



HSR  
HOCHSCHULE FÜR TECHNIK  
RAPPERSWIL



**Studienarbeit**

# **HERAS<sup>AF</sup>: Interzeptoren für Spring AOP und AspectJ**

Abteilung Informatik  
Hochschule für Technik Rapperswil

Sommersemester 2007

Studenten:  
Verantwortlicher Dozent:

Sandro Strebel, Massimo Cerqui  
Wolfgang Giersche



# Inhaltsverzeichnis

## HERAS<sup>AF</sup> HSR Dokumentation

<b>TEIL I: EINFÜHRUNG</b> .....	<b>7</b>
<b>1 Abstract</b> .....	<b>8</b>
<b>2 Aufgabenstellung</b> .....	<b>9</b>
2.1 Zielsetzung.....	9
2.2 Vorgehen.....	9
2.3 Termine.....	10
<b>3 Management Summary</b> .....	<b>11</b>
3.1 Ausgangslage.....	11
3.2 Durchführung der Arbeit.....	12
3.3 Erreichte Ziele.....	13
3.4 Ausblick.....	13
<b>TEIL II: PROJEKTMANAGEMENT</b> .....	<b>14</b>
<b>1 Projektplanung</b> .....	<b>15</b>
1.1 Einführung.....	15
1.2 Projektübersicht.....	15
1.3 Projektplan.....	16
1.4 Risikomanagement.....	18
1.5 Qualitätsmassnahmen.....	18
1.6 Verwendete Technologie.....	19
1.7 Verwendete Standards.....	20
<b>2 Schlussbericht Projektmanagement</b> .....	<b>21</b>
2.1 Zeitauswertung.....	21
<b>TEIL III: TECHNISCHER BERICHT</b> .....	<b>24</b>
<b>1 Übersicht</b> .....	<b>25</b>
<b>2 Analyse</b> .....	<b>25</b>
2.1 XACML Request.....	25
2.2 Domain.....	29
2.3 Sequenz Diagramm.....	30
<b>3 Anforderungen</b> .....	<b>31</b>
3.1 Funktionale.....	31
3.2 Nichtfunktionale.....	31
<b>4 Schlussbericht</b> .....	<b>32</b>
4.1 Zusammenfassung.....	32
4.2 Erreichte Ziele.....	32
4.3 Ausblick.....	32



<b>TEIL IV: ANHANG A</b> .....	<b>33</b>
<b>5 Persönliche Berichte</b> .....	<b>34</b>
5.1 Sandro Strebel .....	34
5.2 Massimo Cerqui .....	34
<b>6 Glossar</b> .....	<b>35</b>
<b>7 Referenzen</b> .....	<b>35</b>



# HERAS<sup>AF</sup> User Guide

<b>PART I: INTRODUCTION .....</b>	<b>3</b>
<b>1 Introduction .....</b>	<b>4</b>
1.1 Definition .....	4
1.2 Glossary .....	5
1.3 Overview .....	6
<b>PART II: SPRING CONFIGURATION.....</b>	<b>7</b>
<b>1 Spring Configuration .....</b>	<b>8</b>
1.1 application.ctx.xml .....	9
1.2 heras-security.ctx.cml .....	9
1.3 herasp-pep.ctx.xml .....	9
1.4 acegisecurity.ctx.xml .....	11
1.5 heras-pdp.ctx.xml .....	12
1.6 heras-springaop.ctx.xml .....	13
1.7 request-factory ctx.xml .....	14
1.8 attribute-config.xml .....	15
<b>PART II: ANNOTATIONS .....</b>	<b>16</b>
<b>2 Annotation Overview .....</b>	<b>17</b>
2.1 @Protected .....	17
2.2 @Attribute .....	17
2.3 @Attributes .....	18
2.4 @RequestCtxOnly .....	19
2.5 @ReturnContext .....	20
<b>PART II: TUTORIALS .....</b>	<b>21</b>
<b>3 HERAS<sup>AF</sup> with Spring AOP and ACEGI .....</b>	<b>22</b>
3.1 Introduction .....	22
3.2 STEP 1 - Environment setup .....	22
3.3 STEP 2 - Spring configuration .....	23
3.4 STEP 3 - Decorating the Interface with annotations.....	27
<b>4 HERAS<sup>AF</sup> with AspectJ and ACEGI .....</b>	<b>29</b>
4.1 Introduction .....	29
4.2 STEP 1 - Environment setup .....	29
4.3 STEP 2 - Spring configuration .....	30
4.4 STEP 3 - Decorating the Interface with annotations.....	34



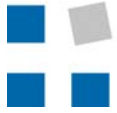
## HERAS<sup>AF</sup> Developer Guide

<b>PART I: INTRODUCTION .....</b>	<b>3</b>
<b>1 Introduction .....</b>	<b>4</b>
1.1 Definition .....	4
1.2 Glossary .....	5
1.3 Architectural Overview .....	6
1.4 Modules.....	7
<b>PART II: A HERAS<sup>AF</sup> SHORT STORY .....</b>	<b>8</b>
<b>2 HERAS<sup>AF</sup> Short Story .....</b>	<b>9</b>
<b>PART III: ARCHITECTURE .....</b>	<b>13</b>
<b>3 Logical Overview .....</b>	<b>14</b>
3.1 Overview .....	14
3.2 Sequence Diagram .....	15
<b>4 Logical View.....</b>	<b>16</b>
4.1 Overview .....	16
4.2 Package org.herasaf.pep.core .....	17
4.3 Package org.herasaf.pep.annotations .....	18
4.4 Package org.herasaf.pep.integration.springaop .....	19
4.5 Package org.herasaf.pep.integration.aspectj .....	20
4.6 Package org.herasaf.pep.resolvers .....	21
4.7 Package org.herasaf.pep.transformers .....	22
<b>5 Architectural Decisions .....</b>	<b>23</b>
5.2 HerasAttributeConfiguration.....	23
5.3 HerasAttribute .....	23
5.4 Multiple Resources and Subjects.....	24



## HERAS<sup>AF</sup> Sitzungsprotokolle

<b>1</b>	<b>Sitzungsprotokolle</b>	<b>3</b>
1.1	Kickoff Meeting vom 02.04.2007	3
1.2	Status Meeting Iteration 2 vom 17.04.2007	4
1.3	Status Meeting Iteration 2 vom 8.05.2007	5
1.4	Status Meeting Iteration 3 vom 11.05.2007	6
1.5	Status Meeting Iteration 4 vom 15.06.2007	8



# Teil I: Einführung



# 1 Abstract

Heras<sup>AF</sup> ist ein Open Source Projekt, welche das Ziel hat, eine zentral verwaltbare unternehmenstaugliche Autorisierungslösung zu realisieren. Heras<sup>AF</sup> baut auf frei verfügbaren, etablierten und zukunftssträchtigen Technologien und Standards auf. Dabei stehen Interoperabilität, Erweiterbarkeit und Austauschbarkeit der integrierten Komponenten im Vordergrund.

In dieser Studienarbeit wurde eine Bibliothek erstellt, welche es erlaubt beliebige Methodenaufrufe von Applikationen abzufangen, sicherheitsrelevante Informationen zu sammeln und eine Entscheidungsanfrage in Form eines standradkonformen SAML/XACML Requests an einen Wohldefinierten Policy Decision Point (PDP) zu senden. Damit wurde die Grundlage für eine zentralisierte Sicherheitsverwaltung in J2EE Landschaften ermöglicht.

Das abfangen der Methodenaufrufe wurde in einer ersten Phase mit SpringAOP als Interzeptor Technologie gelöst und implementiert.  
In einer zweiten Phase wurden die Methodenaufrufe dann mithilfe von AspectJ abgefangen.

Um sicherheitsrelevanten Informationen zu definieren wurde ein Set von Java 1.5 Annotations entworfen, welche es ermöglichen den Methodenaufruf auf einen SAML/XACML Request abzubilden.



## 2 Aufgabenstellung

### 2.1 Zielsetzung

Es ist eine Bibliothek zu erstellen, die es erlaubt in beliebigen Applikationen Methodenaufrufe abzufangen, sicherheitsrelevante Informationen zu sammeln und eine Entscheidungsanfrage in Form eines standardkonformen SAML/XACML Requests an einen Wohldefinierten Policy Decision Point (PDP) zu senden. Damit wird die Grundlage für eine zentralisierte Sicherheitsverwaltung in J2EE Landschaften ermöglicht.

### 2.2 Vorgehen

Die Studienarbeit ist grob in vier Phasen unterteilt.

#### **Phase 1: XACML Request**

In der ersten Phase geht es darum die existierende Projektstruktur kennen zu lernen und die Machbarkeit der Arbeit zu eruieren. Ebenfalls wird die XACML Spezifikation analysiert um den Ablauf sowie die verschiedenen Attribute (Subject, Resource, Action, Environment) zu verstehen. Um frühzeitig Reserven einzuplanen wird eine Risikoanalyse aufgesetzt. In einem Analyse Dokument wird der Kontext der verschiedenen Attribute definiert und dokumentiert.

#### **Phase 2: Spring AOP**

Das Ziel von Phase 2 ist es, einen funktionierenden vertikalen Prototyp mittels SpringAOP zu erstellen. Um das Ziel zu erreichen wird zuerst der Rahmen der Studie grob abgesteckt und der bestehende Code analysiert. Danach wird das Design eruiert und entsprechend umgesetzt.

#### **Phase 3: AspectJ**

Nachdem der vertikale Prototyp mit Spring AOP realisiert ist, soll als Interzeptor Technologie AspectJ verwendet werden. Wenn das Design erstellt und umgesetzt ist, wird ein Refactoring auf das komplette Projekt angewendet.

#### **Phase 4: Integration**

In der letzten Phase werden Integrationstests geschrieben und die Dokumentation überarbeitet und soweit abgeschlossen.



## 2.3 Termine

02.04.2007            Beginn der Studienarbeit

- 06.07.2007
- Abgabe der Kurzbeschreibungen an G. Dittli, [gdittli@hsr.ch](mailto:gdittli@hsr.ch)
  - Abgabe des Berichtes an den Betreuer bis 17.00 Uhr.

Ort, Datum:

---

Verantwortlicher Dozent, Wolfgang Giersche:

---



## 3 Management Summary

### 3.1 Ausgangslage

<i>Motivation</i>	<p>Zentrale Policyverwaltung und Zugriffssteuerung ist noch nicht weit verbreitet. Die meisten kommerziellen und nicht gewerblichen Anwendungen kommen mit ihrem eigenen Benutzerspeicher und ihren Zugriffssteuerungsmechanismen daher. Quelloffene, standardisierte Lösungen gibt es noch keine. Häufig führt diese Situation zu unvermeidlichen zusätzlichen Bemühungen, Kosten und Sicherheitsgefahren. Es kann sogar zu Inkonsistenzen kommen, in denen unterschiedliche Anwendungen, unterschiedliche Niveaus des Zugangs zur gleichen tatsächlichen Ressource gewähren. Dies kann auch geschehen, wenn eine schützenswerte Ressource einfach übersehen wird und somit nicht geschützt wird. Aber auch wenn Synchronisationen zwischen unterschiedlichen Anwendungen zu überflüssigen Policyinformationen führt. Daher ist eine konsistente, zentrale Policyverwaltung und Zugriffssteuerung einer der ersten Grundsteine, wenn es darum geht Gefahr, Kosten und Bemühungen zu verringern. Heras<sup>AF</sup> ist der Versuch, solch eine Lösung nur mit frei verfügbaren, quelloffenen Bestandteilen zu realisieren.</p>
<i>Ziel von HERAS<sup>AF</sup></i>	<ul style="list-style-type: none"> <li>• HERAS<sup>AF</sup> soll unternehmenstauglich sein</li> <li>• HERAS<sup>AF</sup> soll einen ganzheitlichen Ansatz des Autorisierungsprozesses verfolgen</li> <li>• Mit „ganzheitlich“ ist aber auch gemeint, dass alle Aspekte des Entwerfens und der Wartung von Policies durch nicht-technisches Personal im Frameworkdesign beachtet werden, speziell im Policy Administration Point.</li> <li>• HERAS<sup>AF</sup> soll sich leicht in existierende IT-Infrastruktur integrieren lassen</li> <li>• HERAS<sup>AF</sup> soll quelloffen sein</li> </ul>
<i>Ziel dieser Studienarbeit</i>	<p>Es ist eine Bibliothek zu erstellen, die es erlaubt in beliebigen Applikationen Methodenaufrufe abzufangen, sicherheitsrelevante Informationen zu sammeln und eine Entscheidungsanfrage in Form eines standardkonformen SAML/XACML Requests an einen Wohldefinierten Policy Decision Point zu senden. Damit wird die Grundlage für eine zentralisierte Sicherheitsverwaltung in J2EE Landschaften ermöglicht.</p>



## 3.2 Durchführung der Arbeit

<i>Involvierte Personen</i>	<b>Person</b>	<b>Funktion</b>
	Wolfgang Giersche	Verantwortlicher Dozent
	René Eggenschwiler	Betreuer der Arbeit
	Sandro Strebel	Architekt, Entwickler
	Massimo Cerqui	Qualitätsmanager, Entwickler
<i>Vorarbeit</i>	<p>Im Vorfeld der Arbeit musste sich in die XACML 2.0 [XACML20SPEC] Spezifikation eingearbeitet werden.</p> <p>Das bereits bestehende HERAS<sup>AF</sup> – Projekt wurde analysiert und es wurde versucht die einzelnen Projekt einzuordnen. Das Grundgerüst von HERAS<sup>AF</sup> wurde in den Diplomarbeiten von Yan Graf [DAYG] und René Eggenschwiler [DARE] erarbeitet.</p> <p>Die Vorarbeit erwies sich als relativ aufwändig und kompliziert, da es viele neue Technologien zu erlernen gab und da das komplette HERAS<sup>AF</sup> – Projekt sehr abstrakt aufgesetzt war.</p>	
<i>Vorgehen</i>	<p>Die Realisierung der Arbeit wurde in sinnvollen Teilschritten durchgeführt.</p> <p><b>XACML Request Analyse</b></p> <p>In dieser Phase ging es darum die XACML Spezifikation zu analysieren um den Ablauf sowie die verschiedenen Attribute (Subject, Resource, Action, Environment) zu verstehen. Danach wurde analysiert wie sich Methodenaufrufe auf einen XACML Request abbilden lassen.</p> <p><b>SpringAOP</b></p> <p>Für das Abfangen der Methodenaufrufe wurde ein vertikaler Prototyp mit SpringAOP als Interzeptor Technologie verwendet.</p> <p><b>AspectJ</b></p> <p>Nachdem die Architektur für die Integration mit SpringAOP ausgelegt war, wurde die Integration von AspectJ als Alternative zu SpringAOP implementiert.</p> <p><b>Refactoring</b></p> <p>Nachdem die Integration von AspectJ abgeschlossen war, wurde ein gezieltes Refactoring auf doppelten Code durchgeführt. Der Code wurde vereinfacht und Abläufe wurden optimiert.</p>	



### 3.3 Erreichte Ziele

*Ergebnisse* In dieser Studienarbeit entstand eine Bibliothek, welche es erlaubt beliebige Methodenaufrufe abzufangen, sicherheitsrelevante Informationen zu sammeln und eine Entscheidungsanfrage in Form eines standardkonformen SAML/XACML Requests an einen Wohldefinierten Policy Decision Point (PDP) zu senden. Damit wurde die Grundlage für eine zentralisierte Sicherheitsverwaltung in J2EE Landschaften ermöglicht.

*Gelerntes*

#### **Gelerntes**

- Wertvolle Erfahrungen im Bereich Software Engineering
- Neue Technologien wie:
  - Spring AOP
  - AspectJ
  - ACEGI Security
  - XACML
- Integration von bestehenden Softwarekomponenten
- Zusammenarbeit mit mehreren Teams

### 3.4 Ausblick

*Folgearbeit*

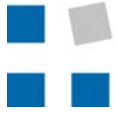
In einer möglichen Folgearbeit könnten weitere Interzeptoren Technogien (wie zum Beispiel EJB 3.0) integriert werden.

Um die Antwort des PDP gemäss XACML 2.0 Spezifikation auszuwerten und zu interpretieren müssen die verschiedenen Ausprägungen des PEP konfiguriert werden können.

Diese umfassen:

- Base PEP
- Deny-biased PEP
- Permit-biased PEP

Eine andere Erweiterung könnte die Integration des Resource Content gemäss XACML 2.0 Spezifikation sein.



## Teil II: Projektmanagement



# 1 Projektplanung

## 1.1 Einführung

*Zweck* Die Projektplanung soll als Leitfaden für das Projekt "HERAS<sup>AF</sup>: Interzeptoren Interzeptoren für Spring AOP und AspectJ" dienen. Sie enthält unter anderem die Zeitplanung, eine grobe Aufteilung der Arbeit, Qualitätsmassnahmen und eine Übersicht über die verwendeten Technologien.

## 1.2 Projektübersicht

*Ziele* Die Ziele der Arbeit sind in der Aufgabenstellung festgehalten.

### *Organisationsstruktur*

#### **Sandro Strebel (STS)**

#### **Verantwortung**

Architekt, Entwickler

- Kontrolle Projektfortschritt
- Kontrolle Design

#### **Massimo Cerqui (MAC)**

#### **Verantwortung**

Qualitätsmanager, Entwickler

- Sicherstellung Qualitätsmanagement
- Kontrolle Analyse

### *Externe Schnittstellen*

#### **Wolfgang Giersche (GIW)**

#### **Kontakt**

Verantwortlicher Dozent

Email: [wgiersch@hsr.ch](mailto:wgiersch@hsr.ch)  
Tel: 079 769 98 12

#### **René Eggenschwiler (EGR)**

#### **Kontakt**

Projektbetreuer

Email: [reggenc@hsr.ch](mailto:reggenc@hsr.ch)  
[rene.eggenschwiler@bluewin.ch](mailto:rene.eggenschwiler@bluewin.ch)  
Tel: 078 629 08 09

### *Termine*

Folgende Termine sind von der HSR vorgegeben:

- Projektstart 02.04.2007
- Abgabe 06.07.2007

Zudem findet während der gesamten Projektdauer wöchentlich eine Besprechung mit dem Betreuer statt.

Jede zweite Woche wird dem Experten der aktuelle Projektstand präsentiert und mit ihm das weitere Vorgehen besprochen.



## 1.3 Projektplan

### *Iterations- planung*

<b>Iteration 1</b>	<b>Start</b>	<b>Ende</b>
	02.04.2007	13.04.2007
<b>Iteration 2</b>	<b>Start</b>	<b>Ende</b>
	16.04.2007	27.04.2007
<b>Iteration 3</b>	<b>Start</b>	<b>Ende</b>
	27.04.2007	01.06.2007
<b>Iteration 4</b>	<b>Start</b>	<b>Ende</b>
	02.06.2007	15.06.2007
<b>Iteration 5</b>	<b>Start</b>	<b>Ende</b>
	16.06.2007	06.07.2007

### *Meilensteine*

<b>MS1 End Iteration 1</b>	<b>Datum</b>
Prototyp Design	13.04.2007
<b>MS2 End Iteration 2</b>	<b>Datum</b>
Vertikaler Schnitt durch das Design inkl. 3 - 4 Beispiele annotierter Business Cases	27.04.2007
<b>MS2 End Iteration 3</b>	<b>Datum</b>
Refinement der Dokumente und Refactoring des Codes	01.06.2007
<b>MS2 End Iteration 4</b>	<b>Datum</b>
Heras <sup>AF</sup> PEP mit Hilfe von AspectJ implementieren	15.06.2007
<b>MS2 End Iteration 5</b>	<b>Datum</b>
Code Refactoring, Dokumente vervollständigen, Abgabe der Arbeit.	06.07.2007



### 1.3.1 Featureliste

<i>Iteration 2</i>	<b>Beschreibung</b>	<b>Erfasst</b>	<b>Stand</b>
	Vertikaler Schnitt durch Architektur mit 3-4 Annotierten Business Cases	13.04.2007	Erledigt
	Mehrere Resolvers inkl. Dynamic Resolver	13.04.2007	Erledigt
	Mehrere Transformers inkl. Dynamic Transformer	13.04.2007	Erledigt
	Definition und Konfiguration von möglichen Attributen	20.04.2007	Erledigt
<i>Iteration 3</i>	<b>Beschreibung</b>	<b>Erfasst</b>	<b>Stand</b>
	SAML PDP Erweiterung (RequestCtxOnly und ReturnCtx)	11.05.2007	Erledigt
	SpringAOP ClassFilter auf @Protected Annotations	11.05.2007	Erledigt
	Developer Guide (Short Story)	11.05.2007	Erledigt
	User Guide	11.05.2007	Erledigt
<i>Iteration 4</i>	<b>Beschreibung</b>	<b>Erfasst</b>	<b>Stand</b>
	AspectJ Integration	01.06.2007	Erledigt
	Refactoring HERAS-PEP	01.06.2007	Erledigt
<i>Iteration 5</i>	<b>Beschreibung</b>	<b>Erfasst</b>	<b>Stand</b>
	Refinement User Guide	15.06.2007	Erledigt
	Refinement Developer Guide	15.06.2007	Erledigt
	Refactoring Code	15.06.2007	Erledigt



## 1.4 Risikomanagement

Um frühzeitig Risiken zu erkennen und Reserven zu planen wird das Risikomanagement fortlaufend aktualisiert und die Risiken neu bewertet.

Risiko	Auswirkungen	Massnahmen	Kosten der Massnahmen in Personenstunden	Maximaler Schaden in Personenstunden	Wahrscheinlichkeit des Eintreffens in Prozent	Gewichteter Schaden in Personenstunden	Priorität
R01 Komplexität Mapping	Verzögerung des Projektes	frühzeitig Prototyp erstellen	40	16	5%	1	hoch
R02 Komplexität Authentication Strategies	Verzögerung des Projektes	frühzeitig Prototyp erstellen	40	40	10%	4	hoch
R03 Ausfall SVN Server, Infrastruktur	Aktueller Stand geht verloren	lokale Kopie verwenden, neu programmieren	8	20	20%	4	mittel
R04 Ausfälle durch Krankheit	Verzögerung des Projektes			8	10%	1	niedrig
R05 Team-Probleme	Schlechte Kommunikation und uneffiziente Arbeit	Sozialer Aspekt berücksichtigen	8	40	5%	2	niedrig
Total Stunden in Arbeitspaketen enthalten			96	124			
Total Rückstellungen						12	

## 1.5 Qualitätsmassnahmen

<i>Dokumentation</i>	Die Dokumentation des ganzen Projekts, vom Projektantrag bis zur Code-Dokumentation, hat einen hohen Stellenwert und wird fortlaufend aktualisiert.
Vier Augen Prinzip	Alle Dokumente sollen von mindestens zwei Personen gelesen werden, damit die Qualität gewährleistet werden kann.
Einsatz von Versionisierungssystem	Das gesamte Projekt wird in seiner Datenstruktur in SVN abgebildet, damit alle am Projekt Beteiligten die gleiche Sicht auf das Projekt haben.
Laufende Tests	Tests die einmal definiert worden sind, sollen archiviert und laufend für alle Tests einbezogen werden.
Automation	Jeder Code der eingecheckt wird, soll Kompilier- und Lauffähig sein. Dies soll durch Automatisierungsskripte ermöglicht werden. Auch das Testen soll durch Skripte automatisiert werden.



## 1.6 Verwendete Technologie

- Spring 2.0* Das Spring Framework ist ein Open Source Applikationsframework für die Java-Plattform. Ziel des Spring Frameworks (kurz Spring) ist es, die Entwicklung mit Java/JavaEE zu vereinfachen und gute Programmierpraktiken zu fördern.
- Über Dependency Injection werden den Objekten die abhängigen Objekte bzw. Ressourcen zugewiesen. Sie müssen sie nicht selbst suchen.
- ACEGI* Acegi Security (<http://www.acegisecurity.org>) ist ein quelloffenes Projekt, welches umfangreiche Authentisierungs- und Autorisierungsdienste für Unternehmensanwendungen zur Verfügung stellt, welche auf dem Spring Framework basieren.
- Acegi kann mit einer Vielzahl von Authentisierungsanbietern verwendet werden und kann Webanfragen und Methodenaufrufe autorisieren.
- Acegi liefert einen integrierten Sicherheitsansatz über die verschiedenen Ziele und bietet auch Zugangskontrolllisten (ACL) an, um einzelne Objekte zu sichern.
- Auf dem Implementierungsniveau wird Acegi durch Spring's Inversion of Control gehandhabt.
- Spring AOP* Spring AOP ist ein Programmierparadigma, um räumlich getrennte Programmbestandteile (zum Beispiel Funktionen) von zentraler Stelle mit bestimmten Eigenschaften (zum Beispiel die Protokollierung von Aufrufen) auszustatten. Dazu werden so genannte Aspekte in eigenen Dateien definiert und frühestens zur Übersetzungszeit automatisch in den Programmcode eingefügt.
- AspectJ* AspectJ ist eine aspekt-orientierte Erweiterung von Java. Ein AspectJ-Compiler bindet Aspekte in normalen Java-Bytecode ein, um das Event-basierte System zu implementieren. Aspekte sind in einer Kombination von Java und AspectJ geschrieben und durch eine zusätzlichen Vor-Klasse in Java eingebunden, um Standard Java Bytecode erzeugen zu können und so AspectJ mit Java kompatibel zu machen.
- Java 1.5 Annotations* Als Annotation wird ein Sprachelement bezeichnet, das die Einbindung von Metadaten in den Quelltext erlaubt. Dieses Element wurde im JSR 175 festgelegt und mit der Version Java 5.0 eingeführt.
- Annotationen werden durch ein @-Zeichen gefolgt vom Namen der Annotation gekennzeichnet. Optional kann eine kommagetrennte Parameterliste folgen, die in runden Klammern eingefasst wird.
- TestNG 5.2* Bei TestNG handelt es sich um ein Framework zum Testen von Java-Applikationen, welches Vorteile gegenüber JUnit bieten soll. Für die Konfiguration kommen JDK 1.5-Annotations zum Einsatz, für JDK 1.4 gibt es eine alternative Notation mit Javadoc-Tags. Es braucht kein Interface implementiert zu werden, und es können flexibel unterschiedliche Gruppierungen von Tests definiert werden. Das Ganze macht die Nutzung allerdings komplizierter und lohnt sich deshalb wohl nur für besonders umfangreiche Test-Sammlungen mit dem Bedarf für unterschiedliche Zusammenstellungen.



## 1.7 Verwendete Standards

*XACML 2.0* eXtensible Access Control Markup Language (XACML) ist ein XML-Schema, das die Darstellung und Verarbeitung von Autorisierungs-Policies standardisiert. Der vom OASIS-Konsortium definierte Standard dient dazu, Regeln aufzustellen, durch deren Auswertung der Zugriff von Subjekten auf Ressourcen eines Systems gesteuert werden kann. Version 2.0 wurde von der Standardisierungs-Organisation OASIS im Februar 2005 ratifiziert. Derzeit ist Version 3.0 in Arbeit. Wesentliche Neuerungen sind die Möglichkeit Rechte zu delegieren und erweiterte Möglichkeiten die Anwendbarkeit von Zugriffsregeln zu spezifizieren.



## 2 Schlussbericht Projektmanagement

### Planung

Die zu Beginn erstellte Planung konnte sehr gut eingehalten werden. Die budgetierten Stunden wurden nur wenig überschritten.

Durch die frühzeitig erstellte Rissikonalanalyse konnte Reserven geschaffen werden, welche in Problemsituationen ausreichten um die Projektplanung und die Termine nicht zu gefährden.

### 2.1 Zeitauswertung

#### Einführung

Die Studienarbeit dauert 14 Wochen. Während diesen 14 Wochen wurde die Arbeitszeit mit dem Zeiterfassungstool Achievo erfasst.

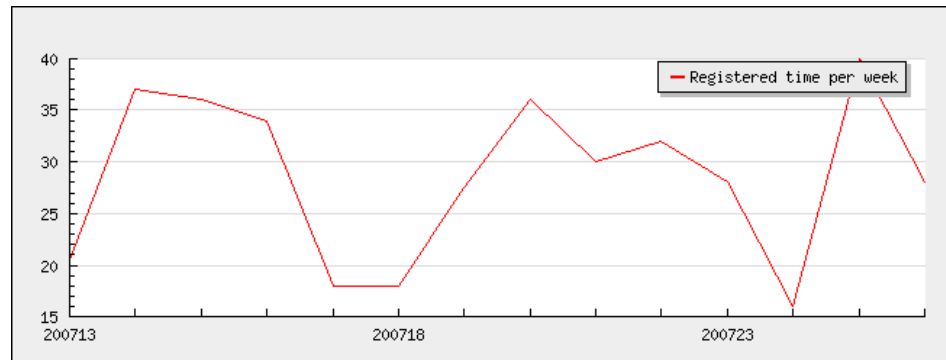
Eine erfolgreiche Studienarbeit erhält 8 ETCS Punkte (1 ETCS Punkt entspricht einer Arbeitsleistung von ca. 25 – 30 h).

Die Studienarbeit ist deshalb mit 200 bis 240 Stunden pro Person geplant. Für das Projekt entspricht unsere Planung deshalb einem Mittelwert von 420 Mannstunden.

#### Überblick

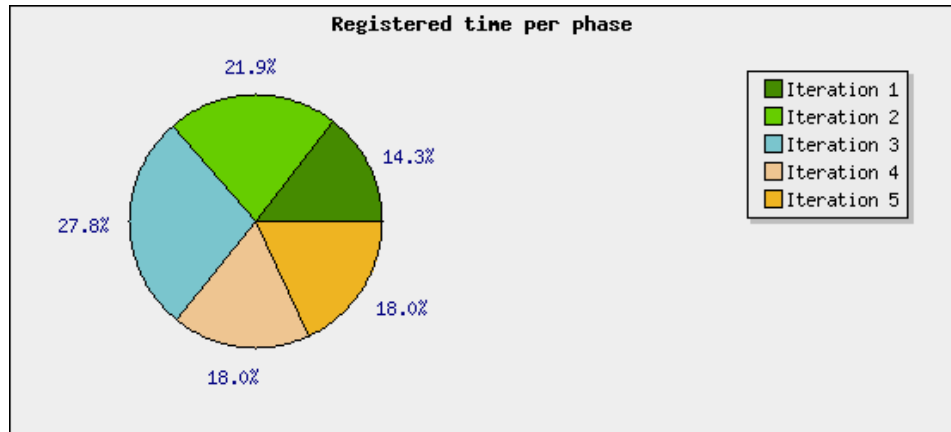
	Geplant	Effektive gearbeitet
Projektstunden Team:	420	444.30
Stunden pro Woche:	15	15.8

#### Zeit pro Woche



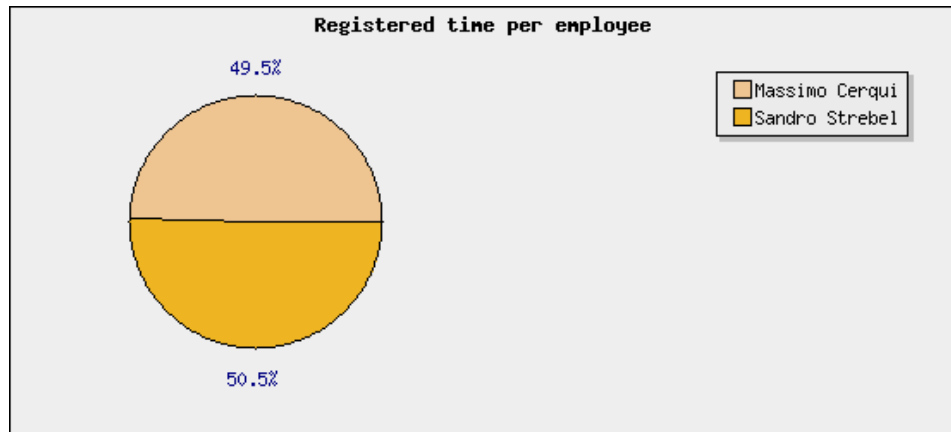


*Zeit pro  
Iteration*



Phase	Zeit in Stunden	Prozent
Iteration 1	57:30	14.34%
Iteration 2	98:00	21.95%
Iteration 3	125:00	27.81%
Iteration 4	76:00	17.96%
Iteration 5	88:00	17.96%
<b>Total</b>	<b>444:30</b>	<b>100.00%</b>

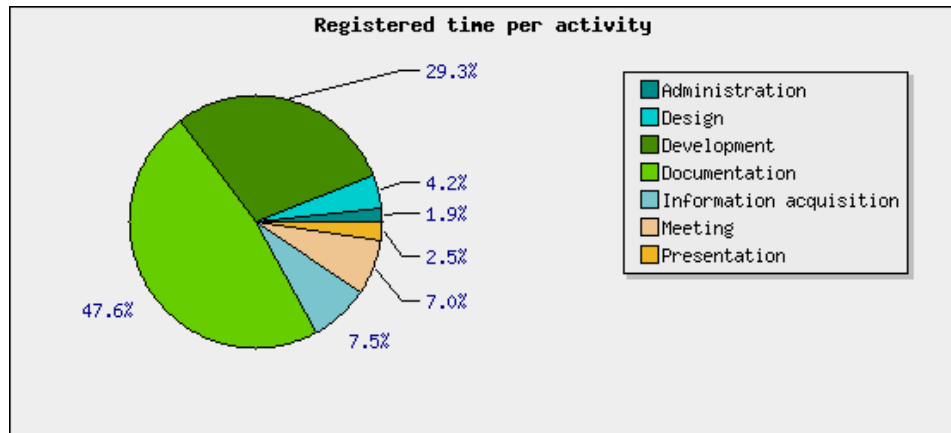
*Zeit pro  
Teammitglied*



Name	Zeit	Prozent
Massimo Cerqui	220:0	49.50%
Sandro Strebel	224:0	50.50%
<b>Total</b>	<b>444:30</b>	<b>100.00%</b>

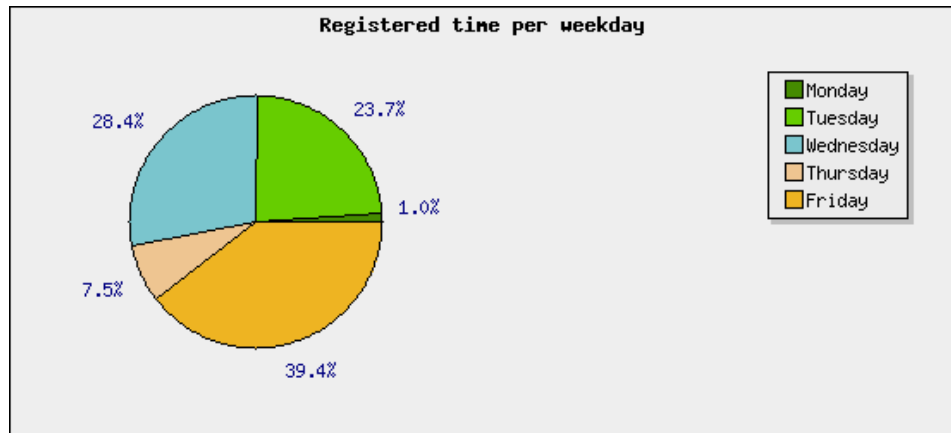


Zeit pro Aktivität

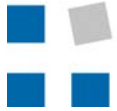


Aktivität	Zeit	Prozent
Administration	13:30	1.87%
Design	17:00	4.24%
Development	139:00	29.30%
Documentation	207:00	47.63%
Information acquisition	30:00	7.48%
Meeting	28:00	6.98%
Presentation	10:00	2.49%
<b>Total</b>	<b>444:30</b>	<b>100.00%</b>

Zeit pro Wochentag



Tag	Zeit	Prozent
Montag	4:00	1.00%
Dienstag	122:30	23.69%
Mittwoch	114:00	28.43%
Donnerstag	30:00	7.48%
Freitag	174:00	39.40%
<b>Total</b>	<b>444:30</b>	<b>100.00%</b>



## Teil III: Technischer Bericht



# 1 Übersicht

*Inhalt* Der Technische Bericht beinhaltet die Analyse und die Anforderungen der einzelnen Projektphasen. Dabei werden einzelne Ansätze und Möglichkeiten aufgezeigt.

*Aufbau* Im Analyse Teil werden die vier Bestandteile eines XACML Requests untersucht und es wird versucht ein geeignetes Mapping für Methodenaufrufe zu finden.

## 2 Analyse

### 2.1 XACML Request

#### 2.1.1 Subject

*Subject* Ein Subject definiert Informationen über das Subjekt eines Request Contexts. Ein Request Context kann mehrere Subjects besitzen wobei jedes Subject wieder mehrere Attribute besitzen kann.

Jedes Subject gehört einer Kategorie an. Eine Subject Kategorie beschreibt die Rolle welche das Subject in der Erstellung eines Decision Requests spielt. Der Default Value einer Subject Category ist:  
`oasis:names:tc:xacml:1.0:subject-category:access-subject.`

Ein Subject könnte zum Beispiel den Benutzer repräsentieren welcher einen Request initiiert hat. Ein anderes Subject könnte den ausführbaren Code beschreiben, welcher für die Erstellung des Requests zuständig ist. Und wieder ein anderes Subject könnte die Maschine repräsentieren auf welcher die Anwendung ausgeführt wird.

Die folgende Tabelle zeigt mögliche Interpretationen eines Subjects und beschreibt dessen Format, sowie einen Lösungsansatz wie auf die Information zugegriffen werden könnte.

Interpretationen des Subject welche erst zur Laufzeit auswertbar sind, werden mit einem „x“ in der Spalte Runtime vermerkt.

Subject	Format	Zugriff	Runtime
Human User	Email RFC822	Threadlocal, Session Context	x
Machine	IP, Hostname	InetAddress.getLocalHost()	x
Executable Code	Request Factory	?	x
Recipient of Resource	Email RFC822	?	?



*Auswertung* Die Analyse hat ergeben, dass sämtliche Interpretationen des Subjects nur zur Laufzeit ausgewertet werden können.

Für das Format des Subjects scheint uns Email RFC822 am besten geeignet, da dieses Format allgemein interpretierbar und kontrollierbar ist.

*ACEGI* Für die Interpretation eines Subjects zur Laufzeit könnte man ACEGI verwenden.

ACEGI Security ist eine mächtige, flexible Sicherheitslösung für Unternehmensanwendungen. ACEGI Security lässt sich sehr einfach in Anwendungen welche den Spring Application Context verwenden integrieren.

Das Subject kann danach über das ACEGI Authentication Objekt angefordert werden, welches im SecurityContextHolder zur Verfügung gestellt wird.

## 2.1.2 Resource

*Resource* Eine Resource definiert Informationen über die Ressource oder mehrere Ressourcen für welchen Zugang angefordert wird. Jede Resource kann aus mehreren Attributen bestehen.

Optional kann eine Resource auch ein Resource Content enthalten. Das Resource Content beschreibt in XML den Anwendungsbereich der Resource auf welche zugegriffen werden soll. Der Resource Content ist im Prinzip ein fiktiver Platzhalter für den Inhalt der Ressource.

Die folgende Tabelle zeigt mögliche Interpretationen einer Resource und beschreibt dessen Format sowie einen Lösungsansatz wie auf die Informationen zugegriffen werden könnte.

Interpretationen der Resource welche erst zur Laufzeit auswertbar sind, werden mit einem „x“ in der Spalte Runtime vermerkt.

Resource	Format	Zugriff	Runtime
URL	URI	Meta information	
File	file:// URI, File Object	Meta information	
Database	SQL Query	Meta information	
XPath Expression	XPath	Meta information	
Class	String, Classname	Meta information	
Method	String, Methodname	Meta information	
Parameter	String, Parametername, Type	Meta information	x

Um den Umgang mit Annotations zu vereinfachen, könnte man folgende Abkürzungen einführen:

Resource Object	Annotation
java.io.File	@FileResource
java.net.URL	@URLResource
java.lang.Class	
java.io.InputStrea	@InputStreamResource



## Auswertung

Die Analyse hat gezeigt, dass die meisten Resource Attribute nicht zur Laufzeit erfasst werden müssen. Resource Attribute können daher meist im Voraus in Form von Annotations an den Services angebracht werden.

### 2.1.2.1 Annotations

#### Attributes

Das folgende Beispiel zeigt eine mögliche Verwendung in einem Interface der Resource Annotations.

```
@Protected
@Resource ( name="urn:smurve-clinic:resource:service" )
public interface AccountMgmt {

    @Resource (name="urn:smurve-clinic:resource:name", value="AccountList")

    Account[] listAccounts (
        @Resource (name="urn:smurve-clinic:account:owner" ) String
        ownerName );

}
```

Da gemäss XACML Spezifikation allerdings nur ein Resource Element zugelassen ist, könnte die @Resource Annotation nicht ganz schlüssig sein, da sie mehrmals vorkommt.

Aus diesem Grund würde es Sinn machen, die Annotation in @ResourceAttribute umzubenennen.

Das folgende Beispiel zeigt die Verwendung der neuen Annotation @ResourceAttribute.

```
@Protected
@ResourceAttribute ( name="urn:smurve-clinic:resource:service" )
public interface AccountMgmt {

    @ResourceAttribute (name="urn:smurve-clinic:resource:name",
    value="AccountList" )

    Account[] listAccounts (
        @ ResourceAttribute (name="urn:smurve-clinic:account:owner"
        ) String ownerName );

}
```

#### Resource Content

Um das Resource Content zu definieren könnte folgende Annotation verwendet werden.

```
@ResourceContent (value="<md:record>
    <md:patient>
        <md:patientDoB>1992-03-
21</md:patientDoB>
        <md:patient-number>55555</md:patient-
number>
    </md:patient>
</md:record")
```

*Auswertung*

Der oben genannte Ansatz mit den Abkürzungen setzt Vererbung von Annotations voraus, da sonst der Code pro Annotation dupliziert werden müsste.

Da zurzeit aber Vererbung unter Annotations nicht unterstützt wird, wird eine einheitliche Annotation `@Attribute` eingesetzt welche über ein Feld `category` kategorisiert werden kann. (Bsp. `@Attribute(category=Category.SUBJECT ...)`)

## 2.1.3 Action

*Action*

Eine Action beschreibt die Aktion welche das Subject auf der Resource ausführen will.

Die folgende Tabelle zeigt mögliche Interpretationen einer Action und beschreibt dessen Format sowie einen Lösungsansatz wie auf die Informationen zugegriffen werden könnte.

Interpretationen der Action welche erst zur Laufzeit auswertbar sind, werden mit einem „x“ in der Spalte Runtime vermerkt.

Action	Format	Zugriff	Runtime
Create	URN	Meta information	
Read	URN	Meta information	
Update	URN	Meta information	
Delete	URN	Meta information	
Method Call	String, Methodname	Meta information	
Runnable	Runnable Object	Meta information	

*Auswertung*

Action Attribute müssen nicht zur Laufzeit ausgewertet werden können.

### 2.1.3.1 Annotations

*Attributes***Attributes**

Das folgende Beispiel zeigt eine mögliche Verwendung einer Action Annotation.

```

@Protected
public interface AccountMgmt {

    @Action (name="urn:smurve-clinic:action:crud-type", value="READ" )

    Account[] listAccounts (
        @Resource (name="urn:smurve-clinic:account:owner" ) String
        ownerName );
}

```



## 2.1.4 Environment

*Environment* Folgende Interpretationen von Environment sind prinzipiell möglich:

- die Anwendung selbst
- ein Zeitstempel mit der Zeit des Logins
- Localhost

## 2.2 Domain

*Domain Analyse* Das folgende Diagramm zeigt das Analyse Diagramm des Heras<sup>AF</sup> PEP im Kontext mit Spring AOP.

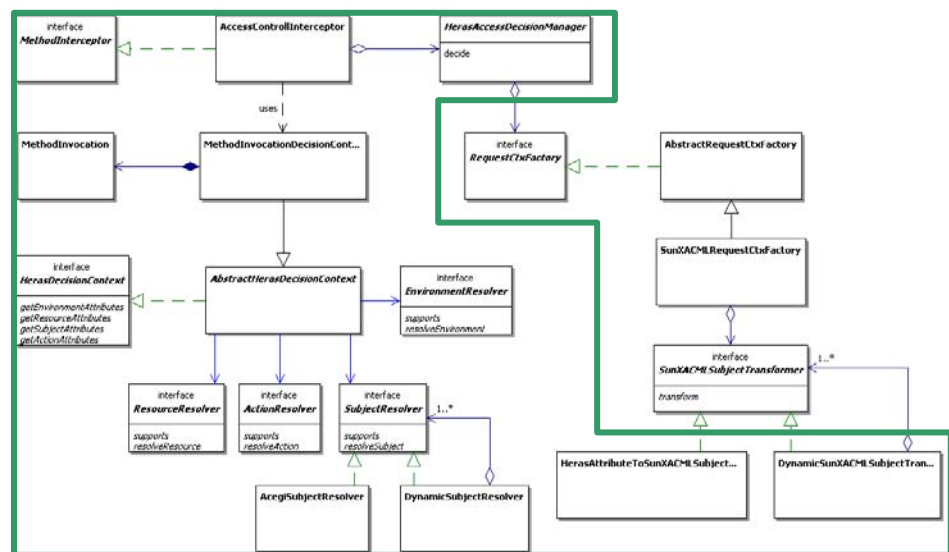


Abbildung 1 Analysis HERAS PEP

*HerasDecision Context* Der HerasDecisionContext agiert als Visitor. Er ist für die Interpretation der HERAS<sup>AF</sup> PEP Annotations zuständig und erstellt die Heras<sup>AF</sup> Attribute für Subject, Action, Resource and Environment. Diese Aufgabe delegiert der HerasDecisionContext zu dem entsprechenden Resolver.

*Resolver* Der Resolver erhält die benötigten Informationen des HerasDecisionContext und erstellt daraus die Heras<sup>AF</sup> Attribute. Um eine best mögliche Flexibilität zu erreichen, können dynamische Resolver konfiguriert werden, welche mehrere Strategien unterstützen. Dynamische Resolver werden als Composite Pattern realisiert.

*Transformer* Die Transformer sind verantwortlich um die generische HERAS Attribute in die spezifischen Implementierungen von XACML zu transformieren (z.Z. SunXACML).



## 2.3 Sequenz Diagramm

### Sequenz Diagram

Abbildung 2 zeigt das Sequenz Diagramm der Analyse Phase für einen Methodenaufruf. Wie in Abbildung 1 bezieht sich auch dieses Beispiel auf Spring AOP.

Der AccessControllInterceptor stellt den konkreten Interzeptor dar. Nachdem die invoke() Methode aufgerufen wurde, erstellt der Interzeptor einen HerasDecisionContext, welcher als Adapter agiert.

Der HerasDecisionContext kapselt die benötigten Informationen für den HerasAccessDecisionManager. Nachdem der HerasDecisionContext erstellt wurde, ruft der Interzeptor die decide() Methode des HerasAccessDecisionManager auf. Der HerasAccessDecisionManager ist nun in der Lage den XACML Request zu erstellen.

Die benötigten Attribute Informationen erhält der HerasAccessDecisionManager über den HerasDecisionContext. Um noch mehr Flexibilität zu erreichen, delegiert der HerasDecisionContext die Anfrage an den entsprechenden Resolver weiter. Der Resolver ist in der Lage die Annotations auszuwerten und daraus Heras Attribute zu bilden. Es existieren Resolver für Subject, Action, Resource und Environment.

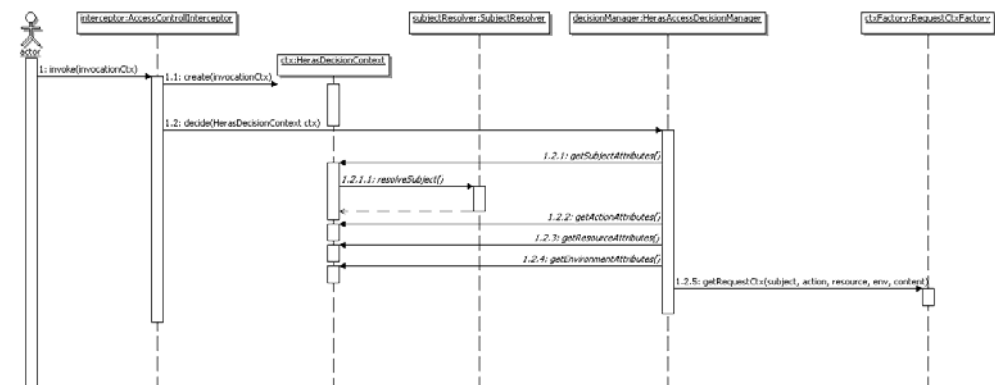


Abbildung 2 Analysis HERAS PEP Sequence Diagram



## 3 Anforderungen

### 3.1 Funktionale

*Standard-konform* XACML-Request und Response müssen standardkonform übertragen werden.

### 3.2 Nichtfunktionale

*Erweiterbarkeit* Bei der Architektur und dem Design soll darauf geachtet werden, dass neue Interzeptoren Technologien integriert und/oder erweitert werden können.

*Wartbarkeit / Austauschbarkeit* Damit die Software Architektur auch zu einem späteren Zeitpunkt noch wartbar bleibt, besteht die Forderung nach einer guten Dokumentation.



## 4 Schlussbericht

### 4.1 Zusammenfassung

Im Vorfeld der Arbeit musste wir uns in die XACML 2.0 [XACML20SPEC] Spezifikation einarbeiten. Die XACML 2.0 [XACML20SPEC] Spezifikation war ein wichtiger Bestandteil unserer Arbeit im Bezug auf das Abbilden von Methodenaufrufen auf einen XACML Request.

Die Vorarbeit erwies sich als relativ aufwändig und kompliziert, da es viele neue Technologien zu erlernen gab und da das komplette HERAS<sup>AF</sup> – Projekt sehr abstrakt aufgesetzt war.

In der ersten Phase – XACML Request Analyse – ging es darum die XACML Spezifikation zu analysieren um den Ablauf sowie die verschiedenen Attribute (Subject, Resource, Action, Environment) zu verstehen.

Es wurde ausserdem analysiert wie sich Methodenaufrufe auf einen XACML Request abbilden lassen.

Diese erste Phase war sehr stark konzeptionell und daher ein guter Einstieg in die XACML 2.0 Materie.

Zum Abschluss der Phase – XACML Request Analyse – eine gängige Methode für das Abbilden von Methodenaufrufen auf einen XACML Request gefunden und soweit dokumentiert.

Um nun einen ersten vertikalen Prototyp zu erstellen, entschieden wir uns für SpringAOP als Interzeptor-Technologie. Die Integration verlief ohne weitere Probleme und am Schluss der Phase hatten wir einen funktionierenden vertikalen Prototyp des Heras<sup>AF</sup> PEP mit SpringAOP realisiert.

Nachdem die Architektur für die Integration mit SpringAOP ausgelegt war, wurde die Integration von AspectJ als Alternative zu SpringAOP implementiert. Da AspectJ für uns Neuland war, mussten wir uns zuerst einarbeiten. Nach einigen Versuchen und Tutorials konnten wir den passenden Pointcut definieren und waren somit bereit für die Integration mit AspectJ.

Nach Abschluss der Phase AspectJ konnten wir sehr viel duplizierten Code mit gezieltem Refactorings eliminieren.

Die nächste Phase – Refactoring – widmeten wir daher voll dem Refactoring. Der Code wurde vereinfacht und Abläufe wurden optimiert.

### 4.2 Erreichte Ziele

In dieser Studienarbeit entstand eine Bibliothek, welche es erlaubt beliebige Methodenaufrufe von Applikationen abzufangen, sicherheitsrelevante Informationen zu sammeln und eine Entscheidungsanfrage in Form eines standradkonformen SAML/XACML Requests an einen Wohldefinierten Policy Decision Point (PDP) zu senden. Damit wurde die Grundlage für eine zentralisierte Sicherheitsverwaltung in J2EE Landschaften ermöglicht.

### 4.3 Ausblick

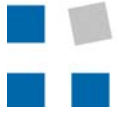
In einer möglichen Folgearbeit könnten weitere Interzeptoren Technogien (wie zum Beispiel EJB 3.0) integriert werden.

Um die Antwort des PDP gemäss XACML 2.0 Spezifikation auszuwerten und zu interpretieren müssen die verschiedenen Ausprägungen des PEP konfiguriert werden können.

Diese umfassen:

- Base PEP
- Deny-biased PEP
- Permit-biased PEP

Eine andere Erweiterung könnte die Integration des Resource Content gemäss XACML 2.0 Spezifikation sein.



## Teil IV: Anhang A



## 5 Persönliche Berichte

### 5.1 Sandro Strebel

Zu Beginn der Studienarbeit war die Aufgabenstellung und die komplexe Materie des Heras<sup>AF</sup> Projektes nicht ganz klar.

Die Workshops zu Beginn der Arbeit waren dabei sehr hilfreich um sich in die Problemstellung einzuarbeiten.

In den ersten Wochen versuchten wir etwas Licht ins Dunkle zu bringen und arbeiteten uns in die XACML 2.0 Spezifikation ein.

Mit der Zeit wurde dann die Problemstellung immer klarer und ich konnte mich immer mehr damit auseinandersetzen. Die komplexen und spannenden Probleme waren stets eine Herausforderung und es was sehr spannend geeignete Konzepte und Architekturen zu entwickeln.

In den wöchentlichen Statusmeetings konnten wir unsere neuen Erkenntnisse jeweils präsentieren. Ich fand diese Statusmeetings sehr wertvoll; einerseits konnten wir unsere Architekturentscheide mit Wolfgang Giersche sehr kompetent diskutieren und andererseits konnten wir so unsere neuen Erkenntnisse festigen.

Dies war meine zweite Studienarbeit an der HSR. Ich habe den ganzen Projektverlauf als sehr positiv empfunden. Die Kommunikation mit Massimo Cerqui und auch die Kommunikation mit der anderen Gruppe erwiesen sich als sehr leicht und angenehm.

### 5.2 Massimo Cerqui

Als wir den Zuschlag für die Studienarbeit erhielten, war mir noch nicht ganz klar, wie der Ablauf dieser Studienarbeit genau einzuschätzen ist.

Zu Beginn war es nicht ganz einfach die Komplexität des Projektes zu verstehen.

Doch nach einer Einarbeitungszeit löste sich der Nebel allmählich auf und es war spannend an dem Projekt zu arbeiten.

Sehr hilfreich waren die Wöchentlichen Meetings mit Wolfgang Giersche und Renè Eggenschwiler, in denen wir unsere Erfahrungen und Probleme austauschen konnten und immer auf offene Ohren gestossen sind.

Ebenfalls vorteilhaft war es, dass wir jeweils an fixen Tagen an der Studienarbeit gearbeitet haben. So konnten wir uns an den jeweiligen Tagen voll und ganz auf die Arbeit konzentrieren.

Mich hat diese zweite Studienarbeit einen ganzen Schritt weiter gebracht. Als sehr angenehm empfand ich die Zusammenarbeit mit Sandro Strebel, der anderen Gruppe und den Betreuern bzw. Experten. Die Kommunikation klappte reibungslos und wir konnten sehr gut miteinander Arbeiten.

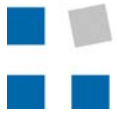


## 6 Glossar

<i>HERAS<sup>AF</sup></i>	Holistic Enterprise Ready Application Security <small>Architecture Framework</small>
<i>XACML</i>	eXtensible Access Control Markup Language
<i>PEP</i>	Policy Enforcement Point
<i>PDP</i>	Policy Decision Point

## 7 Referenzen

- [DAYG]* Yan Graf, Diplomarbeit, Distributed Access Control Policies - Enterprise Ready, Abteilung Informatik Hochschule für Technik Rapperswil, 2006
- [DARE]* René Eggenschwiler, Diplomarbeit, HERAS<sup>AF</sup> Manageable policy-based access control for J2EE, Abteilung Informatik Hochschule für Technik Rapperswil, 2006
- [XACML20SPEC]* T. Moses, ed., OASIS eXtensible Access Control Markup Language (XACML) Versions 1.0, 1.1, and 2.0. Available on the OASIS XACML TC web page at [http://www.oasis-open.org/committees/tc\\_home.php?wg\\_abbrev=xacml](http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml).



HSR  
HOCHSCHULE FÜR TECHNIK  
RAPPERSWIL



**Student's Research Project**  
**HERAS<sup>AF</sup> PEP User Guide**

Department of Computer Sciences  
University of Applied Sciences Rapperswil

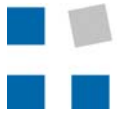
Summer semester 2007

Students: Sandro Strebelt, Massimo Cerqui  
Responsible lecturer: Wolfgang Giersche



## Table of contents

<b>PART I: INTRODUCTION.....</b>	<b>3</b>
1 INTRODUCTION.....	4
1.1 DEFINITION.....	4
1.2 GLOSSARY .....	5
1.3 OVERVIEW .....	6
<b>PART II: SPRING CONFIGURATION .....</b>	<b>7</b>
1 SPRING CONFIGURATION.....	8
1.1 APPLICATION.CTX.XML .....	9
1.2 HERAS-SECURITY.CTX.CML .....	9
1.3 HERASP-PEP.CTX.XML .....	9
1.4 ACEGISEcurity.CTX.XML.....	11
1.5 HERAS-PDP.CTX.XML.....	12
1.6 HERAS-SPRINGAOP.CTX.XML.....	13
1.7 REQUEST-FACTORY CTX.XML.....	14
1.8 ATTRIBUTE-CONFIG.XML .....	15
<b>PART II: ANNOTATIONS.....</b>	<b>16</b>
2 ANNOTATION OVERVIEW.....	17
2.1 @PROTECTED.....	17
2.2 @ATTRIBUTE .....	17
2.3 @ATTRIBUTES .....	18
2.4 @REQUESTCTXONLY.....	19
2.5 @RETURNCONTEXT .....	20
<b>PART II: TUTORIALS.....</b>	<b>21</b>
3 HERAS <sup>AF</sup> WITH SPRING AOP AND ACEGI.....	22
3.1 INTRODUCTION .....	22
3.2 STEP 1 - ENVIRONMENT SETUP.....	22
3.3 STEP 2 - SPRING CONFIGURATION .....	23
3.4 STEP 3 - DECORATING THE INTERFACE WITH ANNOTATIONS .....	27
4 HERAS <sup>AF</sup> WITH ASPECTJ AND ACEGI .....	29
4.1 INTRODUCTION .....	29
4.2 STEP 1 - ENVIRONMENT SETUP.....	29
4.3 STEP 2 - SPRING CONFIGURATION .....	30
4.4 STEP 3 - DECORATING THE INTERFACE WITH ANNOTATIONS .....	34



# Part I: Introduction



# 1 Introduction

## 1.1 Definition

- What is HERAS<sup>AF</sup> PEP* HERAS<sup>AF</sup> PEP is part of the HERAS<sup>AF</sup>. It is responsible to intercept actions, create a RequestCtx and send that to the PDP. Afterwards it must interpret the response and grant or deny access.
- What is HERAS<sup>AF</sup> PDP* HERAS<sup>AF</sup> PDP is part of HERAS<sup>AF</sup>. It is responsible to evaluate requests and grant access based on stored policies.  
The HERAS<sup>AF</sup> PDP offers a webservice to the HERAS<sup>AF</sup> PEP. It is responsible for receiving requests from a HERAS<sup>AF</sup> PEP, evaluating the request and sending a response back to the PEP.
- What is XACML* XACML stands for eXtensible Access Control Markup Language. It is a declarative access control policy language implemented in XML and a processing model, describing how to interpret the policies.
- What are annotations* *Annotations* provide data about a program that is not part of the program itself. They have no direct effect on the operation of the code they annotate.  
Annotations can be applied to a program's declarations of classes, fields, methods, and other program elements.<sup>1</sup>
- What is AOP* *Aspect-Oriented Programming (AOP)* complements Object-Oriented Programming (OOP) by providing another way of thinking about program structure. In addition to classes, AOP gives you *aspects*. Aspects enable modularization of concerns such as transaction management that cut across multiple types and objects. (Such concerns are often termed *crosscutting* concerns.)<sup>2</sup>
- What is SpringAOP* The Spring AOP framework is interception-based and configured at runtime. This removes the need for a special compilation step or load-time weaving. On the other hand interception only allows for public or protected method executions on existing object as join point. The trade-off between simplicity and available features is an important one as Spring AOP is reasonably powerful yet remains reasonably easy to learn.<sup>3</sup>

---

<sup>1</sup> <http://java.sun.com/docs/books/tutorial/java/javaOO/annotations.html>

<sup>2</sup> <http://www.springframework.org/docs/reference/aop.html>

<sup>3</sup> <http://de.wikipedia.org>



*What is AspectJ*      *AspectJ* is an aspect-oriented extension to the Java programming language created at Xerox PARC and now available in Eclipse Foundation open-source projects, both stand-alone and integrated into Eclipse. AspectJ has become the widely-used de-facto standard for AOP by emphasizing simplicity and usability for end users. It uses Java-like syntax and has included IDE integrations for displaying crosscutting structure since its initial public release in 2001.<sup>4</sup>

## 1.2 Glossary

<i>HERAS<sup>AF</sup></i>	Holistic Enterprise Ready Application Security <small>Architecture Framework</small>
<i>XACML</i>	eXtensible Access Control Markup Language
<i>Action</i>	An operation on a resource
<i>Resource</i>	Data, service or system component
<i>Subject</i>	An actor whose attributes may be referenced by a predicate
<i>Environment</i>	The set of attributes that are relevant to an authorization decision and are independent of a particular subject, resource or action
<i>Attribute</i>	Characteristic of a subject, resource, action or environment that may be referenced in a predicate or target
<i>Decision request</i>	The request by a PEP to a PDP to render an authorization decision

---

<sup>4</sup> wikipedia



## 1.3 Overview

### *Introduction*

Welcome to the user's guide for the Heras<sup>AF</sup> PEP Implementation.

This guide will introduce you to the APIs, the Spring Configuration, the XACML (eXtensible Access Control Markup Language) standard, and how to build support for XACML into your application using Heras<sup>AF</sup>. For more information about XACML, you should visit the OASIS XACML technical committee's web site<sup>5</sup>.

In a nutshell, XACML is a general-purpose access control policy language. This means that it provides a syntax (defined in XML) for managing authorization decisions.

### *XACML*

#### **A brief Introduction to XACML**

XACML is an OASIS<sup>6</sup> 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).<sup>7</sup>

### *PEP*

#### **What is HERAS<sup>AF</sup> PEP**

The Policy Enforcement Point (PEP) makes a request to whatever actually protects that resource (like a web server for example). The PEP will then send this request to a Policy Decision Point (PDP), which will look at the request, find some policy that applies to the request, and come up with an answer about whether access should be granted. That answer is returned to the PEP, which can then allow or deny access to the requester.

To generate such a request and evaluate it against the PDP multiple information and metadata is required.

Heras<sup>AF</sup> PEP lets you annotate your services - which should be protected - with Java 1.5 annotations and defines an easy way for the mapping between method invocations and XACML requests.

---

<sup>5</sup> <http://www.oasis-open.org/committees/xacml/>

<sup>6</sup> <http://www.oasis-open.org/>

<sup>7</sup> <http://sunxacml.sourceforge.net/>



## Part II: Spring Configuration



# 1 Spring Configuration

## Configuration Files

To use the HERAS<sup>AF</sup> PEP the following configuration files are needed:

### **application.cxt.xml**

Main Spring application context.

### **heras-security.ctx.xml**

Heras<sup>AF</sup> Security configuration.

### **heras-pep.ctx.xml**

Heras<sup>AF</sup> PEP core configuration.

### **acegisecurity.ctx.xml**

ACEGI Security configuration.

### **heras-pdp.ctx.xml**

Configures the PDP relevant information.

### **heras-springaop.ctx.xml (Not used if you work with AspectJ)**

Spring AOP Interceptor configuration.

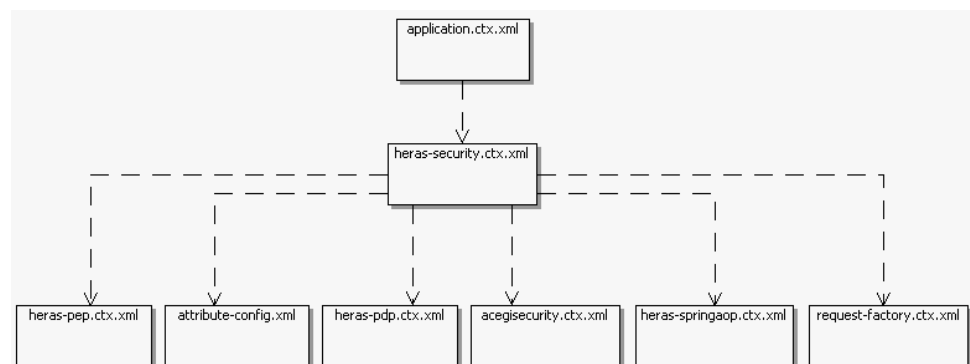
### **request-factory.ctx.xml**

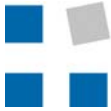
Configuration for the different Transformers.

### **attribute-config.xml**

Conversion table between Attribute Id and XACML Type.

## Overview





## 1.1 application.ctx.xml

application.ctx.xml The application.cxt.xml is the context that will be loaded. It imports all the other application context files listed below.

## 1.2 heras-security.ctx.cml

heras-security.ctx.xml The heras-security.ctx.xml is the context that covers the complete Heras<sup>AF</sup> PEP configuration by importing the relevant configurations.

## 1.3 herasp-pep.ctx.xml

Overview The Heras PEP core configuration is located in the heras-pep.ctx.xml file.

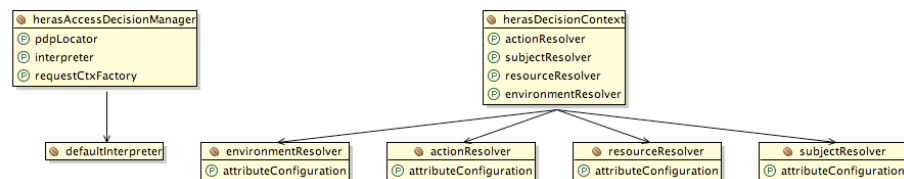


Figure 1 Heras PEP context

### herasDecision Context

The herasDecisionContext defines the interceptor specific DecisionContext. Each DecisionContext consists of an actionResolver, a subjectResolver, a resourceResolver and an environmentResolver.

Actually Heras<sup>AF</sup> PEP supports the following interceptor technologies:

- SpringAOP
  - [org.herasaf.pep.integration.springaop.MethodInvocationDecisionContext](#)
- AspectJ.
  - [org.herasaf.pep.integration.aspectj.JoinPointDecisionContext](#)

To configure the Heras<sup>AF</sup> PEP to work with SpringAOP the following implementation of the DecisionContext may be used:

```
<bean id="herasDecisionContext"
      class="org.herasaf.pep.integration.springaop.MethodInvocationDecisionContext">
```



*actionResolver* The four resolvers are responsible to resolve the specific metadata to a set of *HerasAttributes*.  
Currently Heras<sup>AF</sup> PEP supports the following metadata strategies:

- Java 1.5 annotations

To configure Heras<sup>AF</sup> PEP to work with Java annotations the adequate Resolver must be used.

The following listing shows a example configuration for Java 1.5 annotation support.

```
<bean id="actionResolver"
      class="org.herasaf.pep.resolvers.AnnotationActionResolver">
  <property name="attributeConfiguration">
    <ref bean="attributeConfiguration" />
  </property>
</bean>
```

*Resource Resolver* See *actionResolver*.

*subjectResolver* See *actionResolver*.

*environment Resolver* See *actionResolver*.

*herasAccess Decision Manager* The *herasAccessDecisionManager* is going to be called from within the *AccessControllInterceptor* of the specific interceptor technology.

At this point no special configuration is needed, the default implementation [org.herasaf.pep.HerasAccessDecisionManagerImpl](#) may be used.

```
<bean id="herasAccessDecisionManager"
      class="org.herasaf.pep.HerasAccessDecisionManagerImpl">

  <property name="pdpLocator">
    <ref bean="pdpLocator" />
  </property>

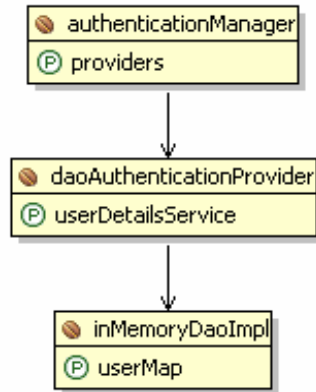
  <property name="interpreter">
    <ref bean="defaultInterpreter" />
  </property>

  <property name="requestCtxFactory">
    <ref bean="request-factory" />
  </property>
</bean>
```



## 1.4 acegisecurity.ctx.xml

### Overview



**Figure 2** ACEGI Security context

Acegi Security includes a production-quality `AuthenticationProvider` implementation called `DaoAuthenticationProvider`. This authentication provider is compatible with all of the authentication mechanisms that generate a `UsernamePasswordAuthenticationToken`, and is probably the most commonly used provider in the framework. Like most of the other authentication providers, the `DaoAuthenticationProvider` leverages a `UserDetailsService` in order to lookup the username, password and `GrantedAuthority[]`s. Unlike most of the other authentication providers that leverage `UserDetailsService`, this authentication provider actually requires the password to be presented, and the provider will actually evaluate the validity or otherwise of the password presented in an authentication request object.<sup>8</sup>

### Authentication Manager

The authentication manager utilizes an authentication provider that has users, passwords and roles configured within this XML file. In real life you would certainly find LDAP or JDBC-based providers here.

### inMemory DaoImpl

Whilst it is easy to use create a custom `UserDetailsService` implementation that extracts information from a persistence engine of choice, many applications do not require such complexity. This is particularly true if you're undertaking a rapid prototype or just starting integrating Acegi Security, when you don't really want to spend time configuring databases or writing `UserDetailsService` implementations. For this sort of situation, a simple option is to configure the `InMemoryDaoImpl` implementation:

```

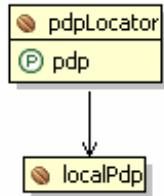
<bean id="inMemoryDaoImpl"
      class="org.acegisecurity.userdetails.memory.InMemoryDaoImpl">
  <property name="userMap">
    <value>
      ama=ama, PRACTITIONER, SUPERVISOR
    </value>
  </property>
</bean>
  
```

<sup>8</sup> <http://www.acegisecurity.org/docbook/acegi.html#dao-provider>



## 1.5 heras-pdp.ctx.xml

### Overview



**Figure 3** Heras PDP context

The heras-pdp.ctx.xml application context configures the PDP relevant information.

```

<!--*****
Locates the actual PDP implementation within the current JVM
***** -->
<bean name="pdpLocator"
      class="org.herasaf.com.pdp.JVMLocalPdpLocator">
  <property name="pdp">
    <ref bean="heras-pdp" />
  </property>
</bean>

</bean>

<bean id="heras-pdp"
      class="org.herasaf.webservice.client.axis.HerasSamlPDP">
  <property name="client">
    <ref bean="pdpService" />
  </property>
</bean>

<bean id="pdpService"
      class="org.herasaf.webservice.client.axis.HerasPDPServiceProxy">
  <constructor-arg>
    <ref bean="serviceLocator" />
  </constructor-arg>
</bean>

<bean id="serviceLocator"
      class="org.herasaf.webservice.client.axis.
              HerasPDPService_ServiceLocator">
  <property name="herasPDPServiceAddress">
    <value>
      http://localhost:8081/heras-ws-server/services/HerasPDPService
    </value>
  </property>

```



```

<property name="herasPDPServiceWSDDServiceName">
  <value>HerasPDPService</value>
</property>

<property name="name">
  <value>HerasPDPService</value>
</property>

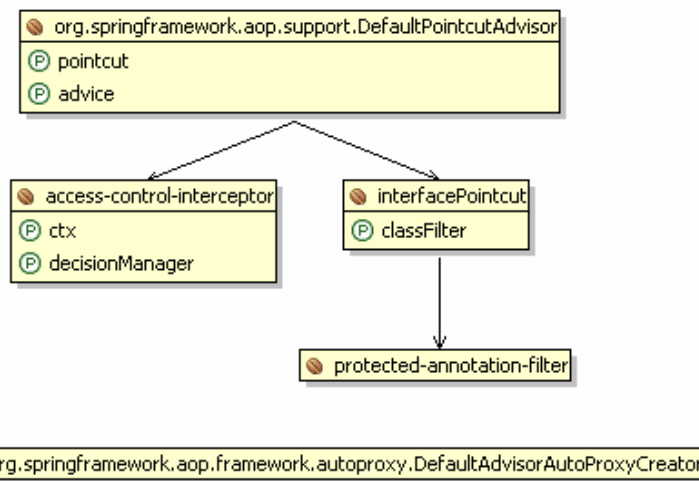
<property name="qualifiedName">
  <value>urn:axis:webservice:herasaf:org</value>
</property>
</bean>

```

*pdpLocator* The pdpLocator locates the actual PDP implementation within the current JVM. Each PdpLocator has a property "pdp" which represents the concrete PDP.

## 1.6 heras-springaop.ctx.xml

### Overview



**Figure 4** SpringAOP context

The heras-springaop.ctx.xml configures the Spring AOP Interceptor relevant information.

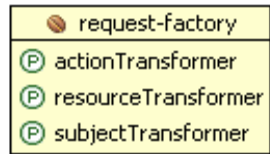
### *Protected Annotation Filter*

SpringAOP uses an interface pointcut which actually only intercepts interfaces or classes which are annotated with @Protected.



## 1.7 request-factory ctx.xml

### Overview



**Figure 5** Request Factory context

The request-factory.ctx.xml configures the different Transformers relevant for the requestFactory.

### *Action Transformer*

The Action Transformer is responsible to transform HerasAttributes to SunXACML Attributes.

By using transformers the specific XACML implementation may easily be replaced by another implementation. This gives us the needed flexibility to work independent of the specific implementation.

### *Resource Transformer*

The Resource Transformer is responsible to transform HerasAttributes to SunXACML Attributes.

By using transformers the specific XACML implementation may easily be replaced by another implementation. This gives us the needed flexibility to work independent of the specific implementation.

### *Subject Transformer*

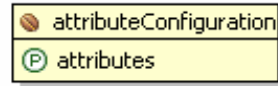
The Subject Transformer is responsible to transform HerasAttributes to SunXACML Attributes.

By using transformers the specific XACML implementation may easily be replaced by another implementation. This gives us the needed flexibility to work independent of the specific implementation.



## 1.8 attribute-config.xml

### Overview

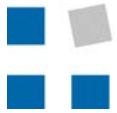


**Figure 6** attribute config

The Data Type of an XACML Attribute describes the data-type of the contents of the `<xacml-context:AttributeValue>` element. This shall be either a primitive type defined by the XACML 2.0 specification or a type defined in a namespace in the `<xacml-context>` element.

The attribute-config.xml configures the conversion table between AttributeId and XACML Type.

```
<bean id="attributeConfiguration"
      class="org.herasaf.util.HerasAttributeConfiguration">
  <property name="attributes">
    <map>
      <entry key="urn:smurve-clinic:resource:name"
            value="http://www.w3.org/2001/XMLSchema#string"/>
    </map>
  </property>
</bean>
```



## Part II: Annotations



## 2 Annotation Overview

The annotation overview shows an overview of all annotations used for HERAS<sup>AF</sup> PEP.

### 2.1 @Protected

*Overview* The `org.herasaf.pep.annotations.Protected` annotation marks a Class or an Interface to be intercepted by the Heras AccessControllInterceptor. Only when the `@Protected` annotation is used, an Interface or Class is protected by HERAS<sup>AF</sup>.

```
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
@Inherited
public @interface Protected {

}
```

*Example*

```
@Protected
public interface AccountMgmt {
}
```

### 2.2 @Attribute

*Overview* The `org.herasaf.pep.annotations.Attribute` annotation is the central abstraction of the XACML Request Context. It contains Attribute meta-data and one or more attribute values. There are four Categories for the Attribute element. SUBJECT, RESOURCE, ACTION and ENVIRONMENT.

If the value is not specified, it will be taken from the context, such as shown in Table 1 (see example):

Layer	Default value	E.g.
Class	Full Class name	<code>org.smurve.clinic.PatientRecordDao</code>
Interface	Full Interface name	<code>org.smurve.clinic.IPatientRecordDao</code>
Method	Method name	<code>retrieveRecord</code>
Parameter	Runtime value	<code>"1040"</code>

**Table 1** Default Attribute annotation values



```
@Target( { ElementType.TYPE, ElementType.METHOD,  
ElementType.PARAMETER } )  
@Retention(RetentionPolicy.RUNTIME)  
@Inherited  
public @interface Attribute {  
    String id();  
    String issuer() default "";  
    String value() default "";  
    Category category();  
}
```

#### Example

Because the value of the `@Attribute` annotation is not specified, the value of the resource attribute `"urn:smurve-clinic:patient"` in the example below will be the runtime value of the `patientId` parameter.

```
public PatientRecord retrieveRecord (  
    @Attribute (category = Category.RESOURCE,  
    name="urn:smurve-clinic:patient" ) String patientId );
```

## 2.3 @Attributes

#### Overview

If more than one `@Attribute` annotation is needed, the `org.herasaf.pep.annotations.Attributes` annotation can be used.

```
@Target( { ElementType.TYPE, ElementType.METHOD,  
ElementType.PARAMETER } )  
@Retention(RetentionPolicy.RUNTIME)  
@Inherited  
public @interface Attributes {  
    Attribute[] value();  
}
```

#### Example

```
@Protected  
public interface PatientRecordDao {  
  
    @Attributes( {  
        @Attribute (category = Category.RESOURCE,  
        id="urn:smurve-clinic:resource:name" ),  
        @Attribute (category = Category.ACTION,  
        id="urn:smurve-clinic:action:crud-type" ,  
        value="read" )  
    } )  
  
    public PatientRecord retrieveRecord(String patientId);  
}
```



## 2.4 @RequestCtxOnly

### Overview

The `org.herasaf.pep.annotations.RequestCtxOnly` may only be used in conjunction with a SAML PDP. It specifies that only the RequestContext must be evaluated to make the decision.

This annotations governs the sources of information that the PDP is allowed to use in making its authorization decision. If the value of `@RequestCtxOnly` is "true", then the authorization decision SHALL be made solely on the basis of information contained in the request context no external attributes MAY be used. If the value of `@RequestCtxOnly` is "false", then the authorization decision MAY be made on the basis of external Attributes.<sup>9</sup>

```
@Target( { ElementType.TYPE, ElementType.METHOD,
          ElementType.PARAMETER })
@Retention(RetentionPolicy.RUNTIME)
@Inherited
public @interface RequestCtxOnly {

    boolean value() default false;

}
```

### Example

```
@Protected
public interface PatientRecordDao {

    @Attributes( {
        @Attribute (category = Category.RESOURCE,
                    id="urn:smurve-clinic:resource:name"),
        @Attribute (category = Category.ACTION,
                    id="urn:smurve-clinic:action:crud-type",value="read")
    })
    @RequestCtxOnly(true)
    public PatientRecord retrieveRecord(String patientId);

}
```

---

<sup>9</sup> SAML 2.0 profile of XACML v2.0



## 2.5 @ReturnContext

### Overview

The `org.herasaf.pep.annotations.ReturnContext` may only be used in conjunction with a SAML PDP.

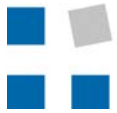
The `@ReturnContext` annotation allows the PEP to request that an `<xacml-context:Request>` element be included in the `<XACMLAuthzDecisionStatement>` resulting from the request. It also governs the contents of that `<xacml-context:Request>` element. If the value of `@ReturnContext` is "true", then the PDP SHALL include the `<xacmlcontext:Request>` element in the `<XACMLAuthzDecisionStatement>` element in the `<XACMLResponse>`. This `<xacml-context:Request>` element SHALL include all those attributes supplied by the PEP in the `<XACMLAuthzDecisionQuery>` that were used in making the authorization decision. The PDP MAY include additional attributes in this `<xacml-context:Request>` element, such as external attributes obtained by the PDP and used in making the authorization decision, or other attributes known by the PDP that may be useful to the PEP in making subsequent `<XACMLAuthzDecisionQuery>` requests. If the value of `@ReturnContext` is "false", then the PDP SHALL NOT include the `<xacmlcontext:Request>` element in the `<XACMLAuthzDecisionStatement>` element of the `<XACMLResponse>`<sup>10</sup>

```
@Target( { ElementType.TYPE, ElementType.METHOD,  
          ElementType.PARAMETER } )  
@Retention(RetentionPolicy.RUNTIME)  
@Inherited  
public @interface ReturnContext {  
  
    boolean value() default false;  
  
}
```

### Example

```
@Protected  
public interface PatientRecordDao {  
  
    @Attributes( {  
        @Attribute (category = Category.RESOURCE,  
                    id="urn:smurve-clinic:resource:name"),  
        @Attribute (category = Category.ACTION,  
                    id="urn:smurve-clinic:action:crud-type",  
                    value="read")  
    } )  
    @ReturnContext(true)  
    public PatientRecord retrieveRecord(String patientId );  
  
}
```

<sup>10</sup> SAML 2.0 profile of XACML v2.0



## Part II: Tutorials



## 3 HERAS<sup>AF</sup> with Spring AOP and ACEGI

### 3.1 Introduction

#### *Introduction*

In the following tutorial we will show you how you can protect your services using HERAS<sup>AF</sup> and Spring AOP.

During this tutorial you will learn, how to configure the HERAS<sup>AF</sup> PEP and how to annotate your interfaces with the HERAS<sup>AF</sup> annotations.

Let's start with the following plain interface.

```
public interface PatientRecordDao {  
    public PatientRecord retrieveRecord(String patientId );  
}
```

### 3.2 STEP 1 - Environment setup

#### *Environment Setup*

To use the HERAS<sup>AF</sup> annotations make sure, the herasaf-pep library is in your class path.

For the configuration of the HERAS<sup>AF</sup> PEP, you will also need the Spring application context libraries in your class path:

- spring-beans-2.0.jar
- spring-context-2.0.jar
- spring-core-2.0.jar
- spring-aop-2.0.jar

#### **Checklist**

- herasaf-pep library in class path
- Spring application context libraries in class path



## 3.3 STEP 2 - Spring configuration

### Introduction

In this step you will learn how to configure the different application contexts of the heras-security context.

### Spring AOP (heras- springaop. ctx.xml)

Because the default Spring AOP configuration is already done in the heras-pep demo, you may use the application context from the demo folder and adapt it for your needs.

The only thing you have to configure is the protected annotation filter of the `protected-annotation-filter` bean.

```
<bean name="protected-annotation-filter" singleton="false"
      class="org.herasaf.pep.integration.springaop.ProtectedAnnotationFilter">
</bean>
```

### ACEGI security (acegisecurity. ctx.xml)

To enable the ACEGI security, introduce a new application context `acegisecurity.ctx.xml` and place it in the same directory as the other application context files.

The heras-pep demo has once again a default configuration ready.

To simplify matters we will use ACEGI with an in memory authentication DAO. Adapt the usernames, passwords and roles for your requirements.

```
<bean id="inMemoryDaoImpl"
      class="org.acegisecurity.userdetails.memory.InMemoryDaoImpl">
  <property name="userMap">
    <value>
      ama=ama,PRACTITIONER, SUPERVISOR
      nok=nok,PRACTITIONER, DIRECTOR
      hum=hum,CONTROLLER
    </value>
  </property>
</bean>
```

### Request Factory (request- factory.ctx.xml)

The implementation that comes with the HERAS<sup>AF</sup> distribution contains a `RequestCtxFactory` implementation and should be configured with four transformers that assist in transforming the proprietary representations of Subjects, Resources, Action and Environment to a XACML specific representations.

The `request-factory.ctx.xml` located in the demo folder shows the factory configured with `HerasAttributeTransformers`.

Nothing to configure here.

```
<bean name="request-factory" class="org.herasaf.integration.
sunxacml.ctx.SunXACMLRequestCtxFactor">
  <property name="actionTransformer">
    <bean class="org.herasaf.pep.transformers.
HerasAttributeToSunXACMLActionTransformer"/>
  </property>
```



```
<property name="resourceTransformer">
  <bean class="org.herasaf.pep.transformers.
    HerasAttributeToSunXACMLResourceTransformer" />
</property>
<property name="subjectTransformer">
  <bean class="org.herasaf.pep.transformers.
    HerasAttributeToSunXACMLSubjectTransformer" />
</property>
</bean>
```

*HERASAF  
PDP (heras-  
pdp.ctx.xml)*

Now it's time to configure the location of the PDP, which the HerasAccessDecisionManager is supposed to consult.

You may again use the heras-pdp.ctx.xml located in the demo folder and change the pdpLocator.

```
<bean name="pdpLocator" class="org.herasaf.com.pdp.
    JVMLocalPdpLocator">
  <property name="pdp">
    <ref bean="localPdp" />
  </property>
</bean>
<bean name="localPdp" class="org.herasaf.pep.demo.ReportingPdp">
</bean>
```

*HERASAF  
PEP (heras-  
pep.ctx.xml)*

The core settings of the HERAS<sup>AF</sup> PEP are located in the heras-pep.ctx.xml. The HerasDecisionContext should be configured with four resolvers that assist in resolving the metadata of the annotations to HerasAttributes. The HerasAccessDecisionManager, who is going to be called from within the HerasAccessControlInterceptor should be configured with the adequate pdpLocator configured before in heras-pdp.ctx.xml, an interpreter and the RequestCtxFactory from the request-factory.ctx.xml.

```
<bean id="herasDecisionContext"
class="org.herasaf.pep.integration.springaop.MethodInvocationDecis
ionContext">
  <property name="actionResolver">
    <ref bean="actionResolver" />
  </property>
  <property name="subjectResolver">
    <ref bean="subjectResolver" />
  </property>
  <property name="resourceResolver">
    <ref bean="resourceResolver" />
  </property>
  <property name="environmentResolver">
    <ref bean="environmentResolver" />
  </property>
</bean>
```



```
<bean id="actionResolver"
      class="org.herasaf.pep.resolvers.AnnotationActionResolver">
  <property name="attributeConfiguration">
    <ref bean="attributeConfiguration" />
  </property>
</bean>

...

<bean id="herasAccessDecisionManager"
      class="org.herasaf.pep.HerasAccessDecisionManagerImpl">
  <property name="pdpLocator">
    <ref bean="pdpLocator" />
  </property>
  <property name="interpreter">
    <ref bean="defaultInterpreter" />
  </property>
  <property name="requestCtxFactory">
    <ref bean="request-factory"/>
  </property>
</bean>
```

*Attribute  
configuration  
(attribute-  
config.xml)*

Now it's time to declare the required attribute ids. You can do this by adapt the attribute-config.xml file - located in the demo folder - for your needs.

```
<bean id="attributeConfiguration"
      class="org.herasaf.util.HerasAttributeConfiguration">
  <property name="attributes">
    <map>
      <entry key="urn:smurve-clinic:resource:name"
            value="http://www.w3.org/2001/XMLSchema#string" />
      <entry key="urn:smurve-clinic:action:crud-type"
            value="http://www.w3.org/2001/XMLSchema#string" />
      <entry key=" urn:smurve-clinic:java:architecture:dao"
            value="http://www.w3.org/2001/XMLSchema#string" />
      <entry key=" urn:smurve-clinic:patient"
            value="http://www.w3.org/2001/XMLSchema#string" />
      <entry key="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
            value="urn:oasis:names:tc:xacml:1.0:data-
            type:rfc822Name" />
    </map>
  </property>
</bean>
```



*Heras Security  
context  
(heras-  
security.ctx.xml)*

Now it's time to bring it all together. Import the just created application contexts in the main heras-security.ctx.xml file.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
    "http://www.springframework.org/dtd/spring-beans.dtd">
<beans>

    <import resource="heras-springaop.ctx.xml" />
    <import resource="request-factory.ctx.xml" />
    <import resource="acegisecurity.ctx.xml" />
    <import resource="heras-pdp.ctx.xml" />
    <import resource="heras-pep.ctx.xml" />
    <import resource="attribute-config.xml" />
    <bean name="patient-record-dao"
        class="org.herasaf.pep.demo.PatientRecordDao">
    </bean>
</beans>
```

*Main  
application  
context  
(application.ctx  
.xml)*

To use the just now created heras-security context import it in your main spring application context.

Here is also space to define your own beans.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
    "http://www.springframework.org/dtd/spring-beans.dtd">
<beans>

    <import resource="heras-security.ctx.xml" />
    <bean name="patient-record-dao"
        class="org.herasaf.pep.demo.PatientRecordDao">
    </bean>
</beans>
```



## 3.4 STEP 3 - Decorating the Interface with annotations

*Introduction* Now we are ready to decorate our interface and enable HERAS<sup>AF</sup> security.

We start with our plain interface.

```
public interface PatientRecordDao {  
    public PatientRecord retrieveRecord(String patientId );  
}
```

*@Protected* Add the @Protected annotation on class level to let Spring AOP intercept your interface.

```
@Protected  
public interface PatientRecordDao {  
  
    public PatientRecord retrieveRecord(String patientId );  
  
}
```

*@Attribute* Next, we add a @Attribute annotation on class level to define the entire DAO architectural layer as resource. This is only one possible interpretation.

```
@Protected  
@Attribute (category = Category.RESOURCE, id="urn:smurve-  
clinic:java:architecture:dao" )  
public interface PatientRecordDao {  
  
    public PatientRecord retrieveRecord(String patientId );  
  
}
```

*@Attributes* Because we need to add two @Attribute annotations on the Method Level we need to use the @Attributes annotation which carries an Array of @Attribute annotations.  
There is one attribute to come with the resource and that attribute has the name "urn:smurve-clinic:resource:name" and the value on method level is by default the name of the method. In this example "retrieveRecord".

There is an Action attribute with the name "urn:smurve-clinic:action:crud-type" and the value "read".



### @Attributes

### @Protected

```
@Attribute (category = Category.RESOURCE, id="urn:smurve-
clinic:java:architecture:dao" )
public interface PatientRecordDao {

    @Attributes( {
        @Attribute (category = Category.RESOURCE,
            id="urn:smurve-clinic:resource:name" ),
        @Attribute (category = Category.ACTION,
            id="urn:smurve-clinic:action:crud-type", value="read" )
    })
    public PatientRecord retrieveRecord( String patientId );

}
```

### Parameter Value

Finally we add another resource attribute with the name `"urn:smurve-clinic:patient"` and the value taken from the method parameter `patientId`.

That's it; your interface is fully annotated and ready to be protected by HERAS<sup>AF</sup>.

### @Protected

```
@Attribute (category = Category.RESOURCE, id="urn:smurve-
clinic:java:architecture:dao" )
public interface PatientRecordDao {

    @Attributes( {
        @Attribute (category = Category.RESOURCE,
            id="urn:smurve-clinic:resource:name" ),
        @Attribute (category = Category.ACTION,
            id="urn:smurve-clinic:action:crud-type", value="read" )
    })

    public PatientRecord retrieveRecord (
        @Attribute (category = Category.RESOURCE,
            name="urn:smurve-clinic:patient" ) String patientId );

}
```



## 4 HERAS<sup>AF</sup> with AspectJ and ACEGI

### 4.1 Introduction

#### *Introduction*

In the following tutorial we will show you how you can protect your services using HERAS<sup>AF</sup> and AspectJ.

During this tutorial you will learn, how to configure the HERAS<sup>AF</sup> PEP and how to annotate your interfaces with the HERAS<sup>AF</sup> annotations.

Let's start with the following plain interface.

```
public interface PatientRecordDao {  
    public PatientRecord retrieveRecord(String patientId );  
}
```

### 4.2 STEP 1 - Environment setup

#### *Environment Setup*

To use the HERAS<sup>AF</sup> annotations make sure, the herasaf-pep library is in your class path.

For the configuration of the HERAS<sup>AF</sup> PEP, you will also need the Spring application context libraries in your class path:

- spring-beans-2.0.jar
- spring-context-2.0.jar
- spring-core-2.0.jar

For the use of AspectJ you also need the AspectJ compiler library in your class path:

- aspectj-1.5.3.jar

If you plan to use AspectJ inside the Eclipse IDE you can just download AJDT since it includes a bundled AspectJ compiler.

- <http://www.eclipse.org/ajdt/>

#### **Checklist**

- herasaf-pep library in class path
- Spring application context libraries in class path
- AspectJ library in class path



## 4.3 STEP 2 - Spring configuration

*Introduction* In this step you will learn how to configure the different application contexts of the heras-security context.

*ACEGI security (acegisecurity.ctx.xml)* To enable the ACEGI security, introduce a new application context acegisecurity.ctx.xml and place it in the same directory as the other application context files.

The heras-peg demo has once again a default configuration ready. To simplify matters we will use ACEGI with an in memory authentication DAO. Adapt the usernames, passwords and roles for your requirements.

```
<bean id="inMemoryDaoImpl"
      class="org.acegisecurity.userdetails.memory.InMemoryDaoImpl">
  <property name="userMap">
    <value>
      ama=ama, PRACTITIONER, SUPERVISOR
      nok=nok, PRACTITIONER, DIRECTOR
      hum=hum, CONTROLLER
    </value>
  </property>
</bean>
```

*Request Factory (request-factory.ctx.xml)* The implementation that comes with the HERAS<sup>AF</sup> distribution contains a RequestCtxFactory implementation and should be configured with 4 transformers that assist in transforming the proprietary representations of Subjects, Resources, Action and Environment to XACML – specific representations. The request-factory.ctx.xml located in the demo folder shows the factory configured with HerasAttributeTransformers.

Nothing to configure here.

```
<bean name="request-factory" class="org.herasaf.integration.
sunxacml.ctx.SunXACMLRequestCtxFactor">
  <property name="actionTransformer">
    <bean class="org.herasaf.peg.transformers.
HerasAttributeToSunXACMLActionTransformer" />
  </property>
  <property name="resourceTransformer">
    <bean class="org.herasaf.peg.transformers.
HerasAttributeToSunXACMLResourceTransformer" />
  </property>
  <property name="subjectTransformer">
    <bean class="org.herasaf.peg.transformers.
HerasAttributeToSunXACMLSubjectTransformer" />
  </property>
</bean>
```



*HERASAF  
PDP (heras-  
pdp.ctx.xml)*

Now it's time to configure the location of the PDP, which the HerasAccessDecisionManager is supposed to consult.

You may again use the heras-pdp.ctx.xml located in the demo folder and change the pdpLocator.

```
<bean name="pdpLocator" class="org.herasaf.com.pdp.  
                                JVMLocalPdpLocator">  
    <property name="pdp">  
        <ref bean="localPdp"/>  
    </property>  
</bean>  
<bean name="localPdp" class="org.herasaf.pep.demo.ReportingPdp">  
</bean>
```

*HERASAF  
PEP (heras-  
pep.ctx.xml)*

The core settings of the HERAS<sup>AF</sup> PEP are located in the heras-pep.ctx.xml. The HerasDecisionContext should be configured with 4 resolvers that assist in resolving the metadata of the annotations to HerasAttributes. The HerasAccessDecisionManager, who is going to be called from within the HerasAccessControlInterceptor should be configured with the adequate pdpLocator configured before in heras-pdp.ctx.xml, an interpreter and the RequestCtxFactory from the request-factory.ctx.xml.

```
<bean id="herasDecisionContext"  
class="org.herasaf.pep.integration.aspectj.  
                                JoinPointDecisionContext">  
    <property name="actionResolver">  
        <ref bean="actionResolver" />  
    </property>  
    <property name="subjectResolver">  
        <ref bean="subjectResolver" />  
    </property>  
    <property name="resourceResolver">  
        <ref bean="resourceResolver" />  
    </property>  
    <property name="environmentResolver">  
        <ref bean="environmentResolver" />  
    </property>  
</bean>  
  
<bean id="actionResolver"  
class="org.herasaf.pep.resolvers.AnnotationActionResolver">  
    <property name="attributeConfiguration">  
        <ref bean="attributeConfiguration" />  
    </property>  
</bean>  
  
...
```



```
<bean id="herasAccessDecisionManager"
      class="org.herasaf.pep.HerasAccessDecisionManagerImpl">
  <property name="pdpLocator">
    <ref bean="pdpLocator" />
  </property>
  <property name="interpreter">
    <ref bean="defaultInterpreter" />
  </property>
  <property name="requestCtxFactory">
    <ref bean="request-factory" />
  </property>
</bean>
```

*Attribute  
configuration  
(attribute-  
config.xml)*

Now it's time to declare the required attribute ids. You can do this by adapt the attribute-config.xml file - located in the demo folder - for your needs.

```
<bean id="attributeConfiguration"
      class="org.herasaf.util.HerasAttributeConfiguration">
  <property name="attributes">
    <map>
      <entry key="urn:smurve-clinic:resource:name"
            value="http://www.w3.org/2001/XMLSchema#string" />
      <entry key="urn:smurve-clinic:action:crud-type"
            value="http://www.w3.org/2001/XMLSchema#string" />
      <entry key="urn:smurve-clinic:java:architecture:dao"
            value="http://www.w3.org/2001/XMLSchema#string" />
      <entry key="urn:smurve-clinic:patient"
            value="http://www.w3.org/2001/XMLSchema#string" />
      <entry key="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
            value="urn:oasis:names:tc:xacml:1.0:data-
            type:rfc822Name" />
    </map>
  </property>
</bean>
```



*Heras Security  
context  
(heras-  
security.ctx.xml)*

Now it's time to bring it all together. Import the just created application contexts in the main heras-security.ctx.xml file.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
    "http://www.springframework.org/dtd/spring-beans.dtd">
<beans>

    <import resource="request-factory.ctx.xml" />
    <import resource="acegisecurity.ctx.xml" />
    <import resource="heras-pdp.ctx.xml" />
    <import resource="heras-pep.ctx.xml" />
    <import resource="attribute-config.xml" />
    <bean name="patient-record-dao"
        class="org.herasaf.pep.demo.PatientRecordDao">
    </bean>
</beans>
```

*Main  
application  
context  
(application.ctx  
.xml)*

To use the just now created heras-security context import it in your main spring application context.

Here is also space to define your own beans.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
    "http://www.springframework.org/dtd/spring-beans.dtd">
<beans>

    <import resource="heras-security.ctx.xml" />
    <bean name="patient-record-dao"
        class="org.herasaf.pep.demo.PatientRecordDao">
    </bean>
</beans>
```



## 4.4 STEP 3 - Decorating the Interface with annotations

*Introduction* Now we are ready to decorate our interface and enable HERAS<sup>AF</sup> security.

We start with our plain interface.

```
public interface PatientRecordDao {  
    public PatientRecord retrieveRecord(String patientId );  
}
```

*@Protected* Add the @Protected annotation on class level to let Spring AOP intercept your interface.

```
@Protected  
public interface PatientRecordDao {  
  
    public PatientRecord retrieveRecord(String patientId );  
  
}
```

*@Attribute* Next, we add a @Attribute annotation on class level to define the entire DAO architectural layer as resource. This is only one possible interpretation.

```
@Protected  
@Attribute (category = Category.RESOURCE, id="urn:smurve-  
clinic:java:architecture:dao" )  
public interface PatientRecordDao {  
  
    public PatientRecord retrieveRecord(String patientId );  
  
}
```

*@Attributes* Because we need to add two @Attribute annotations on the Method Level we need to use the @Attributes annotation which carries an Array of @Attribute annotations.  
There is one attribute to come with the resource and that attribute has the name "urn:smurve-clinic:resource:name" and the value on method level is by default the name of the method. In this example "retrieveRecord".

There is an Action attribute with the name "urn:smurve-clinic:action:crud-type" and the value "read".



### @Attributes

### @Protected

```
@Attribute (category = Category.RESOURCE, id="urn:smurve-
clinic:java:architecture:dao" )
public interface PatientRecordDao {

    @Attributes( {
        @Attribute (category = Category.RESOURCE,
            id="urn:smurve-clinic:resource:name" ),
        @Attribute (category = Category.ACTION,
            id="urn:smurve-clinic:action:crud-type", value="read" )
    })
    public PatientRecord retrieveRecord( String patientId );

}
```

### Parameter Value

Finally we add another resource attribute with the name `"urn:smurve-clinic:patient"` and the value taken from the method parameter `patientId`.

That's it; your interface is fully annotated and ready to be protected by HERAS<sup>AF</sup>.

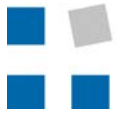
### @Protected

```
@Attribute (category = Category.RESOURCE, id="urn:smurve-
clinic:java:architecture:dao" )
public interface PatientRecordDao {

    @Attributes( {
        @Attribute (category = Category.RESOURCE,
            id="urn:smurve-clinic:resource:name" ),
        @Attribute (category = Category.ACTION,
            id="urn:smurve-clinic:action:crud-type", value="read" )
    })

    public PatientRecord retrieveRecord (
        @Attribute (category = Category.RESOURCE,
            name="urn:smurve-clinic:patient" ) String patientId );

}
```



HSR  
HOCHSCHULE FÜR TECHNIK  
RAPPERSWIL



**Student's Research Project**  
**HERAS<sup>AF</sup> PEP Developer Guide**

Department of Computer Sciences  
University of Applied Sciences Rapperswil

Summer semester 2007

Students: Sandro Streb, Massimo Cerqui  
Responsible lecturer: Wolfgang Giersche



## Table of contents

<b>PART I: INTRODUCTION.....</b>	<b>3</b>
1 INTRODUCTION.....	4
1.1 DEFINITION.....	4
1.2 GLOSSARY .....	5
1.3 ARCHITECTURAL OVERVIEW .....	6
1.4 MODULES.....	7
<b>PART II: A HERAS<sup>AF</sup> SHORT STORY .....</b>	<b>8</b>
2 HERAS <sup>AF</sup> SHORT STORY .....	9
<b>PART III: ARCHITECTURE .....</b>	<b>13</b>
3 LOGICAL OVERVIEW .....	14
3.1 OVERVIEW .....	14
3.2 SEQUENCE DIAGRAM .....	15
4 LOGICAL VIEW.....	16
4.1 OVERVIEW .....	16
4.2 PACKAGE ORG.HERASAF.PEP.CORE.....	17
4.3 PACKAGE ORG.HERASAF.PEP.ANNOTATIONS .....	18
4.4 PACKAGE ORG.HERASAF.PEP.INTEGRATION.SPRINGAOP.....	19
4.5 PACKAGE ORG.HERASAF.PEP.INTEGRATION.ASPECTJ.....	20
4.6 PACKAGE ORG.HERASAF.PEP.RESOLVERS .....	21
4.7 PACKAGE ORG.HERASAF.PEP.TRANSFORMERS .....	22
5 ARCHITECTURAL DECISIONS.....	23
5.1.1 MetaDataTransferHolder.....	23
5.1.2 AnnotationActionResolver .....	23
5.2 HERASATTRIBUTECONFIGURATION .....	23
5.3 HERASATTRIBUTE.....	23
5.4 MULTIPLE RESOURCES AND SUBJECTS.....	24



# Part I: Introduction



# 1 Introduction

## 1.1 Definition

- What is HERAS<sup>AF</sup> PEP* HERAS<sup>AF</sup> PEP is part of the HERAS<sup>AF</sup>. It is responsible to intercept actions, create a RequestCtx and send that to the PDP. Afterwards it must interpret the response and grant or deny access.
- What is HERAS<sup>AF</sup> PDP* HERAS<sup>AF</sup> PDP is part of HERAS<sup>AF</sup>. It is responsible to evaluate requests and grant access based on stored policies. The HERAS<sup>AF</sup> PDP offers a web service to the HERAS<sup>AF</sup> PEP. It is responsible for receiving requests from a HERAS<sup>AF</sup> PEP, evaluating the request and sending a response back to the PEP.
- What is XACML* XACML stands for eXtensible Access Control Markup Language. It is a declarative access control policy language implemented in XML and a processing model, describing how to interpret the policies.
- What are annotations* *Annotations* provide data about a program that is not part of the program itself. They have no direct effect on the operation of the code they annotate. Annotations can be applied to a program's declarations of classes, fields, methods, and other program elements.<sup>1</sup>
- What is AOP* *Aspect-Oriented Programming (AOP)* complements Object-Oriented Programming (OOP) by providing another way of thinking about program structure. In addition to classes, AOP gives you *aspects*. Aspects enable modularization of concerns such as transaction management that cut across multiple types and objects. (Such concerns are often termed *crosscutting* concerns.)<sup>2</sup>
- What is SpringAOP* The Spring AOP framework is interception-based and configured at runtime. This removes the need for a special compilation step or load-time weaving. On the other hand interception only allows for public or protected method executions on existing object as join point. The trade-off between simplicity and available features is an important one as Spring AOP is reasonably powerful yet remains reasonably easy to learn.<sup>3</sup>

<sup>1</sup> <http://java.sun.com/docs/books/tutorial/java/javaOO/annotations.html>

<sup>2</sup> <http://www.springframework.org/docs/reference/aop.html>

<sup>3</sup> <http://de.wikipedia.org>



*What is AspectJ*      *AspectJ* is an aspect-oriented extension to the Java programming language created at Xerox PARC and now available in Eclipse Foundation open-source projects, both stand-alone and integrated into Eclipse. AspectJ has become the widely-used de-facto standard for AOP by emphasizing simplicity and usability for end users. It uses Java-like syntax and has included IDE integrations for displaying crosscutting structure since its initial public release in 2001.<sup>4</sup>

## 1.2 Glossary

<i>HERAS<sup>AF</sup></i>	Holistic Enterprise Ready Application Security <small>Architecture Framework</small>
<i>XACML</i>	eXtensible Access Control Markup Language
<i>Action</i>	An operation on a resource
<i>Resource</i>	Data, service or system component
<i>Subject</i>	An actor whose attributes may be referenced by a predicate
<i>Environment</i>	The set of attributes that are relevant to an authorization decision and are independent of a particular subject, resource or action
<i>Attribute</i>	Characteristic of a subject, resource, action or environment that may be referenced in a predicate or target
<i>Decision request</i>	The request by a PEP to a PDP to render an authorization decision

---

<sup>4</sup> <http://de.wikipedia.org>

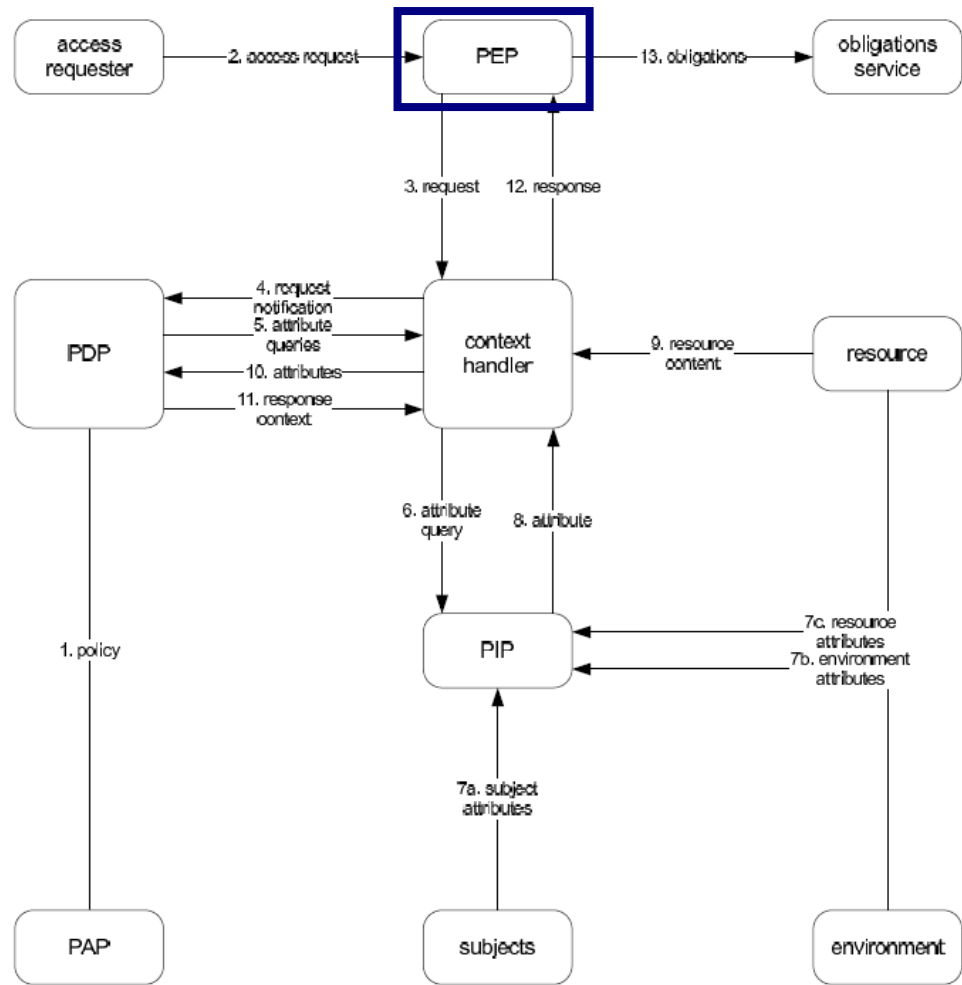


### 1.3 Architectural Overview

*Overview*

The major actors in the XACML Domain are shown in the data flow diagram. This document covers the software architecture of the policy enforcement point (PEP).

*data flow diagram XACML*



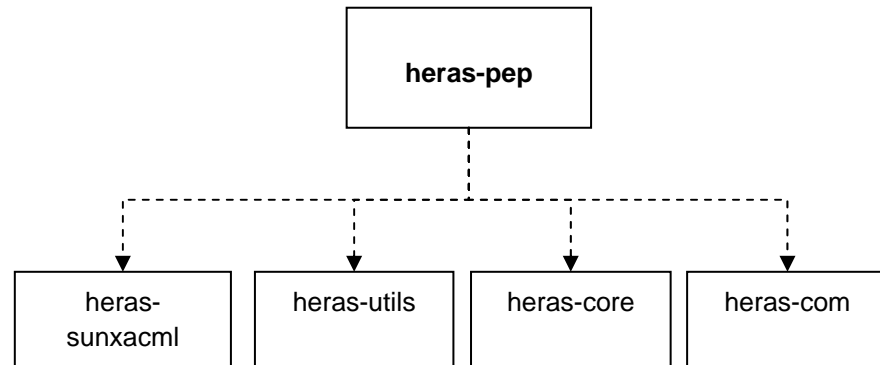
**Figure 1** data flow diagram



## 1.4 Modules

### *Module dependencies*

The following Figure 2 shows the module dependencies of the heras-pep module. Each depended module is described later in the chapter.



**Figure 2** module dependencies

### *heras-pep*

PEP is a component of policy-based management. When a user tries to access a file or other resource on a computer network or server that uses policy-based access management, the PEP will describe the user's attributes to other entities on the system. The PEP will give the Policy Decision Point (PDP) the job of deciding whether or not to authorize the user based on the description of the user's attributes. Applicable policies are stored on the system and are analyzed by the PDP. The PDP makes its decision and returns the decision. The PEP will let the user know whether or not he has been authorized to access the requested resource.

### *heras-sunxacml*

heras-sunxacml contains the integration code for the sun XACML implementation.

### *heras-utils*

heras-utils contains utilities and classes which are use by different modules. SpringUtils for example is located in the heras-util module.

### *heras-core*

Heras-core contains the structural model, the core functionality as well as the HERAS<sup>AF</sup> logic and whose implementation.



## Part II: A HERAS<sup>AF</sup> Short Story



## 2 HERAS<sup>AF</sup> Short Story

### Introduction

Dr. Jill Valentine wants to check the patient record from Barry Burton, which are the Boyfriend of her daughter Caroline Valentine and actually a patient of Dr. Chris Redfield. So Jill clicks on the button that says, "Search all Patients" in the patient record management application. The Application returns a button that says: "patient record from Barry Burton". That's what she wants. She wants to find out if Barry actually takes drugs, because she wants to protect her daughter. As she clicks on the button to Berry's patient record the `PatientRecordDao.retrieveRecord(String patientName)` method will be called and the request is being intercepted by the HERAS `AccessControlInterceptor(Spring AOP)`, which is configured in `heras-springaop.ctx.xml`. Different pointcuts are defined in this configuration file. E.g. an `AnnotationPointcut`, which defines, that only methods of a Class which is annotated with `@Protected` will be intercepted. The reason for the interception is that the `PatientRecordDao` interface is annotated as follows:

### Protected Interface

```
@Protected
@Attribute (category = Category.RESOURCE, id="urn:smurve-
clinic:java:architecture:dao" )
public interface PatientRecordDao {

    @Attributes( {
        @Attribute (category=Category.RESOURCE,
            id="urn:smurve-clinic:resource:name" ),
        @Attribute (category=Category.ACTION,
            id="urn:smurve-clinic:action:crud-type" ,value="read" )
    })

    public PatientRecord retrieveRecord (
        @Attribute (category = Category.RESOURCE,
            name="urn:smurve-clinic:patient" )
        String patientId );

}
```

This means:

- Interface `PatientRecordDao` is supposed to be protected by HERAS<sup>AF</sup>  
The Spring AOP is configured to intercept any method invocation on classes annotated with `@Protected`.
- The entire DAO architectural layer is considered a resource. This is only one possible interpretation.
- There is one attribute to come with the resource and that attribute has the name `"urn:smurve-clinic:resource:name"` and the value on method level is by default the name of the method. In this example `"retrieveRecord"`.
- There is an Action attribute with the name `"urn:smurve-clinic:action:crud-type"` and the value `"read"`.



- There is another resource attribute with the name `"urn:smurve-clinic:patient"` and the value taken from the method parameter `patientId`.

*XACML  
Request*

**Creating the XACML Request and calling the PDP**

It is now in the responsibility of the interceptor to collect the relevant information from the AOP Alliance's `MethodInvocation`. First the interceptor will set the `MethodInvocation` in the `HerasDecisionContext`. The `HerasDecisionContext` is now in a position to collect all `Attribute` annotations found in the `MethodInvocation`. If the value of an attribute annotation is empty, the `HerasDecisionContext` uses the value of the runtime object. The dynamic value and the specific annotation technology are encapsulated in a `MetaDataTransferHolder`. In the example below the value of the resource attribute `"urn:smurve-clinic:patient"` will be the runtime value of the `patientId` parameter.

```
public PatientRecord retrieveRecord (
    @Attribute (category = Category.RESOURCE,
        name="urn:smurve-clinic:patient" )
    String patientId );
```

*Attribute  
annotations  
values*

Table 1 shows the default values on the different layers:

Layer	Default value	Example
Class	Full Class name	<code>org.smurve.clinic.EJBPatientRecordDao</code>
Interface	Full Interface name	<code>org.smurve.clinic.PatientRecordDao</code>
Method	Method name	<code>retrieveRecord</code>
Parameter	Runtime value	<code>"1040"</code>

**Table 1** Default Attribute annotation values

The `HerasDecisionContext` is now passed to the `HerasAccessDecisionManager`, which makes the decision based on the `HerasDecisionContext`.

The `HerasAccessDecisionManager` then requests the different `HerasAttributes` for subject, resource, action and environment from the `HerasDecisionContext`.

Because the patient record management application of the Clinic uses annotations the according Annotation Resolvers were injected using Spring in the `HerasDecisionContext`.

To receive the `HerasAttributes`, the `HerasDecisionContext` passes a list of `MetaDataTransferHolder` grouped by `Category` to the corresponding Resolver.

The Annotation Resolvers are now able to cast the native metadata object to a `@Attribute` annotation and take the value from the `MetaDataTransferHolder` (which may contain the dynamic value).



To create a `HerasAttribute` the Resolver performs a lookup with the id of the `@Attribute` annotation on the `HerasAttributeConfiguration`. The `HerasAttributeConfiguration` is configured using Spring and contains the conversion table between the `Attributeld` and the XACML type.

*Conversion  
table Attributeld  
XACML Type*

```

<!--
=====
HERAS Attribute Id to XACML Type
=====
-->
<bean id="attributeConfiguration"
class="org.herasaf.util.HerasAttributeConfiguration">
  <property name="attributes">
    <map>
      <entry key="urn:smurve-clinic:resource:name"

value="http://www.w3.org/2001/XMLSchema#string" />
      <entry key="urn:smurve-clinic:action:crud-type"

value="http://www.w3.org/2001/XMLSchema#string" />
      <entry key=" urn:smurve-
clinic:java:architecture:dao"
value="http://www.w3.org/2001/XMLSchema#string" />
      <entry key=" urn:smurve-clinic:patient"

value="http://www.w3.org/2001/XMLSchema#string" />
    <entry
key="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
value="urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name"
/>
    </map>
  </property>
</bean>

```

The `HerasAttributeConfiguration` now creates a skeleton of a `HerasAttribute` with the id and the type.

To complete the creation of the `HerasAttribute` the issuer and the value of the `@Attribute` annotation is set.

To resolve the Subject `HerasAttributes` and receive the authenticated user the `SubjectResolver` uses the `SecurityContextHolder` of ACEGI and reads the principal from the Authentication.

The `HerasAccessDecisionManager` is now in a position to create a `RequestCtx` by using the configured `RequestCtxFactory` with the just created `HerasAttributes`.

The `RequestCtx` is then passed to a PDP for evaluation. The PDP is configured in the Spring Application Context `heras-pdp.ctx.xml`.

*PDP  
Configuration*

```

<!--*****
Locates the actual PDP implementation within the current JVM

```



```
***** ->

<bean name="pdpLocator" class="org.herasaf.com.pdp.JVMLocalPdpLocator">

    <property name="pdp">
        <ref bean="localPdp"/>
    </property>

</bean>

    <bean name="localPdp"
class="org.herasaf.pep.demo.ReportingPdp">
    </bean>

</beans>
```

As a result of the evaluation the `HerasAccessDecisionManager` receives a `ResponseCtx`.

The `ResponseCtx` is then interpreted and an exception is thrown if required.

Because Dr. Jill Valentine doesn't have the required privileges to read the patient record of Barry Burton, an `AccessDeniedException` is thrown and the patient record management application shows an error message.

In this case the patient record management application was protected by Heras<sup>AF</sup> Security and therefore the sensitive patient data of Barry Burton were successfully secured against unauthorized access.



## Part III: Architecture

## 3 Logical Overview

### 3.1 Overview

Figure 3 provides an overview of the Heras<sup>AF</sup> PEP by describing the objects and classes inside the system and the relationships between them.

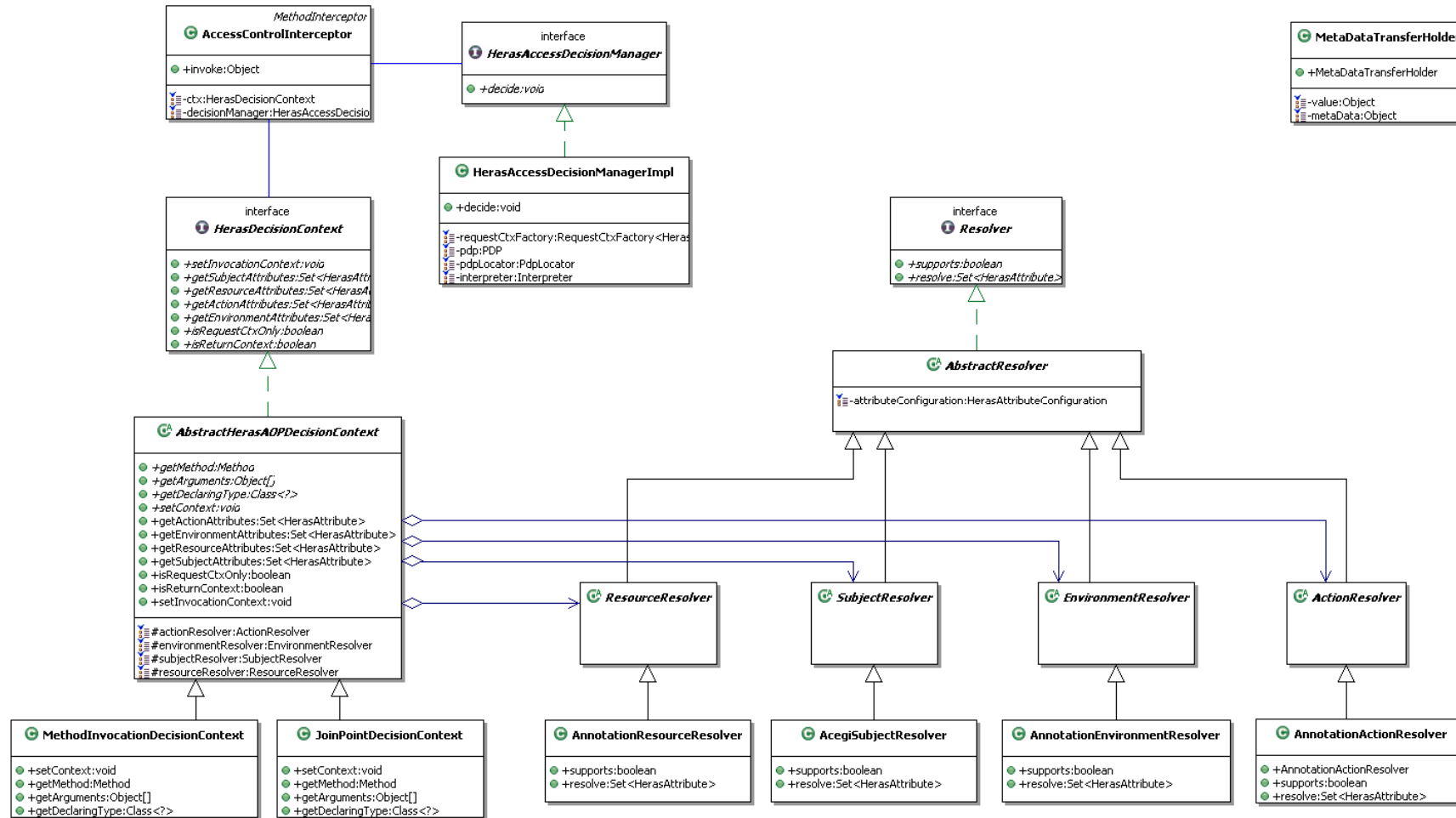


Figure 3 Logical Overview



## 3.2 Sequence Diagram

Figure 4 shows the sequence diagram of a method invocation intercepted by Heras<sup>AF</sup> Access Control Interceptor. The `AccessControlInterceptor` intercepts the Method call and first sets the AOP alliance's `MethodInvocaton` in the `HerasDecisionContext`. Then the `AccessControlInterceptors` calls the `decide` Method on the `HerasAccessDecisionManager` which makes the decision based on the `HerasDecisionContext`. To collect the annotated information, the `HerasDecisionContext` delegates the created `MetaDataTransferHolder`s to the adequate `Resolver` and receives then a `Set` of `HerasAttributes`. Finally the `HerasAccessDecisionManager` evaluates the `RequestCtx` against the `PDP`.

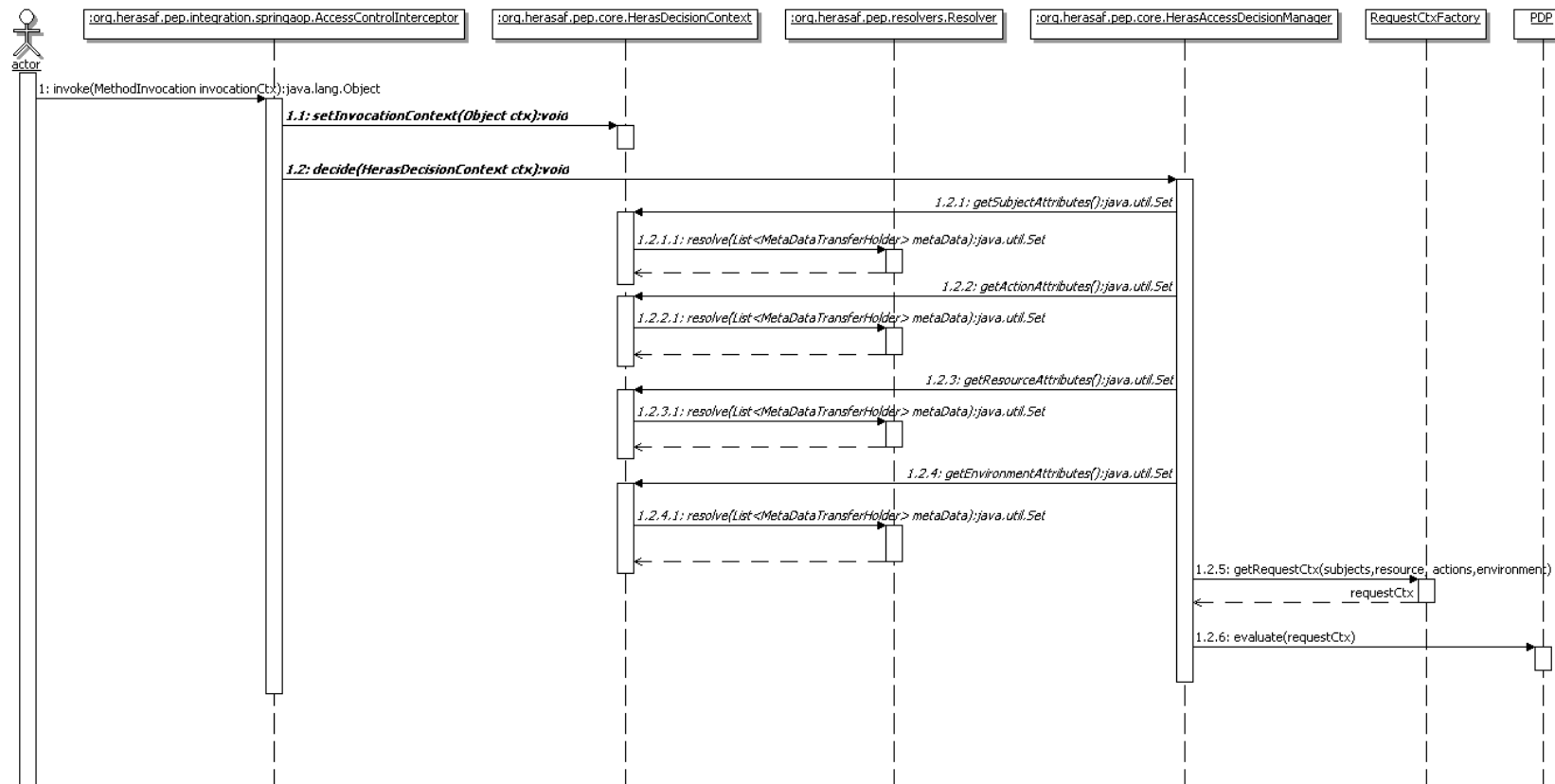


Figure 4 Sequence Diagram

## 4 Logical View

### 4.1 Overview

Package  
Diagram  
Overview

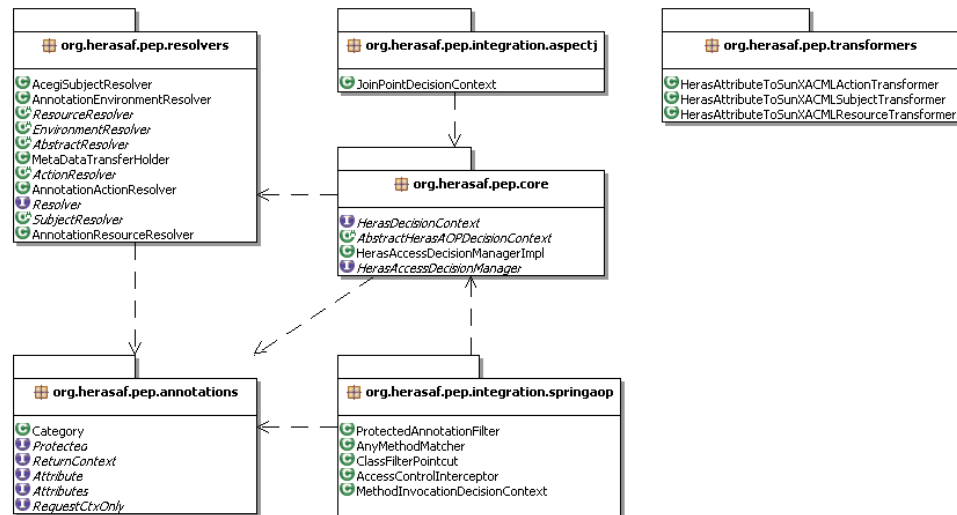


Figure 5 Heras<sup>AF</sup> PEP Packages

The Heras<sup>AF</sup> PEP is logically structure in five packages.

- The core classes and interfaces are located in the org.herasaf.pep.core package.
- The specific interceptor technologies are located in the org.herasaf.pep.integration package.
- All relevant Heras<sup>AF</sup> PEP annotations are located in the org.herasaf.pep.annotations package.
- The resolvers which are able to resolve meta data to HerasAttributes are located in the org.herasaf.pep.resolvers package.
- The transformers which generate SunXACML attributes are located in the org.herasaf.pep.transformers package



## 4.2 Package org.herasaf.pep.core

Package  
Diagram



Figure 6 org.herasaf.pep.core

org.herasaf.pep  
.core

The heras core classes and interfaces are located in the org.herasaf.pep.core package.

One of the key concepts of the Heras<sup>AF</sup> PEP is the HerasDecisionContext. The HerasDecisionContext 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.

Based on the HerasDecisionContext the HerasDecisionManager is now able to evaluate the Subject, Resource, Action and Environment attributes. Besides the HerasAccessDecisionManager generates a RequestCtx using the RequestCtxFactory and evaluates the RequestCtx against the PDP. As a result of the evaluation the HerasAccessDecisionManager gets a ResponseCtx which he then interprets and comes to a decision whether to deny or allow access to the protected resource.

The AbstractHerasAOPDecisionContext encapsulates the functionality for Aspect Oriented Programming approaches and is able to operate on java.lang.reflect.Method Objects, arrays of arguments and on java.lang.Class Objects. The AbstractHerasAOPDecisionContext is then able to gather the relevant security information in the form of annotations over Java Reflections.



## 4.3 Package org.herasaf.pep.annotations

Package  
Diagram

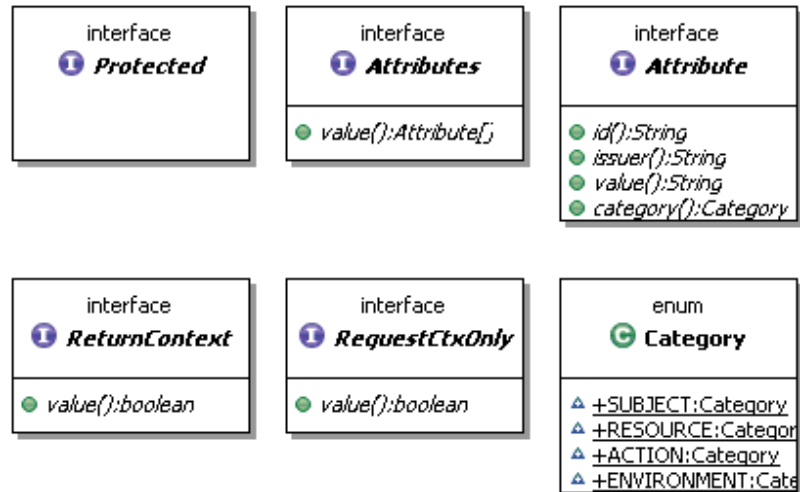


Figure 7 org.herasaf.pep.annotations

org.herasaf.pep  
.annotations

The Heras<sup>AF</sup> annotations which are used to decorate a Service Facade to enable Heras<sup>AF</sup> Security are located in the org.herasaf.pep.annotations package.

The @Protected annotation is used to mark a class or an interface to be intercepted by the AccessControlInterceptor. The class or interface may then be decorated with various @Attribute annotations. Each @Attribute belongs to a specific Category which is defined in the Category Enumeration.

The two annotations @ReturnContext and @RequestCtxOnly may only be used in conjunction with a SAML<sup>5</sup> PDP.

**Note:** Each Heras<sup>AF</sup> annotation is automatically inherited. If an Inherited meta-annotation is present on an annotation type declaration, and the user queries the annotation type on a class declaration, and the class declaration has no annotation for this type, then the class's superclass will automatically be queried for the annotation type. This process will be repeated until an annotation for this type is found, or the top of the class hierarchy (Object) is reached.

<sup>5</sup> SAML Specification



## 4.4 Package org.herasaf.pep.integration.springaop

Package  
Diagram



Figure 8 org.herasaf.pep.intergration.springaop

org.herasaf.pep  
.intergration.  
springaop

The specific interceptor technology is defined in the `org.herasaf.pep.intergration` package. The Spring AOP specific classes are hence located in `org.herasaf.pep.intergration.springaop`.

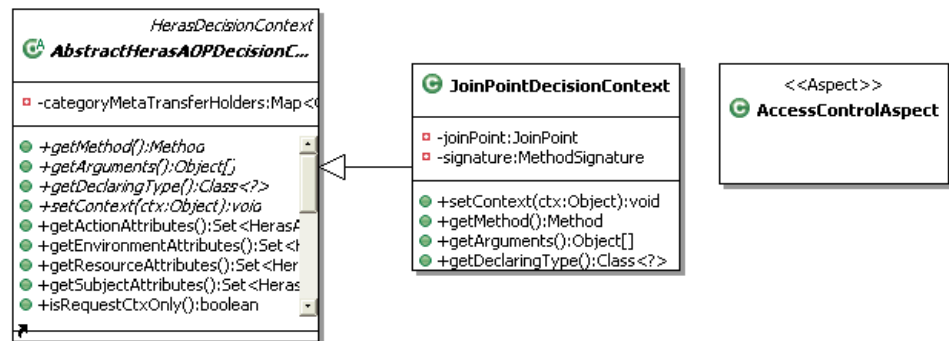
The `MethodInvocationDecisionContext` class is a subclass of the `AbstractHerasAOPDecisionContext` and is responsible to deliver the required information for the decision based on the Spring AOP `MethodInvocationContext`.

Besides the `MethodInvocationDecisionContext` collects the decorated `@Attribute` annotations, sorts them by category and hence creates `MetaDataTransferHolders` which are resolved to `HerasAttributes` by the different Resolvers.



## 4.5 Package org.herasaf.pep.integration.aspectj

Package  
Diagram



*org.herasaf.pep*  
*.integration.*  
*aspectj*

The `JoinPointDecisionContext` is a subclass of `AbstractHerasAOPDecisionContext` and is responsible to deliver the required information for the decision based on an AspectJ `JoinPoint`. The needed information used by the `AbstractHerasAOPDecisionContext` is the `Method` object, the arguments and the declaring type.

The needed information for the `AbstractHerasAOPDecisionContext` is obtained by accessing the `JoinPoint` and the `MethodSignature`.

`MethodSignature` is used to receive the `Method` object by calling `java.lang.reflect.Method getMethod()` and the declaring type by calling `java.lang.Class getDeclaringType()`.

The arguments may be directly received using `java.lang.Object[] getArgs()` of the `JoinPoint` object.



## 4.6 Package org.herasaf.pep.resolvers

Package  
Diagram

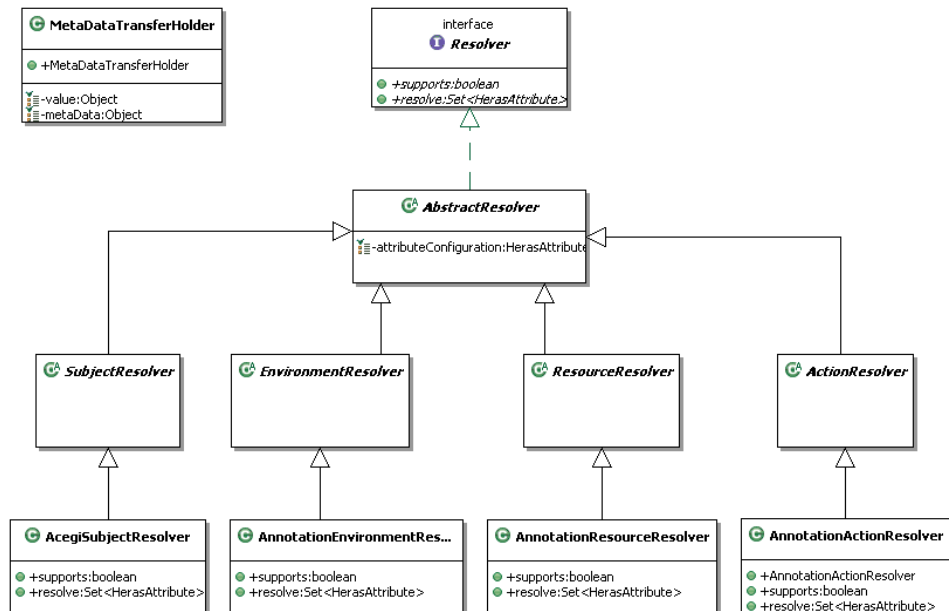


Figure 9 org.herasaf.pep.resolver

org.herasaf.pep  
.resolvers

The Resolvers are responsible to build `HerasAttributes` based on `MetadataTransferHolder` objects. Each `HerasDecisionContext` holds a Resolver for Resource, Action, Subject and Environment and delegates a Collection of `MetadataTransferHolders` to the `resolve()` Method.

A Resolver is able to resolver specific meta data to `HerasAttributes`. By using the resolvers concept new resolvers can easily integrated through the spring application context.

To support different annotation technologies (such as java annotations or an xml declarative approach) the `MetadataTransferHolder` is used. The `MetadataTransferHolder` encapsulates the concrete annotation technology and the value of the annotation. With this holder object the Heras<sup>AF</sup> PEP resolver infrastructure may be used irregardless of the concrete annotation technology.

Actually there are resolvers who may interpret Java 1.5 annotations and a subject resolver which is able to handle Acegi subjects.



## 4.7 Package org.herasaf.pep.transformers

Package  
Diagram

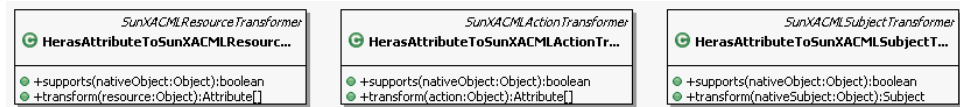


Figure 10 org.herasaf.pep.transformers

org.herasaf.pep  
.transformers

Transformers are responsible to transform HerasAttributes to SunXACML Attributes. By using transformers the specific XACML implementation may easily be replaced by another implementation. This gives the needed flexibility to work independent of the specific implementation.



## 5 Architectural Decisions

### 5.1.1 MetaDataTransferHolder

*Issue* Who is responsible to evaluate the dynamic value attribute of an `@Attribute` annotation

*Solutions* The creator of the `MetaDataTransferHolder` evaluates the value attribute

*Decision* Because the `MetaDataTransferHolder` does not know the concrete class of the metadata object, he is not in a position to decide whether to use the value attribute of the `MetaDataTransferHolder` or the value attribute of the annotation.

For this reason, the evaluation of the value attribute is done before the creation of the `MetaDataTransferHolder`. The value attribute of the `MetaDataTransferHolder` is therefore never empty and is always used to retrieve the value of the annotation.

### 5.1.2 AnnotationActionResolver

*Issue* Should an `AnnotationResolver` also support normal annotations?

*Decision* No, the different `AnnotationResolvers` only support `@Attribute` annotations.

## 5.2 HerasAttributeConfiguration

*Issue* Which information should provided in the skeleton of the `HerasAttribute` created by the `HerasAttributeConfiguration`

*Decision* Because the lookup on the `HerasAttributeConfiguration` is done with the attribute id, the created `HerasAttribute` skeleton contains the id and the XACML type.

## 5.3 HerasAttribute

*Issue* Does a `HerasAttribute` have an affiliation to a `Category`?

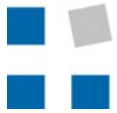
*Decision* No, a `HerasAttribute` does not have an affiliation to a `Category` (Action, Resource, Subject, and Environment). This is not necessary because the `HerasAttributes` are always handled as collection of a specific category.



## 5.4 Multiple Resources and Subjects

*Issue* Must the Heras<sup>AF</sup> PEP support multiple Resources and Subjects?

*Decision* Actually the Heras<sup>AF</sup> PEP does not support multiple Resources or Subjects. Because the Heras<sup>AF</sup> PEP should be as small as possible this feature is not needed.



HSR  
HOCHSCHULE FÜR TECHNIK  
RAPPERSWIL



**Studienarbeit**

# **HERAS<sup>AF</sup> PEP Sitzungsprotokolle**

Abteilung Informatik  
Hochschule für Technik Rapperswil

Sommersemester 2007

Studenten:  
Verantwortlicher Dozent:

Sandro Strebel, Massimo Cerqui  
Wolfgang Giersche



## Inhalt

<b>1</b>	<b>Sitzungsprotokolle</b>	<b>3</b>
1.1	Kickoff Meeting vom 02.04.2007	3
1.2	Status Meeting Iteration 2 vom 17.04.2007	4
1.3	Status Meeting Iteration 2 vom 8.05.2007	5
1.4	Status Meeting Iteration 3 vom 11.05.2007	6
1.5	Status Meeting Iteration 4 vom 15.06.2007	8



# 1 Sitzungsprotokolle

## 1.1 Kickoff Meeting vom 02.04.2007

### 1.1.1 Sitzungsinformationen

**Sitzungsdatum:** 02.04.2007  
**Anwesend:** Wolfgang Giersche, René Eggenschwiler, Massimo Cerqui, Sandro Strebel, Sacha Dolski, Florian Huonder, Stefan Oberholzer,  
**Ort:** Mensa, HSR  
**Beginn:** 16:45  
**Ende:** 17:40  
**Protokollführer:** Sandro Strebel

### 1.1.2 Traktanden

Traktandum	Auftrag / Information	Zuständig	Bemerkungen
Herausforderung der Studienarbeit	Methodenaufrufen abfangen, nötige Zusatzinformationen bereitstellen um Request zu bilden		
Vorgehen erste zwei Wochen	<ul style="list-style-type: none"> <li>▪ Möglichkeiten für konzeptionelle Mappings suchen</li> <li>▪ Alternativen zu Annotations untersuchen</li> <li>▪ Designvorschlag erstellen</li> </ul>		<p>Wie werden Action, Resource, Subject und Environment gemappt</p> <p>Strategy Pattern verwenden Test Cases schreiben</p>
Geforderte Dokumente	<ul style="list-style-type: none"> <li>▪ Working Code mit wohldokumentierten Testcases</li> <li>▪ HSR Bericht (Vorgehensweise)</li> <li>▪ User Guide in Englisch <ul style="list-style-type: none"> <li>○ Äussere Seite</li> <li>○ Für Kunde</li> <li>○ Minimal technisch</li> </ul> </li> <li>▪ Developer Guide <ul style="list-style-type: none"> <li>○ Innere Sicht</li> <li>○ Analyse</li> <li>○ Design</li> <li>○ Details</li> <li>○ Entscheide, Konzepte</li> </ul> </li> </ul>		<ul style="list-style-type: none"> <li>○ Zeitauswertung kann mit Online Tool „Achievo“ geführt werden</li> <li>○ Projektplan in MS Project sinnvoll</li> </ul>

### 1.1.3 Nächste Sitzung

**Ort:** HSR  
**Datum:** 13.04.2007  
**Beginn:** 10:00  
**Bereits bekannte Traktanden:** kurze Präsentation, 10 min 2 Slides.  
Was vorgenommen und was erreicht.



## 1.2 Status Meeting Iteration 2 vom 17.04.2007

### 1.2.1 Sitzungsinformationen

**Sitzungsdatum:** 04.05.2007  
**Anwesend:** Wolfgang Giersche, René Eggenschwiler, Massimo Cerqui, Sandro Strebel  
**Ort:** Dozentenraum, HSR  
**Beginn:** 13:00  
**Ende:** 15:00  
**Protokollführer:** Sandro Strebel

### 1.2.2 Traktanden

Traktandum	Auftrag / Information	Zuständig	Bemerkungen
Präsentation der Iteration 2	<ul style="list-style-type: none"> <li>▪ Die Ergebnisse der ersten Iteration wurden präsentiert</li> </ul>		
Spring AOP	<ul style="list-style-type: none"> <li>▪ Class Filter könnte mit einem Annotation Filter ersetzt werden</li> </ul>		Somit wäre @Protected nicht mehr zwingend nötig
Ausblick	<ul style="list-style-type: none"> <li>▪ Refactoring</li> <li>▪ Code „sauber machen“</li> <li>▪ Exception Hierarchie beschreiben</li> <li>▪ Spring Configuration beschreiben</li> <li>▪ User Story für Developer Guide erstellen</li> <li>▪ AspectJ anschauen</li> </ul>		

### 1.2.3 Nächste Sitzung

**Ort:** Dozentenraum HSR  
**Datum:** 11.05.2007  
**Beginn:** Noch unbekannt



## 1.3 Status Meeting Iteration 2 vom 8.05.2007

### 1.3.1 Sitzungsinformationen

**Sitzungsdatum:** 04.05.2007  
**Anwesend:** Wolfgang Giersche, René Eggenschwiler, Massimo Cerqui, Sandro Strebel  
**Ort:** Dozentenraum, HSR  
**Beginn:** 13:00  
**Ende:** 15:00  
**Protokollführer:** Sandro Strebel

### 1.3.2 Traktanden

Traktandum	Auftrag / Information	Zuständig	Bemerkungen
Präsentation der Iteration 2	<ul style="list-style-type: none"> <li>▪ Die Ergebnisse der ersten Iteration wurden präsentiert</li> </ul>		
Sping AOP	<ul style="list-style-type: none"> <li>▪ Class Filter könnte mit einem Annotation Filter ersetzt werden</li> </ul>		Somit wäre @Protected nicht mehr zwingend nötig
Ausblick	<ul style="list-style-type: none"> <li>▪ Refactoring</li> <li>▪ Code „sauber machen“</li> <li>▪ Exception Hierarchie beschreiben</li> <li>▪ Spring Configuration beschreiben</li> <li>▪ User Story für Developer Guide erstellen</li> <li>▪ AspectJ anschauen</li> </ul>		

### 1.3.3 Nächste Sitzung

**Ort:** Dozentenraum HSR  
**Datum:** 11.05.2007  
**Beginn:** Noch unbekannt



## 1.4 Status Meeting Iteration 3 vom 11.05.2007

### 1.4.1 Sitzungsinformationen

**Sitzungsdatum:** 11.05.2007  
**Anwesend:** Wolfgang Giersche, René Eggenschwiler, Massimo Cerqui, Sandro Strebel, Stefan Dolski, Florian Huonder  
**Ort:** Dozentenraum, HSR  
**Beginn:** 13:00  
**Ende:** 15:00  
**Protokollführer:** Sandro Strebel

### 1.4.2 Traktanden

Traktandum	Auftrag / Information	Zuständig	Bemerkungen
Stand der Arbeit	<ul style="list-style-type: none"> <li>▪ Die Ergebnisse der ersten Iteration wurden präsentiert</li> </ul>		
Anpassungen RequestCtxFactory	<ul style="list-style-type: none"> <li>▪ Die SunXACML Implementation unterstützt nicht vollständig den XACML Standard. So ist es nicht möglich mehrere Ressourcen in SunXACML zu definieren. Aus diesem Grund ist eine Anpassung der Heras RequestCtxFactory notwendig.</li> </ul>		
Anpassung RequestCtxFactory Interface	<ul style="list-style-type: none"> <li>▪ Das Heras RequestCtxFactory Interface muss entsprechend angepasst werden</li> </ul>	PDP	
Anpassung SunXACMLRequestCtxFactory	<ul style="list-style-type: none"> <li>▪ Die SunXACML Implementation der RequestCtxFactory muss entsprechend angepasst werden</li> </ul>	PEP	
Gemäss der SAML Spezifikation muss der Client dem PDP zwei weitere Parameter mitteilen können: <ul style="list-style-type: none"> <li>- RequestCtxOnly</li> <li>- ReturnContext</li> </ul>	<ul style="list-style-type: none"> <li>▪ Um diese zwei neuen Parameter dem PEP mitzuteilen, werden zwei neue Annotations eingeführt.</li> <li>▪ @RequestCtxOnly</li> <li>▪ @ReturnContext</li> </ul>	PEP	Der DecisionContext wird um zwei neue Methoden erweitert: <ul style="list-style-type: none"> <li>- isRequestCtxOnly()</li> <li>- isReturnContext()</li> </ul>



Anpassung SAML PDP	<ul style="list-style-type: none"><li>Die PDP Gruppe liefert eine entsprechende Schnittstelle zu einem SAML PDP welche im PEP konfiguriert wird.</li></ul>	PDP	
--------------------	--	-----	--

### 1.4.3 Nächste Sitzung

**Ort:** Dozentenraum HSR  
**Datum:** 1.06.2007  
**Beginn:** Noch unbekannt



## 1.5 Status Meeting Iteration 4 vom 15.06.2007

### 1.5.1 Sitzungsinformationen

**Sitzungsdatum:** 15.06.2007  
**Anwesend:** Wolfgang Giersche, René Eggenschwiler, Massimo Cerqui, Sandro Strebel  
**Ort:** Dozentenraum, HSR  
**Beginn:** 13:00  
**Ende:** 15:00  
**Protokollführer:** Massimo Cerqui

### 1.5.2 Traktanden

Traktandum	Auftrag / Information	Zuständig	Bemerkungen
Stand der Arbeit	<ul style="list-style-type: none"> <li>▪ Die Ergebnisse der Iteration wurden präsentiert</li> </ul>		
Anpassungen Developer Guide	<ul style="list-style-type: none"> <li>▪ Das Kapitel Design Context soll zu Architektur umbenannt werden</li> </ul>		
Anpassung Developer & User Guide	<ul style="list-style-type: none"> <li>▪ Die Diagramme sollen vergrößert auf je einer ganzen Seite dargestellt werden.</li> </ul>		
Anpassung Tests	<ul style="list-style-type: none"> <li>▪ Es soll je einen separaten Test für die Verwendung der beiden Technologien( AspectJ &amp; Spring AOP) erstellt werden.</li> <li>▪ Es soll ein Integration Test erstellt werden, welcher einmal Ja und einmal Nein als ergebniss liefert.</li> </ul>		