storing and retrieving data in memory in any XML format, without the need to implement a specific set of XML loading and saving routines for the program's class structure. [wikiJAXB]
<i>Maven 2.x</i>	Maven is a software project management and comprehension tool. Based on the concept of a project object model (POM), Maven can manage a project's build, reporting and documentation from a central piece of information. [Maven]
<i>Spring 2.x.x</i>	<p>The Spring Framework (or Spring for short) is an open source application framework for the Java platform.</p> <p>Although the Spring Framework does not enforce any specific programming model, it has become popular in the Java community as an alternative, replacement, or even addition to the Enterprise Java Bean model.</p> <p>The Spring Framework can be considered as a collection of smaller frameworks. [Spring]</p>
<i>Spring AOP 2.x.x</i>	<p>The Spring Framework can be considered as a collection of smaller frameworks. Most of these frameworks are designed to work independently of each other yet provide better functionalities when used together.</p> <p>These frameworks are divided along the building blocks of typical complex applications. One of them is Spring AOP.</p>

Aspect-oriented programming framework: working with functionalities which cannot be implemented with Java's object-oriented programming capabilities without making sacrifices. [wikiSpring] [SpringAOP]

Spring WS 1.5.x

The Spring Framework can be considered as a collection of smaller frameworks. Most of these frameworks are designed to work independently of each other yet provide better functionalities when used together.

These frameworks are divided along the building blocks of typical complex applications. One of them is Spring WS.

Spring Web Services (Spring-WS) is a product of the Spring community focused on creating document-driven Web services. Spring Web Services aims to facilitate contract-first SOAP service development, allowing for the creation of flexible web services using one of the many ways to manipulate XML payloads. The product is based on Spring itself, which means you can use the Spring concepts such as dependency injection as an integral part of your Web service. [SpringWS]

TestNG 5.x

TestNG is a testing framework for the Java programming language inspired by JUnit and NUnit but introducing some new functionality that purport to make it more powerful and easier to use.

TestNG is designed to cover all categories of tests, including unit, functional, and integration tests. [wikiTestNG]

XMLUnit 1.x

XMLUnit is an Open Source project which enables assertions to be made about the content and structure of XML. It allows distinguishing between valid XML with correct content and valid XML with incorrect content and doesn't just validate with a DTD or a schema. [XMLUnit]

1.6.3 Standards

SAML 2.0

Security Assertion Markup Language (SAML) is an XML standard for exchanging authentication and authorization data between security domains, that is, between an identity provider (a producer of assertions) and a service provider (a consumer of assertions). SAML is a product of the OASIS Security Services Technical Committee. [wikiSAML]

SOAP 1.1

SOAP is a protocol for exchanging XML-based messages over computer networks, normally using HTTP/HTTPS. SOAP forms the foundation layer of the web services protocol stack providing a basic messaging framework upon which abstract layers can be built.

There are several different types of messaging patterns in SOAP, but by far the most common is the Remote Procedure Call (RPC) pattern, in which one network node (the client) sends a request message to another node (the server) and the server immediately sends a response message to the client. SOAP is the successor of XML-RPC, though it borrows its transport and interaction neutrality and the envelope/header/body from elsewhere, probably from WDDX. [wikiSOAP]

XACML 2.0

XACML is an OASIS standard that describes both a policy language and an access control decision request/response language (both encoded 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]

1.7 Project management final report

Schedule

The new goals, which were introduced during the bachelor thesis, were all achieved successfully.

The team worked more than scheduled through the whole bachelor thesis. This was because the goals were arranged weekly. Unforeseen problems occurred and bigger research work had to be done with which the goals had to be rescheduled.

Cost estimate

The difference between scheduled and done work is approximately 2.5 week for each team member. The discrepancy results because the team worked from the beginning till the end more than it was requested.

The additional work done was on voluntary basis. The team wanted to achieve the goals and didn't want to let the HERAS^{AF} team down.

2 Evaluation

2.1 Introduction

Intention The evaluation was made in a late project state. The architecture and code quality was made after the code freeze. The time evaluation was made in the last week of the project, for the rest of the week a time assumption was used.

The evaluation shows the end state of the project. It gives the project team an overview and a feedback about their work.

2.2 Architecture and code quality

General The architecture and code quality evaluation takes the productive and test code of the modules into account.

<i>PEP Core module</i>	Metrics	
	Numbers of packages	12
	Number of interfaces	11
	Number of classes	40
	Number of methods	124
	Number of productive code lines	1082
	Number of test code lines	399
<i>PEP Herasafxacml module</i>	Metrics	
	Numbers of packages	3
	Number of interfaces	0
	Number of classes	18
	Number of methods	41
	Number of productive code lines	619
	Number of test code lines	325
<i>PEP Spring AOP module</i>	Metrics	
	Numbers of packages	2
	Number of interfaces	0
	Number of classes	13
	Number of methods	26
	Number of productive code lines	270
	Number of test code lines	0

*PEP Utilities
module*

Metrics

Numbers of packages	4
Number of interfaces	7
Number of classes	17
Number of methods	78
Number of productive code lines	631
Number of test code lines	334

*PEP WS
module*

Metrics

Numbers of packages	5
Number of interfaces	2
Number of classes	10
Number of methods	35
Number of productive code lines	266
Number of test code lines	180

*PDP Context-
WS module*

Metrics

Numbers of packages	4
Number of interfaces	0
Number of classes	8
Number of methods	31
Number of productive code lines	218
Number of test code lines	54

SAML module

Metrics

Numbers of packages	13
Number of interfaces	0
Number of classes	106
Number of methods	647
Number of productive code lines	4833
Number of test code lines	140

*PDP Context-
WS Integration
Tests module*

Metrics

Numbers of packages	4
Number of interfaces	0
Number of classes	8
Number of methods	31
Number of productive code lines	0
Number of test code lines	272

*PEP Integration
Tests module*

Metrics

Numbers of packages	3
Number of interfaces	2
Number of classes	10
Number of methods	46
Number of productive code lines	0
Number of test code lines	792

2.3 Time evaluation

General

The duration of the project was 14 part-time weeks and 2 full-time weeks. During a part-time week 20 hours per person and full-time week is 40 hours per person was scheduled.

Overview

	Scheduled	Real
Work-hours:	720	965
Average hours per week:	45	60.3

Time per discipline

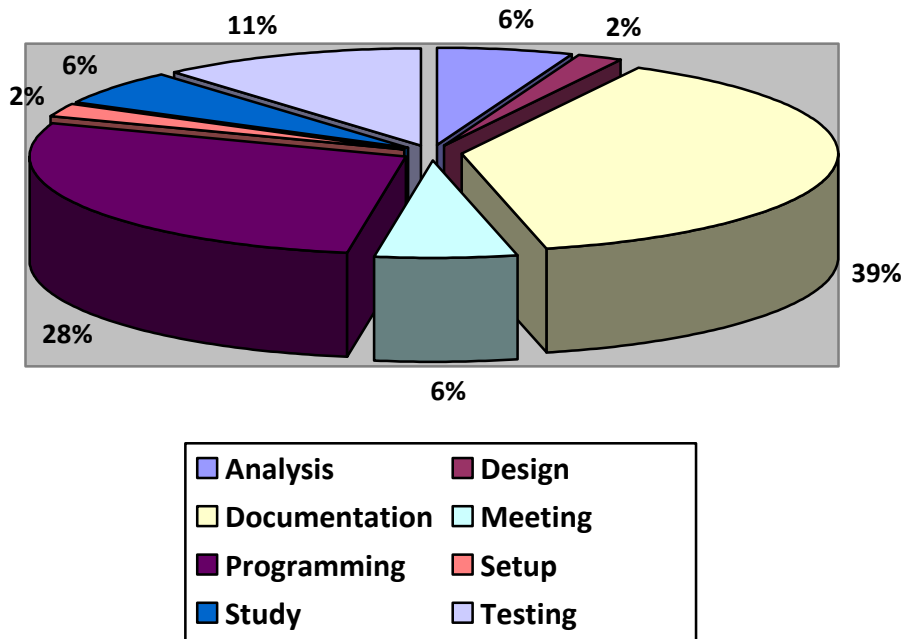


Figure 9: Time per discipline

Time per project member and iteration/week

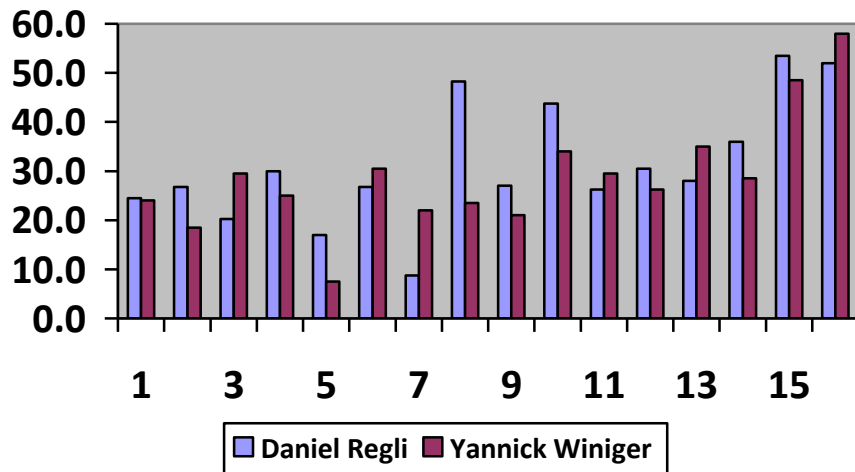


Figure 10: Time per project member and iteration/week

Part III: Technical Report

1 Analysis

Intention This chapter specifies functional and non-functional requirements for the new HERAS^{AF} PEP and HERAS^{AF} PDP Web Service implementation. Alternative approaches to meet these requirements are discussed. Which approach was taken and the implementation of it is discussed in the subsequent chapters.

1.1 Requirements for the PEP

Functional requirements

Requirements

Intercepting / Filter method	<p>A way must be provided to intercept an application flow before a secured item can be accessed.</p> <p>See part 3 chapter 1.5.1 for more information.</p>
Request information	<p>For each secured item the user needs to be able to give additional request information about the subject, resource, action and environment as specified by the XACML 2.0 specification [XACML].</p> <p>See part 3 chapter 1.5.2 for more information.</p>
XACML request creation	<p>The PEP must be able to generate a standard-compliant XACML 2.0 requests, based on the request information and data from other sources.</p> <p>See part 3 chapter 1.5.7 for more information.</p>
Handling of XACML response	<p>The PEP must assure that obligations in an authorization decision are handled properly and that they influence the decision handling depending on the decision and the fulfillment of the obligations.</p> <p>See part 3 chapter 1.5.9 for more information.</p>
Concurrency	<p>The PEP must handle concurrency because it can be used in a concurrent application.</p>
Relay response to application	<p>The PEP must provide a method to relay the response based on the 4 decisions <i>Permit</i>, <i>Deny</i>, <i>Indeterminate</i> and <i>NotApplicable</i> to the application. The additional information contained in the response must be included.</p> <p>See part 3 chapter 1.5.9 for more information.</p>
Extensibility	<p>There must be interfaces to extend the current PEP implementation.</p> <ul style="list-style-type: none"> • Interception: The integration of any technology that intercepts the application flow must be supported. • Request information: The integration of any technology to add additional request information for a secured item must be supported. (e.g. request information in a XML file or a data base)



- PDP: The integration of any PDP implementation must be supported. (e.g. PDP Web Service Stub)
- Attribute data types: The integration or usage of any attribute data type must be supported.
- Response handling: The integration of an application specific response handler must be supported.
- Obligation handling: The integration of custom obligation handler into the obligation handling process must be supported.
- XACML implementation: The integration of any XACML 2.0 implementation must be supported.

Non-functional requirements

Usability

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

Maintainability, Changeability

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

Reliability

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

1.2 Requirements for the PEP-PDP Communication

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 [SOAP] data that encapsulates XACML 2.0 data by using the SAML 2.0 Profile of XACML [SAMLXACML]. See Part 3 chapter 1.6.3 for more information.
SAML elements	The Web Service server and client must provide a method to include the handling of SAML [SAML] elements. See Part 3 chapter 1.6.3 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

The PEP-PDP Communication cannot tolerate any faults. The security depends on the PEP-PDP Communication as much as where the policies are evaluated.

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.

1.3 Old HERAS^{AF} PEP

Intention

The goal of this chapter is to describe the analysis of the old HERAS^{AF} PEP. This contains a brief description of the core elements and concepts.

The documentation of the old PEP can be found in the paper [CS07PEP].

Overview

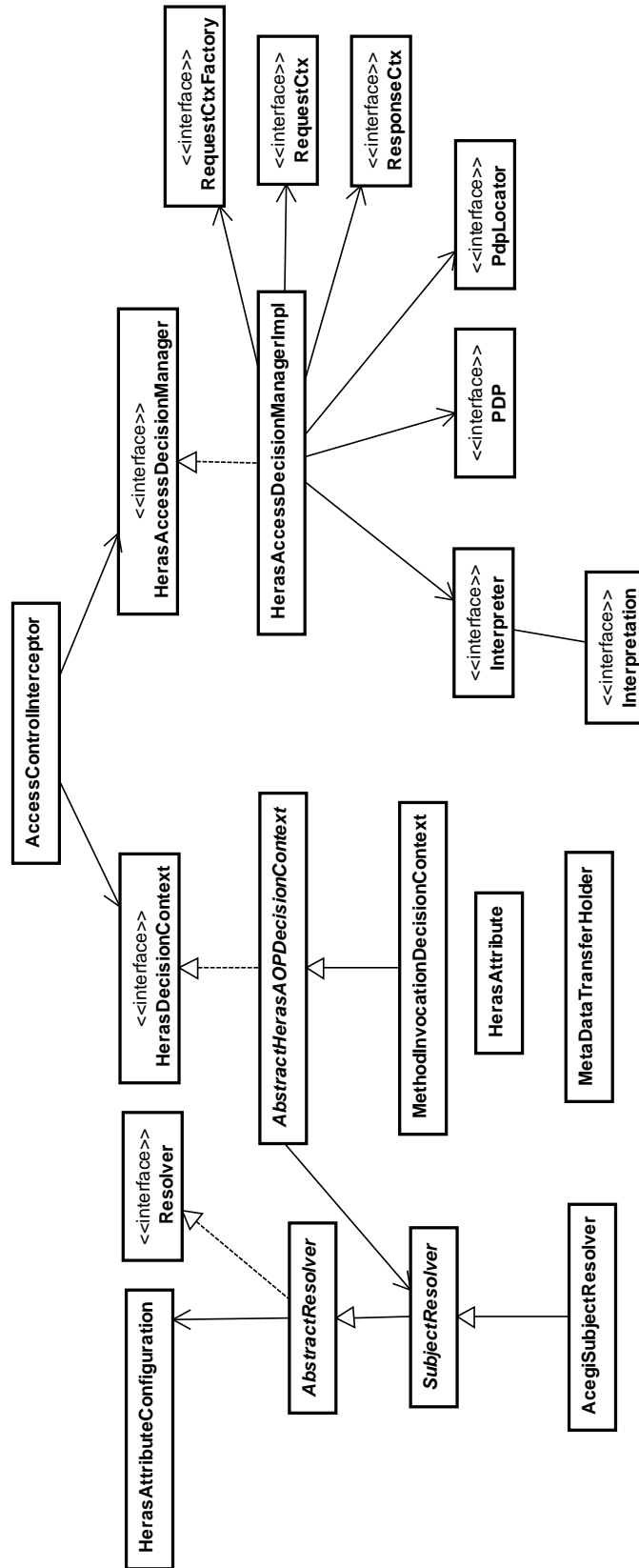


Figure 11: Class diagram of old PEP

HerasAccess Decision Manager Impl

The `HerasAccessDecisionManagerImpl` is the core of the old PEP. It defines the program flow by combines many of the adaption points like the `RequestCtxFactory`, the `PdpLocator/PDP` and the `Interpreter/Interpretation` with each other.

PdpLocator and PDP

The interfaces `PDPLocator` allows to implement different strategies how PDPs are located.

The interface `PDP` introduces a level of abstraction that hides the actual implementation. This allows using local PDPs but also remote PDP for example abstracted by a PDP Web Service client that implements this interface.

We further analyzed this concept for the new implementation in chapter 1.5.6.

Interpreter & Interpretation

Responses from the PDP are handled by an interpreter that returns an interpretation. Depending on the type of interpretation the decision manager decides to throw a runtime exception or not.

The `Interpretation` class limits the interpretation of a response to a few possible ACEGI oriented result states and a simple text message. The decision manager needs to know this interpretation class.

In addition, the decision manager limits the possibilities of result interpretations, by throwing exceptions on the basis of a fixed set of interpretation results.

To be fully customizable as specified by the new requirements the decision manager should delegate the handling of a response completely to an interpreter and only be concerned whether the interceptor should continue the execution of an action or abort it. With this design the interpreter would have the power to decide if he wants to throw an exception or otherwise handle a denied decision request.

HerasAttribute

The class `HerasAttribute` aims to be a generic value object that can represent a Subject, Resource, Action or Environment attribute. A Subject for example is represented by a collection of these `HerasAttribute`.

Representing a subject only by a collection of `HerasAttributes` doesn't fully describe one or multiple Subjects as specified by the XACML 2.0 standard. A collection of `HerasAttributes` does not contain information about the Subject's subject category nor is there a way of telling which `HerasAttribute` is part of which `Subject`.

To be fully compatible with XACML 2.0 as specified by the new requirements the PEP's internal data structure needs to be based on the XACML 2.0 XML schemas and the elements defined there. By having this kind of data structure it is possible to allow the user of a PEP to define the content of each part of the decision request and thereby use the full functionality of XACML 2.0.

HerasAttribute Configuration

The goal behind the `HerasAttributeConfiguration` is to simplify the writing of decision request information (e.g. with Java annotations) by removing all data from Attributes that can be retrieved on the basis of other data.

For example Attributes with the same ids always have the same data types; therefore the data type can be omitted when writing decision request information and is later resolved from the Attribute's id.

The `HerasAttributeConfiguration` allows mapping ids to data types and provides methods to look up the data type of a specific id.

We applied this concept in the new design. See part 3 chapter 0 for further information.

Resolver

The Resolvers are responsible to build `HerasAttributes` based on `MetaDataTransferHolder` objects. Each `HerasDecisionContext` holds a `Resolver` for Subject, Resource, Action and Environment and delegates a collection of `MetaDataTransferHolders` to the `resolve` method.

A `Resolver` is able to resolve specific meta data to `HerasAttributes`. By using the Resolvers concept new Resolvers can easily be integrated through the Spring application context. To support different technologies to define request information, such as Java annotations or an XML declarative approach, the `MetaDataTransferHolder` is used. The `MetaDataTransferHolder` encapsulates the concrete request information definition technology and the value of the request information. With this holder object the resolver infrastructure may be used regardless of the concrete request information definition technology.

Heras DecisionContext

The `HerasDecisionContext` is responsible to obtain the different `HerasAttributes` for the `HerasAccessDecisionManager`.

Therefore it encapsulates the relevant information gathered from the specific interceptor technology by providing `HerasAttributes`. The `AbstractHerasAOPDecisionContext` encapsulates the logic of analyzing and evaluating meta data in an aspect oriented programming approach.

`AbstractHerasAOPDecisionContext` works on method objects, classes and interfaces and there collects the Attribute annotations.

Reasonable Defaults

Request information definition technology like a Java annotation can define a request information element without setting an actual value. In such a case it is assumed that the value of that element should be filled with the name or value of the class, method or the parameter it is defined on.

For example, a Resource element without a value defined on a class would result in a Resource Attribute with a value equal to the class's name.

We applied this concept in the new design. See part 3 chapter 3.1.3 for further information.

1.4 Spring Web Service

Intention

An analysis for the PDP Web Service of the Spring Web Service (Spring WS 1.5.x) framework was done to check if it meets the requirements and therefore could be used for the Web Service server and client.

The goal of this chapter is to describe some of the key benefits and features of Spring WS in relation to the PDP Web Service requirements.

More details about Spring WS can be found on the official Spring WS website [SpringWS].

Transports

Spring Web Services supports multiple transport protocols. The most common is the HTTP transport, for which a custom Servlet is supplied, but it is also possible to send messages over JMS, and even email.

The following part is about the HTTP transport.

MessageDispatcherServlet

This is the standard Servlet for HTTP request handling and dispatching of messages. It is specified in the web.xml in the web application.

TransportContext

The `TransportContext` class provides access to the underlying transport, either on the client or server side. This can be helpful if it is needed to set custom HTTP headers or to resolve the IP address of a client.

WSDL

The WSDL describes the Web Services. It can be automatically exposed with two different ways. Either it is generated from an XML Schema or it is supplied as .wsdl file by the user. The generation will happen during the application context wiring phase.

*Marshalling /
Unmarshalling*

A marshaller serializes an object to XML, and an unmarshaller deserializes XML stream to an object.

Spring WS provides a JAXB1 and JAXB2 [wikiJAXB] marshaller implementation. The following explanation is for the JAXB2 marshaller class (`Jaxb2Marshaller`) because HERAS^{AF} uses JAXB2. The `Jaxb2Marshaller` can marshal and unmarshal directly into or from JAXB objects.

The `Jaxb2Marshaller` needs to know which JAXB classes can be used. This can be easily defined with the `contextPath` property.

The `Jaxb2Marshaller` or any other marshaller can be used in connection with an `AbstractMarshallingPayloadEndpoint`. When extending from `AbstractMarshallingPayloadEndpoint`, the `invokeInternal(Object)` needs to be implemented. The passed `Object` represents the unmarshalled request payload (a concrete JAXB object), and returns a JAXB object that will be marshalled into the response payload.

*Endpoint
mapping*

The endpoint mapping is responsible for mapping incoming messages to appropriate endpoints.

Spring WS provides a few endpoint mappings out of the box. The two most useful for the PEP-PDP Context Web Services as well as the PAP-PDP Policy Web Services are explained.

PayloadRootQNameEndpointMapping

This endpoint mapping uses the qualified name of the root element of the request payload to determine the endpoint that handles it.

The class also allows defining a default endpoint mapping. It maps everything which is not mapped to the default endpoint.

SimpleActionEndpointMapping

This endpoint mapping maps WS-Addressing actions to endpoints via an exposed mappings property in the standard Spring application context.

Ws Security

There are basically two ways to add security to Spring WS. Both ways can be configured easily through the application context of Spring.

XwsSecurityInterceptor

Spring WS provides a `XwsSecurityInterceptor` class which implements the following basic security features:

- Authentication
- Digital Signature
- Encryption & Decryption

The above security mechanisms are not for free. All of them, especially encryption and description require substantial amounts of memory, and will also decrease performance.

Wss4jSecurityInterceptor

Spring WS provides also a `Wss4jSecurityInterceptor` class which allows to use the following standards implemented by WSS4J [WSS4J]:

- OASIS Web Services Security: SOAP Message Security 1.0 Standard 200401, March 2004
- Username Token profile V1.0
- X.509 Token Profile V1.0

1.5 Key aspects of new HERAS^{AF} PEP

Intention

The goal of this chapter is to outline the key aspects of the new HERAS^{AF} PEP implementation. The different solution possibilities and their advantages and disadvantages are discussed.

1.5.1 Interception of a method call

Overview

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

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

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

The definition of the pointcut is in Spring AOP syntax. There is no use to run everything through a proxy object. Only the classes which are protected by the PEP should be run through the proxy object.

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



Advantage

- A solution already exists from the proof of concept

Disadvantage

- All method calls are made through a proxy object, because there is no possibility to define a package. Makes the application slower.
- Configuration is very complicated to run some packages or a few classes through the proxy
- The user cannot cast to a concrete type anymore. Only interfaces are supported.

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

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

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

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

Example 1: Spring AOP AspectJ syntax pointcut code

Advantage

- Configuration is simple, to run only some packages or a few classes through the proxy
- Only classes which are defined with the regular expression are run through the proxy
- The user is able to cast to concrete types.

Disadvantage

- none

Decision

See part 3 chapter 3.1.3.1 for more information on which alternative got implemented.



1.5.2 Resolving request information

Overview

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

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

Alternatives

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

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

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

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

Figure 12 shows an example with 1 interceptor technology (pink), 2 request information sources (green) and 4 XACML Attributes (yellow). The resulting resolver classes are grey.

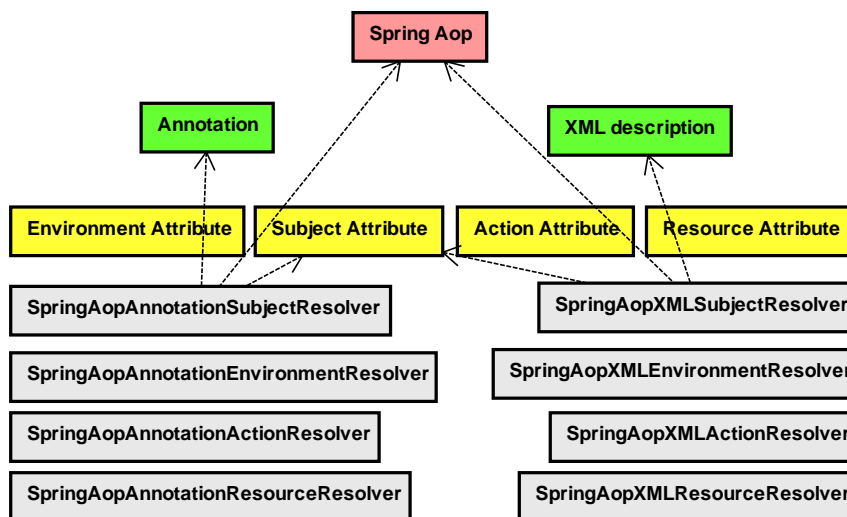


Figure 12: Resolvers alternativ 1

Advantage

- The user can use or create a decision context and resolvers specifically for the interceptor technology and request information source he needs.

Disadvantage

- High implementation costs, if resolvers and decision contexts for multiple interceptor technologies and request information sources should be



 implemented

- Each decision context is doing almost the same work.
-

Alternative 2: An abstract resolver for each interceptor

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

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

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

Figure 13 shows an example with 1 interceptor technologies (pink), 2 request information sources (grey) and 4 XACML Attributes (yellow). The resulting resolver classes are grey.

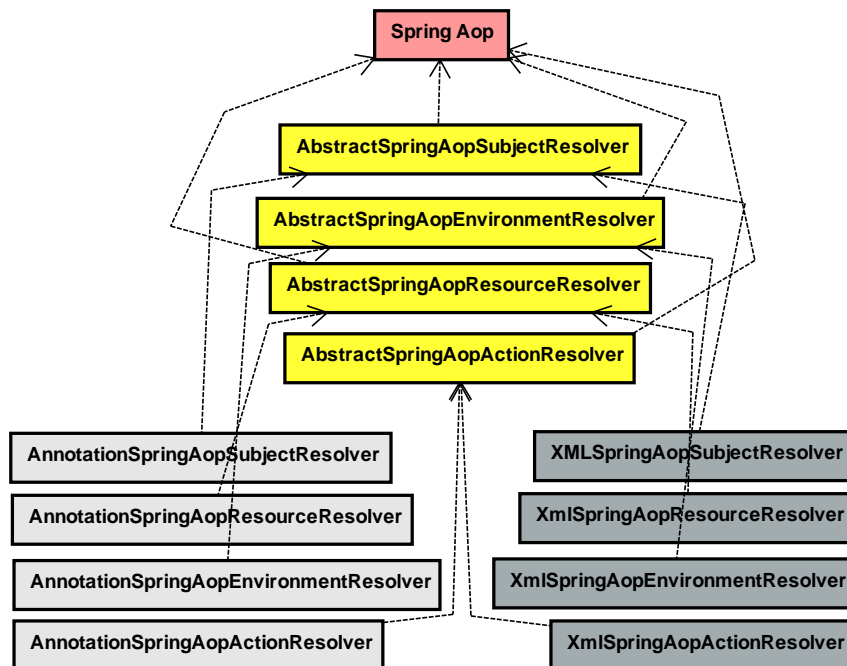


Figure 13: Resolvers alternativ 2

Advantage

- Decision contexts only depend on the interceptor technology.
- The user can use or create a decision context and resolvers, specifically for the interceptor technology and request

Disadvantage

- Medium implementation costs, if resolvers and decision contexts for multiple interceptor technologies should be implemented

information source he needs.

Decision See part 3 chapter 3.1.3.1 for more information on which alternative got implemented.

1.5.3 Parsing request information

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

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

Alternatives

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

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

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

Advantage

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

Disadvantage

- Poor performance

Alternative 2: Parse data only once (with cache)

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

Advantage

- New request information can be added to the system at runtime.
- Good performance

Disadvantage

- Changes to already parsed request information are not possible at runtime.

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

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

Advantage

- Good performance

Disadvantage

- New request information and changes to this information are not possible at runtime.



Decision See part 3 chapter 3.1.4.1 for more information on which alternative got implemented.

1.5.4 Java annotation gathering algorithm

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

Alternatives

Alternative 1: Default annotation inheritance logic of Java

The default annotation inheritance logic of Java is used.

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

Advantage

- Low costs, because the existing annotation inheritance logic of Java 1.5 can be used.
- Good performance, because all information is already present on the processed class and no class hierarchy has to be traversed.

Disadvantage

- Little flexibility on how to place annotations into the code.
- The data of the same annotations on different levels in the class hierarchy can't supplement each other.

Alternative 2: Custom annotation inheritance and combination logic

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

Advantage

- High flexibility on how to place annotations into the code.
- The data of the same annotations on different levels in the class hierarchy can supplement each other.

Disadvantage

- High costs, because a custom logic has to be implemented.
- Bad performance, because the information has to be gathered from the whole class hierarchy up to the root class.

Decision See part 3 chapter 3.1.4.1 for more information on which alternative got implemented.

1.5.5 Attribute data types

Overview

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

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

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

Alternatives

Alternative 1: All data types are allowed.

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

Advantage

- Simple design and low costs.
- The PEP can be used with any kind of attribute data type.

Disadvantage

- The user is responsible to only use data types that are accepted by the PDP.
- All data types need to implement the conversion to the XML form directly in their `toString` method.

Alternative 2: Injection of attribute data type handlers.

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

Advantage

- The conversion between an object of a certain data type into its XML form is separated from the object itself.
- The handler can still use the `toString` method, if this approach is preferred or necessary.

Disadvantage

- The set of data type handlers needs to be synchronized manually with the set of data types accepted by the PDP.

Alternative 3: Dynamic supply of attribute data type handlers.

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

Depending on the type of locator, the data type handlers could be injected directly into the locator or gathered from another source.

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

Advantage

- Allows obtaining data type handlers from any possible source in a dynamic way.

Disadvantage

- Complex design in association with the PDP locator.

Decision

See part 3 chapter 3.1.1.1 for more information on which alternative got implemented.

1.5.6 Finding/locating a PDP

Overview

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

Alternatives

Alternative 1: Integration with locator pattern

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

Advantage

- PEP doesn't know which PDP is given to him
- Multiple PDPs could be configured within the PDP locator.

Disadvantage

- Users implementing a simple PDP without special locating behavior would still need to use and understand the locator pattern.

Alternative 2: Direct injection of PDPs

PDPs are directly injected into the PEP.

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

Advantage

- PEP doesn't know which PDP is given to him
- Multiple PDPs could be configured within the PDP implementation.
- Users implementing a simple PDP without special behavior are not bothered by the concept of locating a PDP.

Disadvantage

- No separation of concerns, the functionality of locating the right PDP is hidden within the PDP implementation.



Decision See part 3 chapter 3.1.1.1 for more information on which alternative got implemented.

1.5.7 XACML request context creation

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

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

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

Alternatives

Alternative 1: Spring wiring with scope attribute to prototype

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

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

Advantage

- Everything is wired through spring.

Disadvantage

- Bad design, inelegance.
- The context object is not a data transfer object.

Alternative 2: Creation with a factory class

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

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

Advantage

- Factory is a singleton and could manage the creation and the transformation.
- The transformer objects could be injected through spring into the factory.
- The request context would act as data transfer object and wouldn’t contain any logic.

Disadvantage

- none



Decision See part 3 chapter 3.1.2.1 for more information on which alternative got implemented.

1.5.8 XACML response

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

Alternatives

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

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

Advantage

- Minimal costs

Disadvantage

- Dependency to JAXB and the XACML implementation
- Not read-only.

Alternative 2: Create own data structure

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

Advantage

- Some XACML elements can be simplified or ignored if not needed. (e.g. enumerations)
- Independent from the XACML implementation.

Disadvantage

- Higher costs

Decision See part 3 chapter 3.1.1.1 for more information on which alternative got implemented.

1.5.9 Decision handling

Overview Decision handlers are responsible to handle the XACML responses returned by a PDP. See [XACML] for more information about the elements which are part of the response.

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

*Alternatives***Alternative 1: Throwing Runtime Exception**

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

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

Advantage**Disadvantage**

- | Advantage | Disadvantage |
|--|--|
| <ul style="list-style-type: none"> • Fast way to go through the long program stack hierarchy back to the application level. • The exception contains all decision information so that it is accessible at the application level. • Applications can define a common way how to handle this specific runtime exceptions (e.g. in a web application define a rule for this exception). • Possibility to define own Runtime Exceptions for easier integration into an existing application. | <ul style="list-style-type: none"> • If access is not permitted, there is always an exception thrown. If an application wants a silent deny (no exception, the called method is just returned with a null value), this alternative is not viable. |

Alternative 2: Return simple Boolean value

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

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

Advantage**Disadvantage**

- | Advantage | Disadvantage |
|---|---|
| <ul style="list-style-type: none"> • Simple way to handle decisions. | <ul style="list-style-type: none"> • The information about a decision is lost, if not handled by the decision handler. • Integration into existing systems is not possible, if it depends on exceptions. • A common way to handle deny-decisions is harder to achieve. |

Decision

See part 3 chapter 3.1.1.1 for more information on which alternative got implemented.

1.5.10 Obligation handling

Overview

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

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

Alternatives

Alternative 1: Default obligation processor with configurable handlers

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

The application flow of handling a response would then be:

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

Advantage

- Obligations are always handled (not possible to ignore obligations at all).
- Easy configuration of handlers with spring.
- Because of a default implementation the user doesn't have to implement an obligation processor.

Disadvantage

- The user can still implement an "all-matcher-obligator-handler" which returns always a true even if the obligation could not be fulfilled.

Alternative 2: Delegate obligation handling to decision handlers.

The handling of obligations is delegated to the decision handler.

Advantage

- Flexibility
- The user can implement decision handlers that ignore obligations or handle them in a special way.

Disadvantage

- With a decision handler that ignores obligations, the PEP would violate its specified behavior.



-
- Within the decision handler, obligations could still be processed by a default obligation processor and benefit from the easy configuration of handlers with spring.

Decision

See part 3 chapter 3.1.1.1 for more information on which alternative got implemented.

1.6 Key aspects of the HERAS^{AF} PEP-PDP Communication

Intention The goal of this chapter is to outline the key aspects of the new HERAS^{AF} PEP-PDP Communication implementation. The different solution possibilities and their advantages and disadvantages are discussed.

1.6.1 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 need. 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 14 shows this approach.

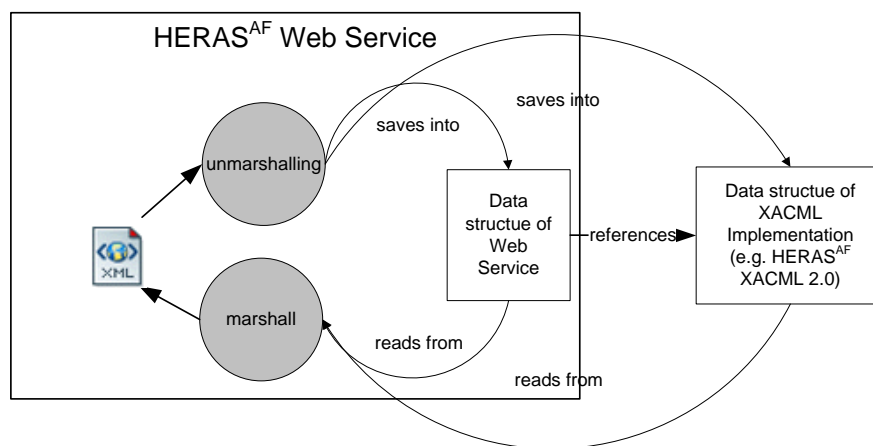


Figure 14: Marshalling alternative 1

Advantage	Disadvantage
<ul style="list-style-type: none"> • Faster processing • No “duplicated code” in HERAS^{AF} 	<ul style="list-style-type: none"> • 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 15 shows this approach.

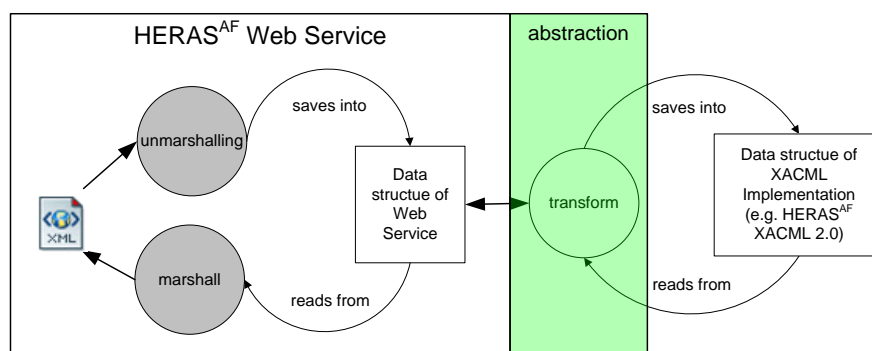


Figure 15: Marshalling alternative 2

Advantage

- No dependency to any XACML implementation
- Abstraction possibility.

Disadvantage

- Slower processing because the XML processing goes through one more abstract layer. (e.g. XACML XML → Web Service data structure → XACML implementation data structure)
- Duplicated code inside HERAS^{AF}

Decision

See part 3 chapter 3.2.3.1 for more information on which alternative got implemented.

1.6.2 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 [SpringWS] 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

See part 3 chapter 3.2.2.1 for more information on which alternative got implemented.

1.6.3 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

Disadvantage

- | Advantage | Disadvantage |
|---|--|
| <ul style="list-style-type: none"> JAXB objects created with the corresponding factory are standard-compliant. | <ul style="list-style-type: none"> 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

Disadvantage

- | Advantage | Disadvantage |
|--|---|
| <ul style="list-style-type: none"> New instances of JAXB classes are guaranteed to be standard-compliant. | <ul style="list-style-type: none"> Could lead to runtime exceptions when usingmarshallers or unmarshallers who cannot create new JAXB objects with non-default constructors. |

Decision

See part 3 chapter 3.2.3.1 for more information on which alternative got implemented.

2 Architecture

2.1 HERAS^{AF} PEP

Intention This part of the documentation explains the architecture of the HERAS^{AF} PEP. It will explain each component and its dependencies. The level of detail is normal as the granular details are explained in the developers guide [RW08PEPDev].

Overview The structure of the HERAS^{AF} PEP is divided into modules. This helps to replace certain modules for using different implementation. Figure 16 shows all the modules and their dependencies to each other.

The violet modules can be replaced by specific implementations (integration moduls).

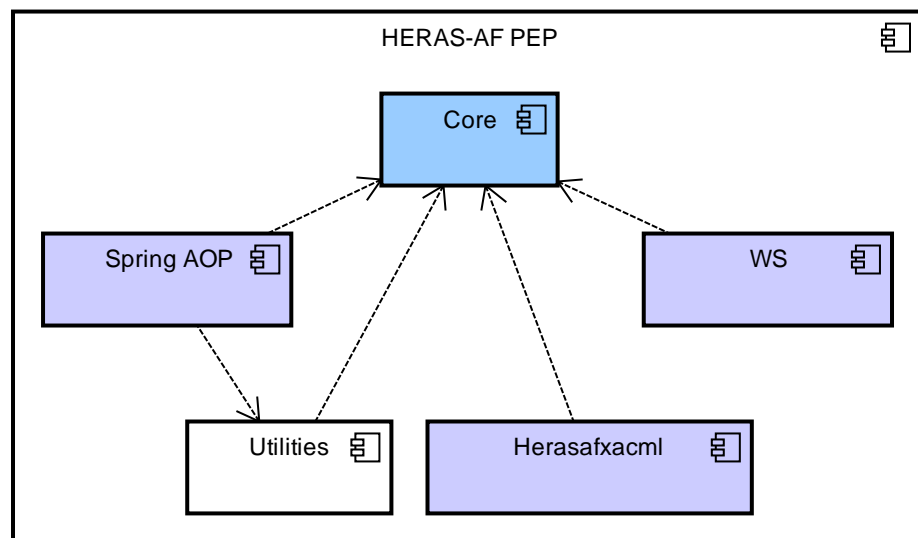


Figure 16: Module overview

Modules

Core

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

Spring AOP

The Spring AOP module contains the Spring AOP interceptor technology, the resolvers and the Spring AOP context. The resolvers are used to get acquire request information. This data is kept in the Spring AOP context and is used to create a request.

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

Herasafxacml

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

made here.

This module is only used if HERAS^{AF} XACML is used. Otherwise it is replaced by an own integration implementation.

Utilities

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

WS

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

Interaction scenario

The following description explains the sequence of Figure 17.

Description

0. A protected method is invoked by a subject.
1. The joint point described by a pointcut is reached and the advice (`AccessControlAdvice`) is called.
2. Within the advice a `DecisionContext` is created via the factory. All request information is resolved in the factory and the values are set in the decision context object.
3. The decide method of a `DecisionManager` is called with the decision context as parameter.
4. The `DecisionManager` creates a `RequestContext` based on the values of the decision context via a factory.
5. The factory uses an XACML implementation to transform the PEP request representation into an actual XACML request.
6. A `Pdp` is located using the locate method in the `PdpLocator` and a `Pdp` instance is returned.
7. The evaluation of the XACML request is made with the evaluate method on the `Pdp` instance which returns a `ResponseContext`.
8. A `DecisionHandler` handles the result (`ResponseContext`) and indicates if access is granted or should be aborted.

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

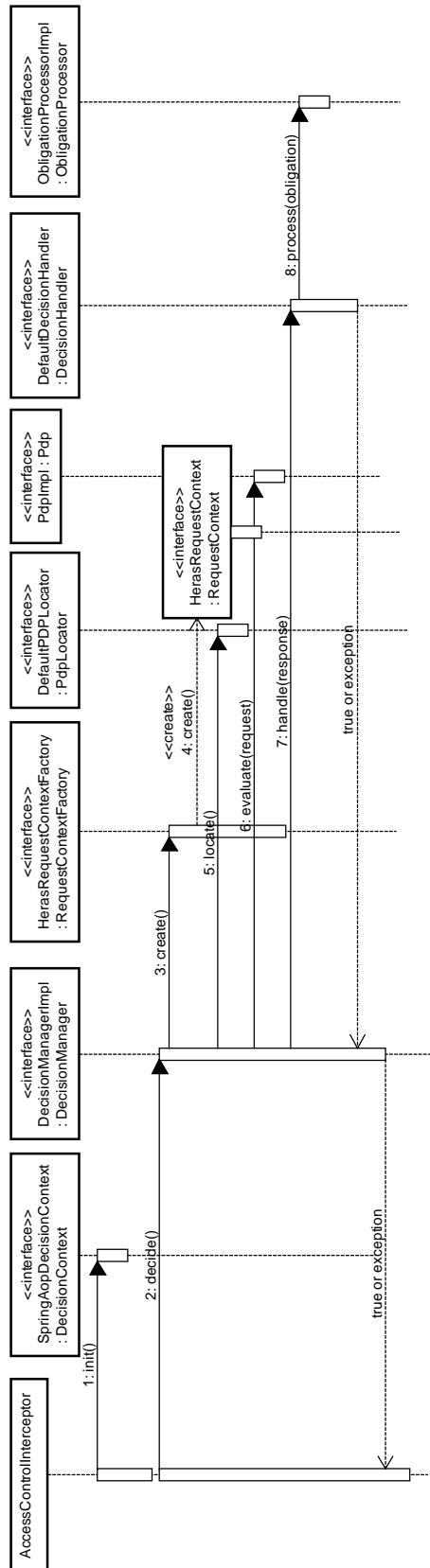


Figure 17: Sequence diagram of the application flow

Further
Scenarios

Further and more details scenarios can be found in the Developer's Guides of the PEP [RW08PEPDev] and PEP-PDP Communication [RW08PEPPDPDev].

2.2 HERAS^{AF} PEP-PDP Communication

Intention

This part of the documentation explains the architecture of the HERAS^{AF} PEP-PDP Communication. It will explain the client/server architecture as well as each component and its dependencies. The level of detail is normal as the granular details are explained in the developers guide [RW08PEPPDPDev].

Client-Server
model

Figure 18 shows the client-server architecture with HERAS^{AF} PEP and HERAS^{AF} PDP components. The HERAS^{AF} PEP agent is attached to an application to control access and acts as a client of the PDP. The HERAS^{AF} PDP acts as the server.

The WS module of the client and the Context-WS module of the server communicating whenever a secured method in the client application is called and a decision request has to be evaluated.

The communication is via HTTP and SOAP over a network.

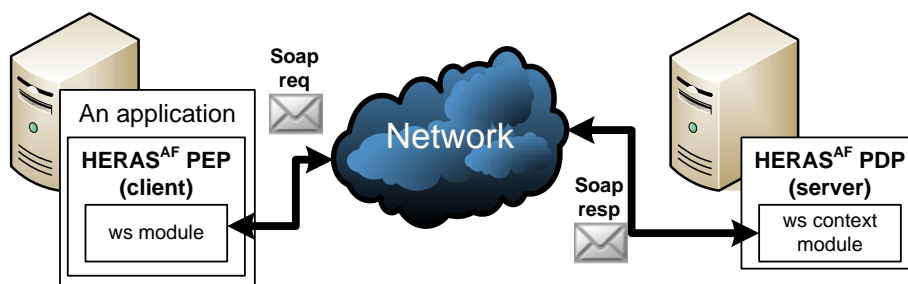


Figure 18: Client-server model of HERAS^{AF} PEP-PDP Communication

Overview

The structure of the HERAS^{AF} PEP-PDP Communication is divided into three modules. This helps to replace certain modules for using different implementation. Figure 19 shows the modules and their dependencies to each other.

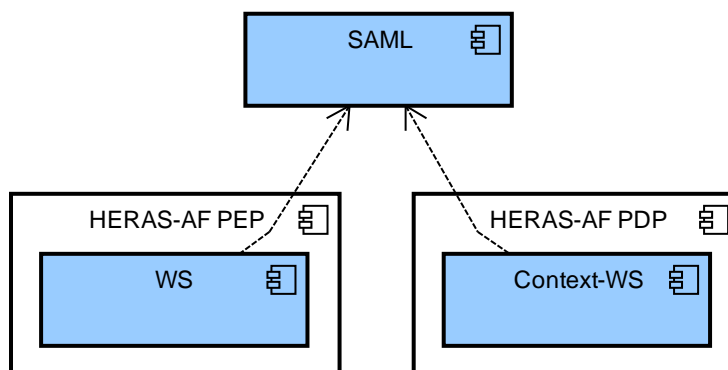


Figure 19: Module dependencies of HERAS^{AF} PEP-PDP Communication



Modules

Context-WS (HERAS^{AF} PDP)

The Context-WS module contains classes for a PDP Web Service endpoint. This includes a server SAML handler.

WS (HERAS^{AF} PEP)

The WS contains classes to connect the PEP to a PDP Web Service. This includes a client SAML handler.

SAML

The SAML module contains common classes for SAML aspects like the exception handling, the generated JAXB classes, a specialized marshaller with namespace resolution and some classes to guarantee the standard-compliant transmission of SAML over SOAP over HTTP.

3 Design

3.1 HERAS^{AF} PEP

Intention

This part of the documentation explains the design of the HERAS^{AF} PEP. It will explain each module and its dependencies. The level of detail is normal as the granular details are explained in the Developer's Guide of the PEP [RW08PEPDev].

3.1.1 Core module

Overview

The Core module is the heart of the HERAS^{AF} PEP implementation. It contains the annotations to annotate all elements for a XACML request and the data structure to save the request as well as the response. This module also provides various interfaces to create the integration layer for a different XACML implementation and intercepting technology. The whole process of a PEP on how to initiate and handle an XACML request/response is contained in this module too.

Package diagram

org.herasaf.pep

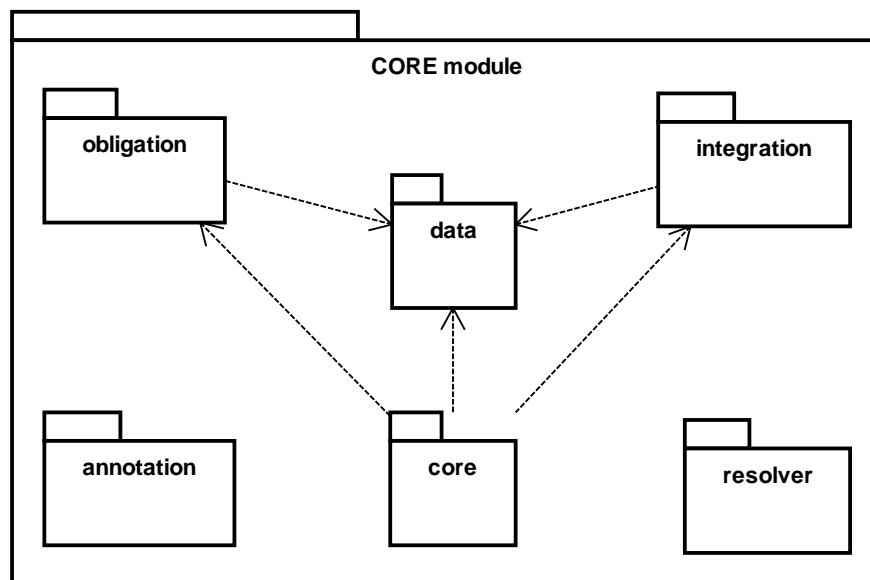


Figure 20: Core module

Package description

annotation

The annotation package contains all annotation to annotate the elements for a XACML request and the annotation to protect a class. The most import ones are @Protected, @Subject, @Resource, @Action, and @Environment.

core

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

data

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

integration

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

obligation

The obligation package contains components in connection with obligations. Classes and interfaces to process and handle obligations.

resolver

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

util

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

3.1.1.1 Decisions*Overview*

This part explains all the decisions which occurred during the implementation phase of the core module. In the previous chapter 1.5 in part 2 were some key aspect mentioned. These helped to make a final decision on which solution to implement.

Attribute data types handling

In the previous chapter 1.5.3 in part 2 are three alternatives described to handle the attribute data types. One solution is to allow all data types without verification. The second possible solution is to inject attribute data type handlers into the PEP and the last solution is to provide a dynamic data type handler locator where the handlers could be added dynamically.

Discussion of the implementation

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

Finding and locating a PDP

In the previous chapter 1.5.6 in part 2 are two alternatives described to find or locate a PDP. The first alternative is to use a PDP locator. The second



alternative is to directly inject the PDP's into a list.

Discussion of the implementation

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

XACML Response

In the previous chapter 1.5.8 in part 2 are two alternatives described to hold the data structure of an XACML response. The first solution is to use the same data structure as the HERAS^{AF} XACML 2.0 Implementation and the second solution is to create an own data structure.

Discussion of the implementation

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

Decision Handling

In the previous chapter 1.5.9 in part 2 are two alternatives described to handle a decision received by the PEP from the PDP. The first solution is to throw a runtime exception whenever the access authorization is not permitted. The second solution is to return a simple Boolean value to indicate if the access authorization is permitted or not.

Discussion of the implementation

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

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

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

Obligation Handling

In the previous chapter 1.5.10 in part 2 are two alternatives mentioned to handle the obligation process. The first solution is to provide a default obligation processor with configurable handlers. The second solution is to simply delegate the obligation handling to the decision handlers.

Discussion of the implementation

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



3.1.2 Herasafxacml module

Overview

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

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

Package diagram

org.herasaf.pep.integration.heasafxacml

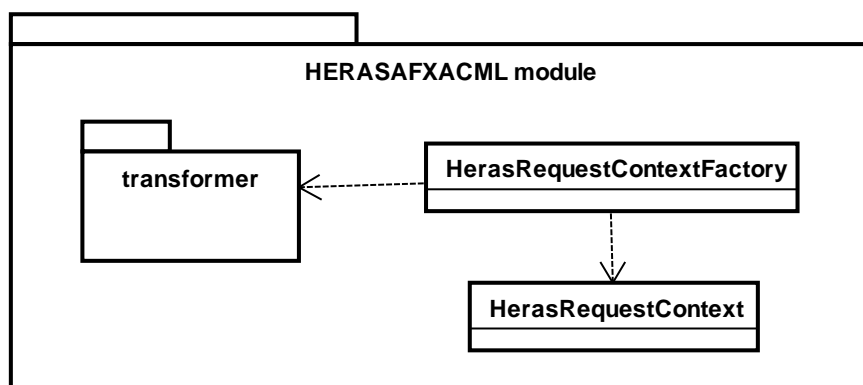


Figure 21: Herasafxacml module

Package description

(default) herasafxacml

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

transformer

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

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

3.1.2.1 Decisions

Overview

This part explains all the decisions which occurred during the implementation phase of the Herasafxacml module. In the previous chapter 1.5 in part 2 were some key aspect mentioned. These helped to make a final decision on which solution to implement.

XACML request context creation

In the previous chapter 1.5.10 in part 2 are two alternatives mentioned to create a XACML request context. The first solution is in connection with Spring and scope attribute "prototype". The second solution creates the context with a



factory.

Discussion of the implementation

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

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

3.1.3 Spring AOP Module

Overview

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

Package Diagram

org.herasaf.pep.integration.springaop

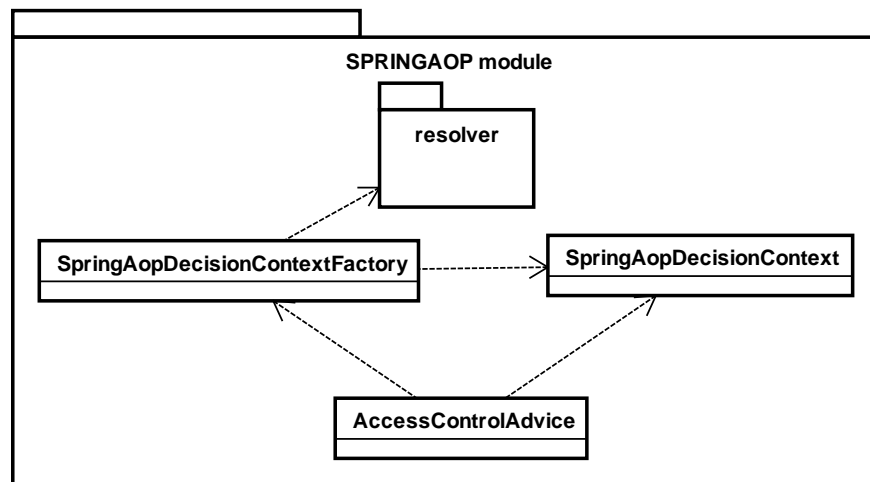


Figure 22: Spring AOP module

Package Description

(default) Spring AOP

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

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

resolver

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

SpringSecurityAopSubjectResolver).

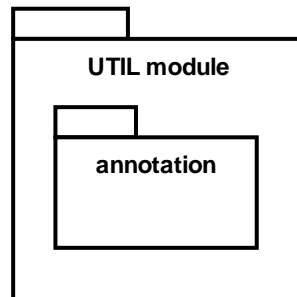
For example a resolver to resolve resource elements is contained in this package (AnnotationSpring AOPResourceResolver).

3.1.3.1 Decisions

<i>Overview</i>	This part explains all the decisions which occurred during the implementation phase of the Spring AOP module. In the previous chapter 1.5 in part 2 were some key aspect mentioned. These helped to make a final decision on which solution to implement.
<i>Interception of a method call</i>	<p>The previous chapter 1.5.1 in part 2 mentions two alternatives how a method call could be intercepted. The first solution is to use Spring AOP syntax to define the pointcut and the second solution is to use the AspectJ syntax but still in connection with the Spring AOP libraries.</p> <p>Discussion of the implementation</p> <p>The chosen solution is to implement the pointcut with the AspectJ syntax (alternative 2). The decision fell on this alternative because it allows the user to simply configure which packages have classes to protect. This means that only those classes are run through the proxy. For this reason the chosen solution has better performance.</p>
<i>Resolving annotated elements</i>	<p>The previous chapter 1.5.2 in part 2 mentions two alternatives how to resolve an annotated element. The first solution is to implement a resolver for each interceptor technology and request information source and the second solution is to implement an abstract resolver for each interceptor.</p> <p>Discussion of the implementation</p> <p>The chosen solution is to implement an abstract resolver for each interceptor technology (alternative 2). The decision fell on this alternative because now the decision context only depends on the interceptor technology.</p>

3.1.4 Utilities Module

<i>Overview</i>	This module contains all utility classes that may be used certain implementations of interfaces in the core module. For example resolvers or interceptor technologies that have to work with Java annotations can use annotation utility classes from this module.
-----------------	--

*Package
Diagram***org.herasaf.pep****Figure 23: Utilities module***Package
Description***Annotation**

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

For example it contains an annotation parser that provides methods to gather all data from HERAS^{AF} PEP annotations present on a certain class and its class hierarchy (all super classes and interfaces). The package also contains an annotation cache that can be used by annotation parsers to save time by caching the data for later reuse.

3.1.4.1 Decisions*Overview*

This part explains all the decisions which occurred during the implementation phase of the utilities module. In the previous chapter 1.5 in part 2 were some key aspect mentioned. These helped to make a final decision on which solution to implement.

*Java annotation
gathering
algorithm*

The previous chapter 1.5.4 in part 2 mentions two alternatives how Java annotations could be gathered by an algorithm. The first solution is to use the default annotation inheritance logic of Java. The second is to implement a custom algorithm with a more advanced and complex logic.

Discussion of the implementation

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

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

*Request
information
parsing &
caching*

The previous chapter 1.5.3 in part 2 mentions three alternatives how request information could be parsed. The first solution is to parse the requested data on every call. The second is to parse the data only once and then cache it for reuse. And the third is to parse and cache all the data only once on startup.

Discussion of the implementation

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

3.2 HERAS^{AF} PEP-PDP Communication

Intention This part of the documentation explains the design of the HERAS^{AF} PEP-PDP Communication. It will explain each module and its dependencies. The level of detail is normal as the granular details are explained in the developers guide.

3.2.1 WS Module

Overview The WS module contains an integration of a Spring Web Service client. But the module provides an additional abstract layer to implement other Web Service clients. For example an Axis Web Service client.

The module also contains classes to handle SAML 2.0 attributes and provides interfaces to implement own SAML 2.0 handlers.

Package Diagram

org.herasaf.pep.integration.ws

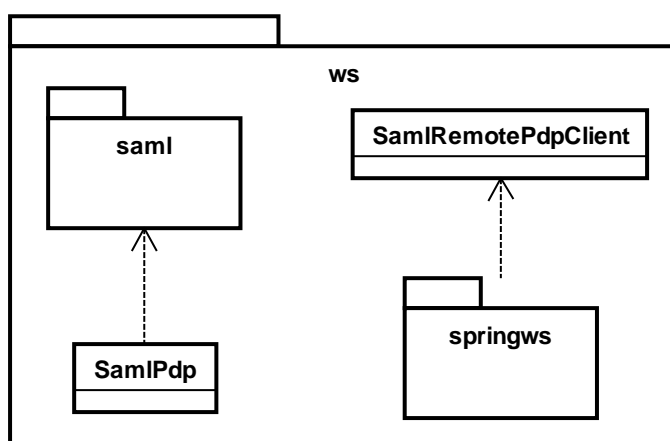


Figure 24: Packages of the WS module

Package Description

(default) ws

This package contains the interfaces and implementations for the integration of a

Web Service client into the HERAS^{AF} PEP.

saml

This package contains all classes related the SAML aspects for the communication with a remote PDP.

springws

This package contains all classes related to a Web Service client using Spring WS.

3.2.2 Context-WS Module

Overview

The Context-WS module contains the integration of the Spring Web Service Endpoint. The processing of the whole SAML/XACML request is done inside this module. It provides SAML Handler implementations and interface to delegate the SAML handling.

The endpoint has a connection to concrete PDP implementation whereas the XACML request is sent to. The received XACML response is handled and sent back to the client who initiated the request.

Package Diagram

org.herasaf.pdp.ws.context

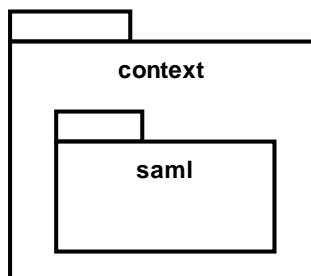


Figure 25: Packages of the Context-WS module

Package Description

saml

This package contains the SAML PDP endpoint and the server SAML handler abstract class as well as a default implementation of a SAML handler.

3.2.2.1 Decisions

Overview

This part explains all the decisions which occurred during the implementation phase of the Context-WS module. In the previous chapter 1.6 in part 2 were some key aspect mentioned. These helped to make a final decision on which solution to implement.

WSDL exposure

The previous chapter 1.6.2 in part 2 mentions two alternatives how to implement the WSDL description creation and exposure. The first alternative is to let Spring

do the generation and the second is to create the WSDL by hand. Spring would handle the exposure of the WSDL in both alternatives.

Discussion of the implementation

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.

3.2.3 Saml Module

Overview

This module contains common SAML handling and processing classes. These are in a separate module because they are used by other HERAS^{AF} sub projects as well.

All the SAML status codes which can occur in the SAML 2.0 specification are specified by a class. The factory classes are another thing to ensure the standard-compliance of the SAML.

The module provides JAXB2.0 classes to represent an SAML-XACML request and response to deploy / undeploy and query a policy.

A JAXB namespace marshaller is provided to marshal the namespaces for the defined JAXB classes.



Package
Diagram

org.herasaf.saml

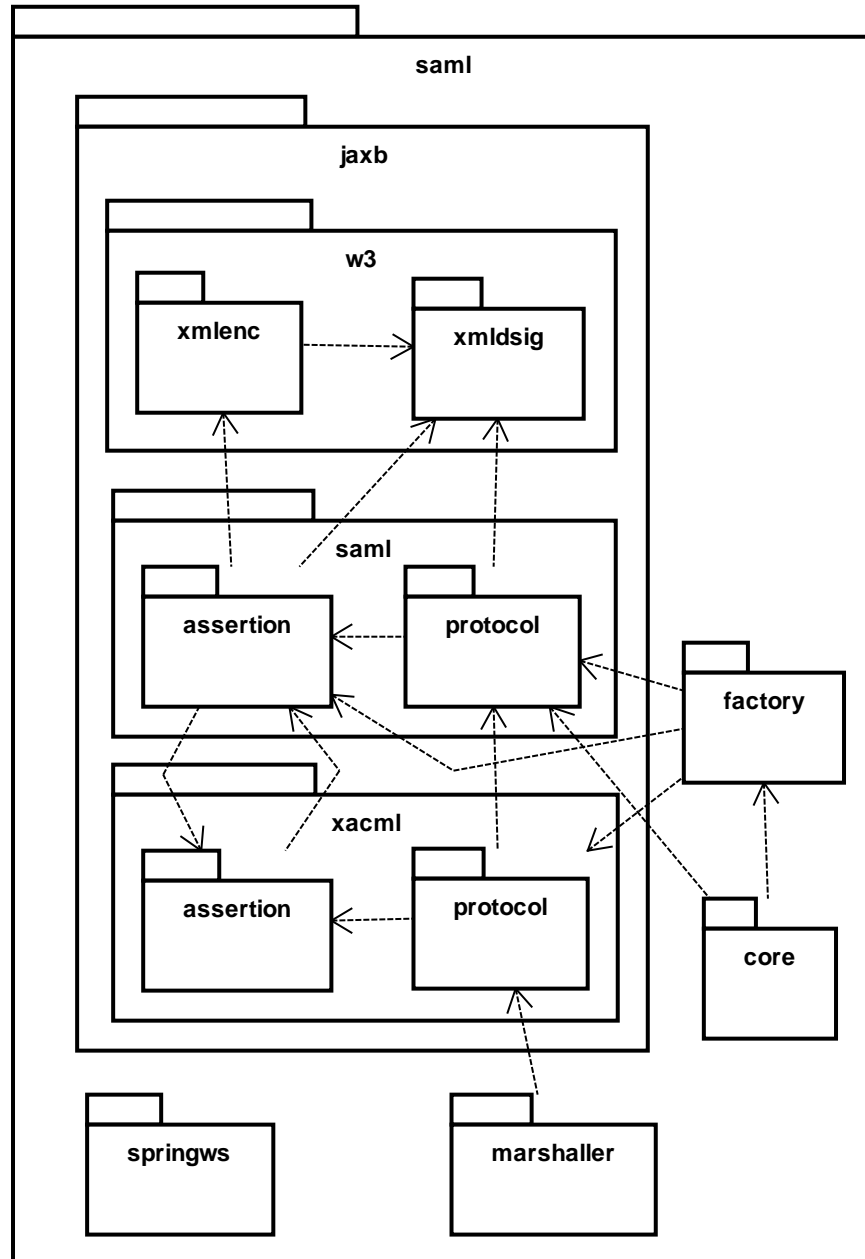


Figure 26: Saml module

Package
Description

core

This package contains the exception SAML status codes as well as the exception handling whenever a SAML element cannot be handled.

factory

This package contains factories to create and pre initialize certain SAML JAXB objects to guarantee that mandatory fields are set properly.

jaxb.saml.assertion

This package contains the generated JAXB 2.0 classes for the SAML assertions.

jaxb.saml.protocol

This package contains the generated JAXB 2.0 classes for the SAML protocols.

jaxb.w3.xmlsig

This package contains the generated JAXB 2.0 classes to add XML digital signature to SAML.

jaxb.w3.xmlenc

This package contains the generated JAXB 2.0 classes to add XML encryption to SAML.

jaxb.xacml.assertion

This package contains the generated JAXB 2.0 classes for the SAML protocols to create assertion for XACML.

jaxb.xacml.protocol

This package contains the generated JAXB 2.0 classes for the SAML protocols to query XACML.

marshaller

This package contains classes for a customized Jaxb2Marshaller. The customization allows mapping a prefix of a namespace with a predefined name.

springws

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

3.2.3.1 Decisions

Overview

This part explains all the decisions which occurred during the implementation phase of the Context-WS module. In the previous chapter 1.6 in part 2 were some key aspect mentioned. These helped to make a final decision on which solution to implement.

*Marshalling /
Unmarshalling
the XML*

The previous chapter 1.6.1 in part 2 mentions two alternatives how to implement the marshalling and unmarshalling of the XML. The first alternative is to use the JAXB classes for the HERAS^{AF} XACML 2.0 implementation and generate the additional used classes. The second alternative is to regenerate all the JAXB classes.

Discussion of the implementation

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.

*Standard
compliance*

The previous chapter 1.6.3 in part 2 mentions two alternatives how to implement the creation of standard-compliant JAXB objects.

Discussion of the implementation

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 Marshalls 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.

4 Tests

Intention In this chapter of the documentation all the issues around and about the testing from the HERAS^{AF} PEP and HERAS^{AF} PEP-PDP implementation are discussed and outlined.

The implementations are 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.

4.1 Unit test

Overview The functionality of the classes in the HERAS^{AF} PEP and HERAS^{AF} PEP-PDP Communication implementations are tested by the TestNG Framework [TestNG]. Each module which can be tested for itself has unit tests.

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 we suspect.

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.

Tests There are two types of tests performed.

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

4.1.1 Code coverage

Overview Code coverage describes the degree to which the source code of a program has been tested. It is a form of testing that inspects the code directly and is therefore a form of white box testing.

EclEmma Plugin The Eclipse Plugin-in EclEmma is used to verify the code coverage of all the classes. [EclEmma]

4.2 Integration tests

Overview The functionality of the HERAS^{AF} PEP as one piece is tested with integration tests. The module herasaf-pep-integrationtests and herasaf-pdp-ws-context-integrationtests contain integration tests scenarios.

<i>PEP core module</i>	The Core integration tests cover the decision handling process. Each of the four decision responses in connection with obligations are tested.
<i>PEP Herasafxacml module</i>	The Herasafxacml module does not contain any integration tests. There is no need to implement some as it does not contain distributed logic over classes.
<i>PEP Spring AOP module</i>	The Spring AOP module does not contain any integration tests. There is no need to implement some as it just contains the pointcut (and advice) as well as the <code>SpringAOPDecisionContextFactory</code> which creates a context.
<i>PEP utilities</i>	The Utilities module tests the process of finding and caching annotations on classes, interfaces, super classes and super interfaces.
<i>PEP ws</i>	The WS module does not contain any integration tests.
<i>PDP ws-context</i>	The Context-WS module does not contain any integration tests.

4.2.1 Cargo-maven2-plugin

<i>General</i>	<p>The cargo-maven2-plugin [cargoPlugin] is a plugin for maven. It allows deploying a maven project (e.g. packed as .war-file) to a web container, start it, run the tests and stop the web container.</p> <p>The herasaf-pdp-ws-context-integrationtests module uses this plugin for the testing of the Web Service.</p>
----------------	---

5 Final Report

Intention This part of the documentation describes the project after it was performed. It summarizes up the realization. This includes the coverage of what was achieved as well as open issues. At the end an outlook is given which steps could be performed next.

5.1 Summary

General In preliminary work we studied and familiarized us with the Spring, up to that point unknown to us. We read some material to understand the basics of the Spring Framework. This was very positive as we were confronted with Spring and its application context configuration already in the first week of the thesis.

We started the thesis in the first week by doing the setup of the project infrastructure and familiarization with additional tools and further technologies (e.g. XACML, Maven2, more Spring, etc.) and existing work. The second week was the migration of the old PEP to the new XACML implementation, which was less complex as we expected.

The analysis of the old PEP and the creation of a prototype was the next step of our path to success. We investigated a lot of time in the analysis of the old PEP because we knew the importance of a good analysis. Soon prototypes and simple tests cases were implemented for the newly created design.

The implementation began after the coaches approved our new design. They told us to completely document the analysis first, but we could not resist and started the implementation. The development went pretty well, if we would not have had troubles with Maven2 and its dependency management. The problem we had with Maven2 was that Maven created a library version mismatch and we could not fix it in the first place.

Florian Huonder told us that the PEP and the documentation of it must be completely finished by next week, as he wanted to start the Web Service part. The code and JavaDoc could be finished by the deadline, but the complete documentation was too much. We proposed to finish the documentation in our free time on weekends, etc. in order to be able to begin with the Web Service part. He agreed.

The Web Service part (PEP-PDP Communication) was up next and with it a bunch of new technologies. Spring Web Service and new standards like SOAP, SAML and the SAML Profile of XACML.

After studies and our analysis, a prototype was build which was able to communicate in a standard-compliant way. For this prototype we had an important deadline, because the other HERAS^{AF} team (PAP-Team) relied on our prototype and our experience about Spring Web Service. We share our knowledge and explained our prototype to enable them to build their own Web Service for the PAP-PDP Communication.

We continued the development and testing of our Web Service components and finished it two weeks before the Bachelor thesis ended.

A code/JavaDoc cleanup and the completion of the documentation were done in the finishing phase. Wolfgang Giersche had a number of inquiries in these two weeks about the integration of our PEP and the PAP into an existing application.

We could handle his inquiries with additional time expenses and could continue with finishing the bachelor thesis.

Altogether we worked more than we originally scheduled, but the result justifies every bit of it. We are very proud of what we achieved in this bachelor thesis.

Coverage

HERAS^{AF} PEP

We implemented the HERAS^{AF} PEP as a framework that controls access by intercepting method calls, sending decision requests to a PDP and enforcing the returned authorization decision. The framework provides the plugability to configure and adapt all components through the Spring Framework. This allows building very specific PEPs for all kind of Java applications.

A set of Java annotations enables the user to choose which methods should be protected by the PEP and what information for a request are collected. This information, acquired to create decision requests, is cached to optimize performance. Calls to protected methods are intercepted with Spring AOP. To generate decision requests it is possible to integrate the HERAS^{AF} XACML 2.0 implementation. Finally, authorization decisions returned by a PDP are handled by a default decision handler.

HERAS^{AF} PEP-PDP Communication

The standardized communication between the PEP and PDP is realized with the Spring Web Services Framework. To be standard-compliant, the XACML requests/responses are wrapped into the SAML 2.0 protocol and SOAP envelopes. The Web Service client is implemented as a PEP component and therefore can be plugged into the PEP to evaluate decision requests by a remote PDP.

5.2 Outlook

General

This chapter describes possible extensions and changes for a further development of the HERAS^{AF} PEP and HERAS^{AF} PEP-PDP Communication.

HERAS^{AF} PEP

Additional request information sources

Implementation of integration modules for further request information sources. For example:

- XML files
- Databases

Additional interceptor technologies

Implementation of intergration modules for further interceptor technologies. For example:

- AspectJ
- EJB interceptors
- Servlet filter

Decision Caching

A decision cache would make the HERAS^{AF} PEP faster. A received decision could be cached temporarily.

The HERAS^{AF} PEP could make a lookup in the decision cache before a decision request is sent to a PDP. With this enhancement a whole roundtrip for a request decision could be saved.

A major requirement for the cache would be the performance. The lookup operation and the caching of decision must be very fast. A question which arises for a temporary cache is the length a decision is valid in the cache.

Optimized implementation of Annotation Cache

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

Renaming of Annotation Cache

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

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

ReturnContext and InputContextOnly

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

HERAS^{AF} PEP-
PDP
Communication

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.

ReturnContext and InputContextOnly

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.

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.

Appendix A: General

1 Personal Reports

1.1 Daniel Regli

Everything began in an Enterprise Computing lecture in the end of the winter semester in 2007. Wolfgang Giersche, our tutor, slobbered over HERAS^{AF} and mentioned that he was looking for motivated and good students for the next bachelor theses. Yannick Winiger and I liked the idea of doing our thesis with the announced technologies and the challenges which a HERAS^{AF} project would bring. After Wolfgang Giersche and Florian Huonder told us more about the HERAS^{AF} project I knew that this was the thesis I wanted to realize. We wrote a letter of application for the thesis and we got accepted.

In my holidays, before the thesis started, I studied the basics of the Spring technology. Already in the first week we were confronted with a Spring Configuration and it was very helpful that this was not all new for me.

The project was very well guided from the beginning. We could attend some additional lectures (called Java Café) about used technologies in HERAS^{AF}. In addition to that the held weekly meetings gave us the possibilities to ask whenever we came across a problem. Mostly a good advice was given so we could go on with our thesis.

The project progression was straight forward. We sometimes had to investigate more time then scheduled because a lot was new for us but it was worth it. The only thing which I had to rack my brain over it was the maven dependency management which did not work well with the first Eclipse Plugin we used. I have never had some many ClassNotFoundExceptions in my life before. The problems could be solved after using additional tools to analyze the dependencies.

Another thing which brought us into time problem was the documentation. We could not keep the documentation up to date to the development progress. A lot of the documentation had to be done in our free time.

In the end we could achieve all the goals and I am very proud of it. I was very amazed when I saw in the end what we had created.

I did learn a lot apart from new technologies. Personally, I gained experiences in software engineering, teamwork, working agile and writing technical reports in English. In addition to that the agile approach with weekly meetings and goals definition was new for me and I really like it. Of course I gained experiences on the new technologies and Standards we used. Just to mention some; Spring (Core, AOP, Web Service), Maven2, JAXB, XACML, SAML and SOAP.

After we have been working with some of these technologies I cannot imagine how to work without them anymore. A good example is Maven2. I do not remember how we did the dependency management in earlier software projects.

I thank everybody who was involved with this bachelor thesis and who supported us in writing this bachelor thesis.

Especially I want to thank Florian Huonder, René Eggenschwiler and Wolfgang Giersche who assisted us in every situation we had got into. We had the chance to ask all the time through Skype, Mobile or personally face to face which is in my opinion not taken to be for granted. Without them the bachelor thesis would not have been such a success!

1.2 Yannick Winiger

It was very crucial to me, to finish my bachelor studies with an ambitious bachelor thesis. With the HERAS^{AF} project I found a topic which would provide a challenging in-depth look into today's software development and modern technologies. I gladly took this opportunity to extend my knowledge in developing java enterprise applications and learn from experienced tutors and coaches.

This thesis took a more Agile approach to software engineering, compared to the already known RUP. We worked in one week iterations and defining the goals for each week on the Friday before. This gave us the flexibility we needed in cases of problems or miscalculations.

For the build process we used Maven2, which is a requirement in many modern software engineering companies. Unfortunately we encountered many problems with a maven-integration plugin for Eclipse and conflicts with different versions of dependencies. Solving these problems cost us a lot of time, but I appreciate the hands-on experience with this tool.

The most challenging and interesting part of this thesis was the large amount of new technologies and standards, like XACML, SAML, Web Services and the Spring Framework. It required a lot of time to study and analysis them for a proper usage and implementation. In addition, I could learn how to design and implement a pluggable, extendable and adaptable framework and its components.

The documentation of the bachelor thesis and the developer's and user's guides were a large part of this thesis. Aside from the challenge to write everything in English, I had problems writing large amount of draft text in a short time, because I rewrote/changed/optimized every sentence, over and over again, until it seemed right to me. We tried to keep the documentation up-to-date during the thesis, but almost every week, other tasks and problems consumed our time. At the end of the first part of our thesis (the HERAS^{AF} PEP implementation) we decided to write the HERAS^{AF} PEP documentation on weekends to catch up.

Daniel Regli and I complemented one another perfectly during this thesis and I am proud that we could successfully realize a significant part of HERAS^{AF}.

Finally, I want to thank our tutor Wolfgang Giersche and our coaches Florian Huonder and René Eggenschwiler for their assistance and guidance throughout this thesis.

2 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

3 Bibliography

3.1 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)

3.2 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 Strebel:

HERAS^{AF}: Policy Administration Point
November 2007

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

[CS07PEP]

Massimo Cerqui, Sandro Strebel:

HERAS^{AF}: Interzeptoren für Spring AOP und AspectJ
July 2007

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

[DOH07DA] Sacha Dolski, Stefan Oberholzer, Florian Huonder:
HERAS^{AF}: XACML 2.0 Implementation
Hauptdokument
November 2007
Diploma Thesis at the University of Applied Sciences Rapperswil (HSR)

[DOH07DADev] Sacha Dolski, Stefan Oberholzer, Florian Huonder:
HERAS^{AF}: XACML 2.0 Implementation
Developer's Guide
November 2007
Diploma Thesis at the University of Applied Sciences Rapperswil (HSR)

[DOH07SA] Sacha Dolski, Stefan Oberholzer, Florian Huonder:
HERAS^{AF}: XACML PDP Web Service Endpoint
July 2007
Diploma Thesis at the University of Applied Sciences Rapperswil (HSR)

[Egg06] René Eggenschwiler:
HERAS^{AF} – Holistic Enterprise-Ready Application Security Architecture
Framework
Manageable policy-based access control for J2EE.
Mai 2006
Diploma Thesis at the University of Applied Sciences Rapperswil (HSR)

[Graf06] Yan Graf:
Distributed Access Control Policies – Enterprise Ready
December 2006
Diploma Thesis at the University of Applied Sciences Rapperswil (HSR)

3.3 Articles and Documents

[SunXACML05] Sun Microsystems
XACML: Access Control, Under Control.
November 2005
http://research.sun.com/spotlight/2005_11_01-XACML.html (06.06.08)

[PressInterop] OASIS Press Release

Eight Companies Demonstrate Interoperability of XACML

28 June 2007

<http://www.oasis-open.org/news/xacml-interop-2007-press-release.pdf>
(19.05.08)

[JBossXACML] User Guide for JBoss XACML
A Guide for Developers
<http://www.jboss.org/file-access/default/members/jbosssecurity/freezone/releases/jbossexacml/2.0.2.GA/guide/pdf/jbossexacml.pdf> (19.05.08)

3.4 Other resources

[CgLib] <http://sourceforge.net/projects/cglib>

[cargoPlugin] <http://cargo.codehaus.org/Maven2+plugin>

[EclEmma] <http://www.eclEmma.org/>

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

[Maven] <http://maven.apache.org/>

[OpenSource] <http://www.opensource.org/docs/definition.php>

[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/index.html>

[SpringSource] <http://www.springsource.com/products/springsecurity>

[SpringWS] <http://static.springframework.org/spring-ws/site/>

[SunXACML] <http://sunxacml.sourceforge.net>

[TestNG] <http://testng.org>

[ValueObject] <http://c2.com/cgi/wiki?ValueObject>

<i>[wikiDSS]</i>	http://en.wikipedia.org/wiki/Digital_Signature_Algorithm
<i>[wikiJavaAnnotation]</i>	http://en.wikipedia.org/wiki/Java_annotation
<i>[wikiJavaDoc]</i>	http://en.wikipedia.org/wiki/Javadoc
<i>[wikiJAXB]</i>	http://de.wikipedia.org/wiki/JAXB
<i>[wikiPKI]</i>	http://en.wikipedia.org/wiki/Public_key_infrastructure
<i>[wikiSAML]</i>	http://en.wikipedia.org/wiki/SAML
<i>[wikiSOAP]</i>	http://en.wikipedia.org/wiki/SOAP
<i>[wikiSPML]</i>	http://en.wikipedia.org/wiki/SPML
<i>[wikiSpring]</i>	http://en.wikipedia.org/wiki/Spring_Framework
<i>[wikiTestNG]</i>	http://en.wikipedia.org/wiki/Testng
<i>[WSS4J]</i>	http://ws.apache.org/wss4j/
<i>[XMLUnit]</i>	http://xmlunit.sourceforge.net/

Appendix B: Minutes of meeting

1 Meeting Information 29.02.08

Date: 29.02.08
Attendee: Wolfgang Giersche, René Eggenschwiler, Florian Huonder, Daniel Regli, Yannick Winiger
Location: Room 6.110, HSR
Start: 3:30 p.m.
End: 5.30 p.m.
Keeper of the minutes: Daniel Regli

1.1 Agenda items

Item	Assignment / Information
Week goals	<ul style="list-style-type: none"> Working PEP with heras-xacml implementation and AOP (greenbar on old tests, analysis and design of abstraction) Documentation (structure, project management)
Abstraction of design	<ul style="list-style-type: none"> Discussion of the design Coaches agreed on abstraction
Problems	<ul style="list-style-type: none"> DataTypesAttribute Map / ContextAndPolicyConfiguration not injected (NullPointerException) SAX-Parser error because of different versions of spring (1.x/2.x) Solution was found. Information how to solve it are on the wiki
Open issues	<ul style="list-style-type: none"> Inconsistent spring-config xml files - template or style guide? (example: spring-template.xml, heras-pep-Spring AOP.ctx.xml, Best Practices) Iteration schedule - we still need the picture of the whiteboard from the gettogether-meeting <ul style="list-style-type: none"> Was given to the project team during the meeting URNTToDataTypeConverter - we use this converter from the xacml-core (dependency to jaxb-impl.) - Ugly design? <ul style="list-style-type: none"> Design is acceptable, because there is not a strong dependency to it UML tool - we are using Jude Community for now, but it isn't able to create UML2.0 diagrams Transformers - AttributeValueType (getOtherAttributes()) - what goes in there? <ul style="list-style-type: none"> AttributeValues with additional attribute Value Part (e.g. <code><AttributeValue newAttrib="test"></code>)
Discussion	<ul style="list-style-type: none"> What happens in the code? Code walkthrough <ul style="list-style-type: none"> Current implementation contains many mistakes and bad design decisions
Goals for next week	<ul style="list-style-type: none"> Documentation first Part – Introduction <ul style="list-style-type: none"> There must be code to show to Wolfgang Giersche Analysis of current PEP in detail and creation of own new design from scratch for PEP Skeleton-Code for PEP

2 Meeting Information 07.03.08

Date: 07.03.08
Attendee: Wolfgang Giersche, René Eggenschwiler, Florian Huonder, Daniel Regli, Yannick Winiger
Location: Room 6.110, HSR
Start: 9:45 a.m.
End: 10.50 p.m.
Keeper of the minutes: Daniel Regli

2.1 Agenda items

Item	Assignment / Information
Week goals	<ul style="list-style-type: none"> • Analysis of old PEP and re-design of PEP • Analysis ok, new design is discussed because of few open issues • Skeleton implementation of new design • Documentation (introduction part)
Design discussion	<ul style="list-style-type: none"> • Design of PEP <ul style="list-style-type: none"> ○ General design points ○ InvocationContext ○ isAvailable Method ○ Obligations ○ DecisionManager • Discussion of all open issues and design together on blackboard
Open issues	<ul style="list-style-type: none"> • PDP Locator / PDP part needed? <ul style="list-style-type: none"> ○ is needed • static resolvers • RequestContextFactory – create method? <ul style="list-style-type: none"> ○ not possible (need factory for program against interface principle)
Goals for next week	<ul style="list-style-type: none"> • Implementation of PEP with tests • Documentation

3 Meeting Information 14.03.08

Date: 14.03.08
Attendee: René Eggenschwiler, Florian Huonder, Daniel Regli, Yannick Winiger
Location: Room 1.137, HSR
Start: 10:30 a.m.
End: 12.00 p.m.
Keeper of the minutes: Daniel Regli

3.1 Agenda items

Item	Assignment / Information
Week goals	<ul style="list-style-type: none"> • Implementation of new PEP Design
Problems	<ul style="list-style-type: none"> • Spring 2.5 + TestNG <ul style="list-style-type: none"> ○ Checking dependency with tools “maven dependency plugin” and retrying of exclusion • Annotation Problem • Considering implementation of own Utility class
Design Implementation	<ul style="list-style-type: none"> • Explaining additional points <ul style="list-style-type: none"> ○ Team agreed on ObligationHandler, ObligationProcessor parts • AttributeValue datatype • Documentation must point out the use of the datatype attribute and it's consequences
Goals for next two weeks	<ul style="list-style-type: none"> • Documentation (Introduction, Analysis, Design, Implementation parts) • Solving annotation problem • Solving Spring 2.5 + TestNG problem • Improve PEP implementation (Javadoc, etc.) • Improve Testing (Test szenarios, testing tests)

4 Meeting Information 28.03.08

Date: 28.03.08
Attendee: René Eggenschwiler, Florian Huonder, Daniel Regli, Yannick Winiger, Wolfgang Giersche
Location: Room 1.137, HSR
Start: 10:30 a.m.
End: 13.00 p.m.
Keeper of the minutes: Daniel Regli

4.1 Agenda items

Item	Assignment / Information
Week goals	<ul style="list-style-type: none"> • Documentation • Will be sent to review to Florian Huonder
Implementation of PEP	<ul style="list-style-type: none"> • Code walkthrough • DecisionManagerImpl should not be a singleton
Annotations look	<ul style="list-style-type: none"> • Discussion of our concept • Concepts of collecting values on methods (for annotation) • We need to define the way annotations are handled
Custom attribute types	<ul style="list-style-type: none"> • Solution with DynamicDataTypeAttribute
Maven Dependency Problem	<ul style="list-style-type: none"> • Discussion and help from team • Problems could be solved
Other issues	<ul style="list-style-type: none"> • Naming conventions • Documentation could be check into svn
Goals for the next week	<ul style="list-style-type: none"> • Implementation of new annotations and collecting algorithm • Naming conventions • Maven problems / migration to spring 2.5

5 Meeting Information 04.04.08

Date: 04.04.08
Attendee: René Eggenschwiler, Florian Huonder, Yannick Winiger, Wolfgang Giersche
Location: Room 1.201, HSR
Start: 13:30 p.m.
End: 15:00 p.m.
Keeper of the minutes: Yannick Winiger

5.1 Agenda items

Item	Assignment / Information
Annotations	<ul style="list-style-type: none"> Attributes are identified by their ID and Issuer. All annotations in the class hierarchy will be aggregated.
Testing & Mocks	<ul style="list-style-type: none"> Mocks should be copied to each module or to a test utilities module. Tests can be extracted into an extra test module per module (e.g. herasaf-pep-coretests)
Documentation	<ul style="list-style-type: none"> Old PEP Analysis should contain more information about what we did and reflect the time we invested. Old PEP Analysis should be written in an objective manner and don't contain words like "flaws", etc. The old PEP was developed in a different context and with different requirements, so there are no flaws... just concepts that don't work with the new requirements.
Annotation Algorithm & Caching	<ul style="list-style-type: none"> Maybe the complete resolved annotations could be cached (our value objects) instead of annotations. Algorithm & Cache should be flexible so that newly deployed classes get analyzed too.
Integration tests	<ul style="list-style-type: none"> (Maybe?) Use only a simple logical class structure with no real business case/scenario.

6 Meeting Information 11.04.08

Date: 11.04.08
Attendee: Florian Huonder, Yannick Winiger
Location: Room 1.201, HSR
Start: 15:15 p.m.
End: 16:30 p.m.
Keeper of the minutes: Daniel Regli

6.1 Agenda items

Item	Assignment / Information
Tests	<ul style="list-style-type: none"> • Different integration tests and module tests • If coverage of integration tests is good we don't need to make many module tests as well
Annotation @protected	<ul style="list-style-type: none"> • For now @protected only needs to work on classes level. • Florian Huonder will discuss this item with the other coaches
Naming Value Object	<ul style="list-style-type: none"> • When we change the value objects to e.g. ActionVO we need to describe why we came up with this naming
Meeting protocols	<ul style="list-style-type: none"> • Meeting protocols should be sent to everybody after the meeting so that everybody knows what was decided.
Goals for the next week	<ul style="list-style-type: none"> • Finish PEP totally (with documentation)

7 Meeting Information 18.04.08

Date: 18.04.08
Attendee: René Eggenschwiler, Florian Huonder, Yannick Winiger, Daniel Regli
Location: Room 1.201, HSR
Start: 10:00 p.m.
End: 11:00 p.m.
Keeper of the minutes: Daniel Regli

7.1 Agenda items

Item	Assignment / Information
PEP implementation	<ul style="list-style-type: none"> The PEP implementation is nearly completed. 80% of all test cases are passed correctly. The rest is scheduled to be completed in the afternoon.
Documentation	<ul style="list-style-type: none"> No more time is scheduled in the regular working hours to finish the documentation. Daniel Regli, Yannick Winiger will complete the documentation on the next 2 weekends.
Resource Content	<ul style="list-style-type: none"> Handling Resource Content is an open issue Extended symbols are converted to html tags (e.g. > to &gt;)
Synchronized Annotation Cache	<ul style="list-style-type: none"> This point should be mentioned as extension point in the documentation
Kick off meeting PDP-WS	<ul style="list-style-type: none"> Introduction from Florian Huonder was made.
Absence of coaches	<ul style="list-style-type: none"> René Eggenschwiler and Florian Huonder won't attend the weekly meeting of the 25.04.08.
Goals for next week	<ul style="list-style-type: none"> Thesis of the PDP Webservice should be read SAML 2.0 Profile of XACML 2.0 should be read and understood (or ready to ask questions) Spring-WS analysis and other possible technologies Architecture ideas how the Webservice could look like Where should the WS-Security be involved in the application flow Ideas to guarantee the standard-compliance

8 Meeting Information 25.04.08

Date: 25.04.08
Attendee: Wolfgang Giersche, Daniel Regli, Yannick Winiger
Location: Room 5.206, HSR
Start: 15:20 p.m.
End: 17:30 p.m.
Keeper of the minutes: Daniel Regli

8.1 Agenda items

Item	Assignment / Information
Documentation	<ul style="list-style-type: none"> • Documentation should be saved as .doc file extension (for compatibility reasons) • Next Friday the documentation will be sent to everybody for a documentation review, which we would like to receive latest a week after it was sent out.
Security aspects for webservice	<ul style="list-style-type: none"> • Security possibilities were outlined in the meeting. There is no need to investigate more time into this issue. • Write the implementation possibilities into the documentation (for WSS4J)
SAML Request/Response	<ul style="list-style-type: none"> • During the meeting the correct SAML XACML request and response syntax couldn't be made out. • Need to discuss the syntax with Florian Huonder
Goals for next week	<ul style="list-style-type: none"> • Investigation in the documentation of the webservice • Finish the prototype so that SOAP/SAML/XACML request/response is sent standard-compliant. • If enough time is left the team should begin to implement the webservice with the architecture discussed in the meeting. Add a new super project in SVN.

9 Meeting Information 02.05.08

Date: 02.05.08
Attendee: Florian Huonder, Daniel Regli, Yannick Winiger
Location: Room 1.201, HSR
Start: 10:00 p.m.
End: 10:50 p.m.
Keeper of the minutes: Daniel Regli

9.1 Agenda items

Item	Assignment / Information
Request/Response message	<ul style="list-style-type: none"> Both messages are standard-compliant Open issue is the Http / Soap header (see saml-bindings-2.0-os.pdf)
WSDL	<ul style="list-style-type: none"> Should be minimal and only contain top elements
Error Handling	<ul style="list-style-type: none"> Investigate time to properly handle errors on client/server side.
Goals for next week	<ul style="list-style-type: none"> Soap / Http Header as described in chapter Soap-Saml binding (Priority 1) Error Handling (Priority 1) Fix the WSDL file (Priority 1) Use a NamespacePrefixMapper to replace the default namespace prefixes (ns2,ns3,...) (Priority 2) Implement the Web Service with the architecture/design discussed in the last two meetings. (Priority 2)

10 Meeting Information 09.05.08

Date: 09.05.08
Attendee: Florian Huonder, René Eggenschwiler, Daniel Regli, Yannick Winiger
Location: Room 1.201, HSR
Start: 13:30 p.m.
End: 15:50 p.m.
Keeper of the minutes: Daniel Regli

10.1 Agenda items

Item	Assignment / Information
Schemas	<ul style="list-style-type: none"> • Schemas will be copied to herasaf.org server. • Schema locations will be communicated through René Eggenschwiler
PEP Exceptions	<ul style="list-style-type: none"> • New Exceptions: abstract HerasException, HerasSystemException, HerasBusinessException • Wrapped exception is the cause in the HerasException
Knowledge transfer	<ul style="list-style-type: none"> • Web Service Knowledge transfer to Christoph Zellweger, Patrick Neyer will be held on next Wednesday
Goals for next week	<ul style="list-style-type: none"> • PDP unit tests • PEP integration test • PEP WS unit tests • Documentation

11 Meeting Information 16.05.08

Date: 16.05.08
Attendee: Florian Huonder, René Eggenschwiler, Wolfgang Giersche, Josef Joller, Daniel Regli, Yannick Winiger
Location: Room 1.201, HSR
Start: 15:00 p.m.
End: 16:00 p.m.
Keeper of the minutes: Daniel Regli

11.1 Agenda items

Item	Assignment / Information
Assert	<ul style="list-style-type: none"> Do not use assert statements in the application because it needs to be enabled to work properly
PEP Exceptions	<ul style="list-style-type: none"> New Exceptions: HerasConfigurationException which is a HerasSystemException
SAML status code	<ul style="list-style-type: none"> Processing-error is always from Responder Syntax-error is always from Requester
Goals for next week	<ul style="list-style-type: none"> Documentation (Bachelor-Thesis, PEP Dev/User-Guide, PDP Context-WS Dev/User-Guide) Code/JavaDoc-Clean Up

12 Meeting Information 22.05.08

Date: 22.05.08
Attendee: Florian Huonder, René Eggenschwiler, Daniel Regli
Location: Room 1.207, HSR
Start: 10:00 p.m.
End: 10:30 p.m.
Keeper of the minutes: Daniel Regli

12.1 Agenda items

Item	Assignment / Information
Goals of thesis	<ul style="list-style-type: none"> goal definition in the hsr-avt tool is outdated Write the goals by ourselves and send it to review to Wolfgang Giersche
@Required	<ul style="list-style-type: none"> marks a property as being 'required-to-be-set' dependency to spring framework can be ignored
Goals for next week	<ul style="list-style-type: none"> Documentation (Bachelor-Thesis, PEP Dev/User-Guide, PDP Context-WS Dev/User-Guide) Code/JavaDoc-Clean Up Migration of ACEGI to Spring Security 2.x

13 Meeting Information 30.05.08

Date: 30.05.08
Attendee: Florian Huonder, René Eggenschwiler, Wolfgang Giersche, Daniel Regli
Location: Room 1.207, HSR
Start: 13:30 p.m.
End: 14:40 p.m.
Keeper of the minutes: Daniel Regli

13.1 Agenda items

Item	Assignment / Information
goal definition in the hsr-avt	<ul style="list-style-type: none"> The goal definition cannot be changed. We just translate it to English
Goals for next week	<ul style="list-style-type: none"> Finish the bachelor thesis