



Diplomarbeit  
**HERAS<sup>AF</sup>: XACML 2.0 Implementierung**  
Hauptdokument

Abteilung Informatik  
Hochschule für Technik Rapperswil

Herbstsemester 07/08

Studenten: Sacha Dolski, Florian Huonder, Stefan Oberholzer  
Verantwortlicher Dozent: Wolfgang Giersche  
Betreuer: René Eggenschwiler



## Abstract

### Übersicht

Um die Produktivität zu erhöhen und Kosten zu senken, entstehen immer mehr sogenannte virtuelle Unternehmen. Dies zwingt Firmen ihre Ressourcen und Informationen zu öffnen. Oft sind Benutzerinformationen über verschiedene Systeme im Unternehmen verteilt. Jede Applikation hat ihre eigene Benutzerverwaltung und Zugriffskontrolle. Dadurch wird das Durchsetzen von firmenbezogenen Sicherheitsrichtlinien erschwert und eine Unternehmensübergreifende Autorisierung ist kaum möglich. HERAS<sup>AF</sup> nimmt die Herausforderung an, verteilte Autorisierung unternehmenstauglich zu machen.

### Ziel der Arbeit

Implementierung der XACML 2.0 Spezifikation in Java 5. Dies beinhaltet folgende Punkte:

- Erstellung einer API mit Konfigurationsmöglichkeit der wichtigen Komponenten.
- Design, Implementierung und Testen des Policy Decision Points mit Indexierung der persistenten Richtlinien.
- Developer's Guide in Englisch.

### Lösung

Die XACML 2.0 Implementierung von HERAS<sup>AF</sup> beinhaltet alle zwingenden Elemente und Funktionen, welche die Spezifikation fordert. Für das möglichst performante Auffinden von Richtlinien wurden Indexierungsalgorithmen realisiert. Alle Komponenten sind konfigurier- und erweiterbar. Das Design ist darauf ausgelegt, dass alle Komponenten mit geringem Aufwand ausgetauscht werden können. Die Implementierung erfüllt alle Konformitätsanforderungen der Spezifikation.



## Erklärung

Wir versichern hiermit, dass wir die vorliegende Arbeit selbstständig verfasst und keine anderen, als die im Literaturverzeichnis angegebenen Quellen benutzt haben.

Die Abbildungen in dieser Arbeit sind von uns selbst erstellt worden oder mit einem entsprechenden Quellennachweis versehen.

In Stellen, welche wörtlich aus veröffentlichten oder noch nicht veröffentlichten Quellen entnommen worden sind, wird auf das Quelldokument referenziert.

Diese Arbeit ist in gleicher oder ähnlicher Form noch bei keiner anderen Prüfungsbehörde eingereicht worden.

Copyright © 2006 Hochschule für Technik, Rapperswil (HSR).

Die HSR behält sich alle Rechte an der vorliegenden Arbeit und den Dokumenten vor. Jede weitergehende Nutzung bedarf der ausdrücklichen vorherigen schriftlichen Genehmigung der Autoren oder der HSR.



## Inhaltsverzeichnis

<b>TEIL I: EINFÜHRUNG</b>	<b>6</b>
<b>1 MANAGEMENT SUMMARY</b>	<b>7</b>
1.1 AUSGANGSLAGE	7
1.2 DURCHFÜHRUNG DER ARBEIT	8
1.3 ERREICHTE ZIELE	9
1.4 AUSBLICK	9
<b>2 AUFGABENSTELLUNG</b>	<b>10</b>
2.1 ZIELSETZUNG	10
2.2 TERMINE	11
<b>3 XACML</b>	<b>12</b>
3.1 HINTERGRUND	12
3.2 AUFBAU	14
3.3 VOR- UND NACHTEILE VON XACML	18
3.4 ZUKUNFT VON XACML	19
<b>4 HERAS<sup>AF</sup></b>	<b>20</b>
4.1 HINTERGRUND	20
4.2 AUFBAU	23
4.3 HERAS <sup>AF</sup> XACML 2.0 IMPLEMENTIERUNG	26
<b>TEIL II: TECHNISCHER BERICHT HERAS<sup>AF</sup> XACML 2.0 IMPLEMENTIERUNG</b>	<b>28</b>
<b>1 ANALYSE</b>	<b>29</b>
1.1 ANFORDERUNGEN	29
1.2 SCHWERPUNKTE UND HERAUSFORDERUNGEN	32
1.2.1 Context und Policy Klassen	32
1.2.2 Indexierung	33
1.2.3 Entscheidungsfindung	35
1.2.4 Auflösen der Policyreferenzen	36
1.2.5 Verhalten bei fehlenden Attributen	38
1.2.6 Abstraktion der Datenhaltung	39
1.2.7 Möglichkeiten von Konfigurationsfiles	40
1.2.8 Fehler in der XACML 2.0 Spezifikation	41
1.3 PROBLEMATIK VON STATISTIKEN	43
1.4 ANALYSE DER INDEXIERUNGSLGORITHMEN	44
1.5 ANALYSE SUNXACML	45
<b>2 ARCHITEKTUR</b>	<b>48</b>
<b>3 DESIGN</b>	<b>51</b>
3.1 MODUL CORE	51
3.1.1 Herausforderungen	53
3.2 MODUL PDP	55
3.2.1 Herausforderungen	57
3.3 MODUL PERSISTENCE	57
3.3.1 Herausforderungen	58
3.4 MODUL LOCATOR	59
3.4.1 Herausforderungen	60
3.4.1.1 Indexierungsalgorithmen	61
3.5 MODUL REFERENCELOADER	65
<b>4 TESTS</b>	<b>66</b>



4.1	INTEGRATIONS-TESTS	66
4.1.1	Konformitäts-Tests	66
4.1.2	Last-Tests	69
4.1.2.1	Test Szenario 1	69
4.1.2.2	Test Szenario 2	70
4.1.2.3	Test Szenario 3	71
4.1.2.4	Test Szenario 4	72
4.1.3	Modul Integration-Tests	73
4.2	UNIT-TESTS	73
<b>5</b>	<b>SCHLUSSBERICHT</b>	<b>74</b>
5.1	ZUSAMMENFASSUNG	74
5.2	UMFANG	75
5.3	ABWEICHUNGEN ZUR XACML 2.0 SPEZIFIKATION	75
5.4	AUSBLICK	76
<b>ANHANG I : PROJEKTMANAGEMENT</b>		<b>77</b>
<hr/>		
<b>1</b>	<b>PROJEKTPLANUNG</b>	<b>78</b>
1.1	EINFÜHRUNG	78
1.2	PROJEKTÜBERSICHT	78
1.3	PROJEKTPLAN	79
1.4	RISIKOMANAGEMENT	80
1.5	QUALITÄTSMASSNAHMEN	81
1.6	VERWENDETE TECHNOLOGIE	81
1.7	VERWENDETE STANDARDS	82
1.8	SCHLUSSBERICHT PROJEKTMANAGEMENT	82
<b>2</b>	<b>AUSWERTUNG</b>	<b>83</b>
2.1	ARCHITEKTUR UND CODE QUALITÄT	83
2.2	ZEIT AUSWERTUNG	85
<b>ANHANG II : ALLGEMEIN</b>		<b>86</b>
<hr/>		
<b>1</b>	<b>PERSÖNLICHE BERICHTE</b>	<b>87</b>
1.1	SACHA DOLSKI	87
1.2	FLORIAN HUONDER	87
1.3	STEFAN OBERHOLZER	88
<b>2</b>	<b>GLOSSAR</b>	<b>89</b>
<b>3</b>	<b>LITERATURVERZEICHNIS</b>	<b>90</b>
3.1	SPEZIFIKATIONEN UND STANDARDS	90
3.2	ARTIKEL UND ARBEITEN	90
3.3	SONSTIGES	91



# Teil I: Einführung



# 1 Management Summary

## 1.1 Ausgangslage

### *Motivation*

Zentrale Richtlinienverwaltung und Zugriffssteuerung sind noch nicht weit verbreitet. Die meisten kommerziellen und nicht gewerblichen Anwendungen verfügen über ihre eigenen Benutzerspeicher und Zugriffssteuerungsmechanismen. Quelloffene, standardisierte Lösungen existieren noch nicht.

Häufig führt diese Situation zu zusätzlichen Bemühungen, Kosten und Sicherheitsgefahren. Es kann sogar zu Inkonsistenzen kommen, in welchen verschiedene Anwendungen unterschiedliche Niveaus des Zugangs zur gleichen Ressource gewähren oder eine schützenswerte Ressource nicht geschützt wird. Ebenfalls können Synchronisationen zwischen unterschiedlichen Anwendungen zu überflüssigen Richtlinieninformationen führen.

Ein zentrale und konsistente Richtlinienverwaltung und Zugriffssteuerung ist daher einer der ersten Grundsteine, um Gefahr, Kosten und Bemühungen zu verringern.

HERAS<sup>AF</sup> ist der Versuch, eine Lösung mit frei verfügbaren und quelloffenen Bestandteilen zu realisieren.

Dabei macht sich HERAS<sup>AF</sup> den von OASIS spezifizierten Standard XACML 2.0 zu Nutzen. Die Analyse einer bestehenden Implementierung dieses Standards ergab, dass einige wichtige Funktionen nicht funktionstüchtig, beziehungsweise noch nicht implementiert sind. Auf Grund dessen entstand die Idee eine eigene XACML 2.0 Implementierung zu entwickeln.

Eine weitere Motivation bildet die Absicht von namhaften Firmen, wie BEA Systems, IBM, Oracle usw. die Interoperabilität mit XACML zu demonstrieren[PressInterop].

Eine Implementierung von XACML 2.0 könnte das Interesse einer dieser Firmen für das Projekt HERAS<sup>AF</sup> wecken, was bewirken könnte, dass der Bekanntheitsgrad von HERAS<sup>AF</sup> steigt.

### *Bestehendes*

Es existiert eine XACML 2.0 Implementierung von Sun. Teile dieser Implementierung wurden in der Studienarbeit [DOH07SA] verwendet. Während dieser Arbeit wurden einige Fehler der Implementierung aufgezeigt. Die HERAS<sup>AF</sup> XACML 2.0 Implementierung ist eine komplette Neuimplementierung der XACML 2.0 Spezifikation.

### *Ziele der HERAS<sup>AF</sup> XACML 2.0 Implementierung*

- Pflichtmässige Elemente der XACML 2.0 Spezifikation abdecken.
- Schnelles Auffinden der anwendbaren Richtlinien durch geeignete Indexierungsalgorithmen
- Konfigurierbare Indexierung
- Modularer Aufbau
- In Memory Richtlinienverwaltung
- Erweiterbarkeit von Funktionen, Datentypen und CombiningAlgorithmen
- Einfache Konfigurierbarkeit



## 1.2 Durchführung der Arbeit

Involvierte Personen	Person	Funktion
	Wolfgang Giersche	Verantwortlicher Dozent
	René Eggenschwiler	Betreuer
	Sacha Dolski	Chefentwickler
	Florian Huonder	Architekt
	Stefan Oberholzer	Projektleiter

Die Funktionen Projektleiter, Chefentwickler und Architekt zeigen lediglich die Verantwortung, nicht aber die Zuständigkeit. Alle drei Funktionsverantwortlichen sind gemeinsam für die Ausführung der Arbeit zuständig.

*Einarbeit* Das Einarbeiten bestand im Wesentlichen darin, die XACML 2.0 Spezifikation durchzulesen und den Wissensstand wieder aufzufrischen.

*Vorgehen* Die Arbeit wurde in fünf Phasen unterteilt. Darunter waren zwei Entwicklungsphasen zu je zwei Wochen, zwei Dokumentationsphasen welche eine, respektive zwei Wochen andauerten, sowie die einwöchige Finishing Phase.

Die fünf Phasen sind im folgenden aufgelistet und kurz beschrieben:

### Entwicklungsphase 1 (Iteration 1)

In der ersten Woche der Entwicklungsphase wurde JAXB 2.1 untersucht und die Datenklassen von XACML 2.0 generiert. Zudem wurde die Persistenzschicht implementiert.

In der zweiten Woche wurden die Indexierungsalgorithmen für das performante Auffinden potentieller Richtlinien entwickelt.

### Dokumentationsphase 1 (Iteration 2)

In der ersten Dokumentationsphase wurde die geleistete Arbeit der ersten Entwicklungsphase dokumentiert.

### Entwicklungsphase 2 (Iteration 3)

In der ersten Woche der zweiten Entwicklungsphase wurde die Entscheidungsfindung im PDP entwickelt. Dies deckt die Kombinationsalgorithmen, die ExpressionTypes, sowie das Matchen von Richtlinien auf Anfragen ab.

Die zweite Woche diente dazu um die Funktionen der XACML 2.0 Spezifikation zu implementieren und die Konformität der HERAS<sup>AF</sup> XACML 2.0 Implementierung zu belegen. Des Weiteren wurde der Programmcode überarbeitet.



### Dokumentationsphase 2 (Iteration 4)

In der zweiten Dokumentationsphase wurde die geleistete Arbeit der zweiten Entwicklungsphase, sowie die komplette Dokumentation überarbeitet. Dies beinhaltet auch das Dokumentieren des Codes. Des Weiteren wurden in dieser Phase der Integrationspfad in den HERAS<sup>AF</sup> PDP Webservice Endpoint [DOH07SA] aufgezeigt.

### Finishing (Iteration 5)

In der abschliessenden Phase wurde die Dokumentation nochmals überarbeitet und anschliessend ausgedruckt und gebunden. Zudem wurde das Plakat für die Präsentation erstellt.

## 1.3 Erreichte Ziele

*Ergebnisse* Das Produkt dieser Arbeit ist eine funktionierende Implementierung der XACML 2.0 Spezifikation. Diese beinhaltet alle pflichtmässigen Elemente der Spezifikation.

Mit der entwickelten Implementierung ist es möglich, anhand von definierten Richtlinien, eingehende Evaluierungsanfragen auszuwerten und eine entsprechende Antwort zurück zu geben. Damit können Ressourcen, mit sinnvoll definierten Richtlinien, geschützt werden.

Eine Autorisierungslösung muss das Ziel verfolgend möglichst performant zu sein, so dass für allfällige Benutzer keine wahrnehmbaren Wartezeiten entstehen. Um einer solchen Anforderung gerecht zu werden, wurden Indexierungsalgorithmen entwickelt. Diese unterstützen das performante Auffinden potentieller Richtlinien. Die Indexierung wurde dabei konfigurierbar gestaltet, um bestmögliche Voraussetzungen für den Einsatz in verschiedenen Unternehmen zu bieten.

- Gelerntes*
- Die Herausforderung der Implementierung eines zukunftssträchtigen Standards
  - Wertvolle Erfahrung im Bereich Software Engineering
  - Erfahrungen im Erstellen von komplexen Algorithmen
  - Arbeit im Team
  - Erweiterte Erfahrung mit Technologien, wie JAXB 2.1, TestNG, Maven 2 und Spring 2.0

## 1.4 Ausblick

*Folgearbeit* Was in einer Folgearbeit noch erledigt werden könnte, ist im Kapitel 5.4 des Technischen Berichts beschrieben.



## 2 Aufgabenstellung

### 2.1 Zielsetzung

Ziel der Diplomarbeit HERAS<sup>AF</sup> XACML 2.0 Implementierung ist eine weitestgehend vollständige Implementierung der OASIS Spezifikation XACML 2.0 in Java 5. Dies beinhaltet folgende Punkte:

- Erstellung der API und Generierung der, durch die XACML Spezifikation definierten, Datenklassen mit JAXB 2.1.
- Alle wichtigen Komponenten sollen mit Spring konfigurierbar sein.
- Design, Implementierung und Testen eines Policy Decision Point der möglichst viele der definierten Funktionen der XACML 2.0 Spezifikation implementiert. Der Policy Decision Point soll über folgende Funktionen verfügen:
  - Persistierung der Datenklassen auf ein quelloffenes RDBMS-Produkt (etwa MySQL 5.0 oder Postgres)
  - Indexierung der Policies. Dafür ist anzunehmen, dass die Erstellzeit des Indexes sowie dessen Speicherbedarf keine Rolle spielt. Wesentlich ist lediglich, dass die Abfrage mit maximaler Performanz stattfindet.
  - Auflösung von Attributen und Policyreferenzen als Mock eingebunden.

Zusätzlich, zu der von der HSR eingeforderten Dokumentation und dem Programmcode, soll ein englischsprachiges Programmierhandbuch erstellt werden. Das Programmierhandbuch soll die Verwendung der API zum Bau einer eigenständigen Applikation unterstützen.

Das Programmierhandbuch soll folgende Themen abdecken:

- Konfiguration der Module
- Konfigurierbarkeit der zu verwendenden Modul-Implementierungen mithilfe des Spring Frameworks.
- Beschreibung und Begründung für die Wahl der Indexierungsalgorithmen.

Falls genügend Zeit vorhanden ist, soll die Integration in den bestehenden PDP Webservice Endpoint [DOH07SA] in einem technologischen Durchstich angedeutet werden. Eine Vollintegration ist nicht geplant.

Es ist beabsichtigt, die Arbeit bei gegebenem Interesse massgeblicher ausserschulischer Parteien als quelloffene Alternative zur existierenden Implementation SunXACML von Seth Proctor zu veröffentlichen. Eine allfällige Entscheidung wird nach Abschluss der Arbeit in Hinblick auf ein vorhandenes Interesse der Industrie getroffen.



## 2.2 Termine

- 01.10.07                    Beginn der Diplomarbeit
- 12.11.07                    • Abgabe Abstract an Betreuer
- 16.11.07                    • Abgabe des revidierten Abstracts durch den Betreuer an das  
Abteilungssekretariat (cfurrer@hsr.ch)
- 23.11.07                    • Abgabe des Berichtes an den Betreuer bis 12.00 Uhr  
• Fertigstellung des A0-Posters bis 12.00 Uhr  
• Präsentation der Diplomarbeit zwischen 15.00 und 19.00 in den Labors
- 26.11.07 –  
28.12.07                    • Mündliche Diplomprüfung

Ort, Datum:

Verantwortlicher Dozent, Wolfgang Giersche:



## 3 XACML

### *Inhalt*

XACML ist ein von der Standardisierungsorganisation OASIS erarbeiteter Standard um allgemein gültige Zugriffskontrollrichtlinien in XML zu beschreiben. Die Version 2.0 wurde im Februar 2005 von OASIS in Zusammenarbeit mit BEA Systems, Computer Associates, Entrust, IBM, Sun Microsystems und weiteren Organisationen definiert.

XACML beschreibt die Syntax von Richtlinien sowie das Format der Autorisierungsanfrage und -antwort. Die Sprache bietet Erweiterungspunkte, für zusätzliche Richtlinien, Kombinationsalgorithmen, Datentypen und Funktionen an.

XACML ist kein komplettes Autorisierungssystem, es beschreibt eine Basis, auf welcher ganzheitliche Lösungen aufbauen können.

Die Folgenden Abschnitte gehen auf den Hintergrund, Anforderungen an eine Richtlinienprache, den Aufbau, sowie die wichtigsten Vorteile von XACML ein.

### 3.1 Hintergrund

#### *Allgemein*

Der Rationalisierungseffekt hat die Hersteller von Computer Plattformen dazu getrieben, immer mehr Produkte mit möglichst generalisierten Funktionen zu entwickeln, damit diese in einem breitestem Spektrum von Situationen eingesetzt werden können. Die Produkte haben dadurch die maximalen Privilegien um auf Daten zuzugreifen und Applikationen auszuführen, damit diese in möglichst vielen Applikationsumgebungen eingesetzt werden können. Dies schliesst auch die Applikationsumgebungen mit den restriktivsten Sicherheitsprivilegien ein.

In einem grossen Unternehmen werden Sicherheitsrichtlinien an verschiedenen Stellen durchgesetzt. Die Richtlinien werden von verschiedenen Abteilungen verwaltet, wie z.B. im Personalwesen, in der Informatik, in der Rechtsabteilung oder der Finanzabteilung. Während diese im Extranet, Mail, WAN oder Remotezugriff durchgesetzt werden, welche eigene restriktivere Sicherheitsrichtlinien besitzen. Momentan ist es üblich, dass jede der erwähnten Plattformen die Konfiguration für die Sicherheitsrichtlinien selbst bewerkstelligt und so für einen restriktiven Zugriff sorgt.

Die Folge davon ist, dass das Durchsetzen von Firmen übergreifenden Richtlinien teuer und unzuverlässig ist, da diese an mehreren Orten konsistent angepasst werden müssen. Ebenfalls ist es dadurch nahezu unmöglich Sicherheitsmassnahmen über das gesamte Unternehmen aufrecht zu erhalten.

Gleichzeitig dazu steigt der Druck auf die Führung eines Unternehmens von Seiten der Kunden, Aktionären und der Aufsichtsbehörden, sowohl Informationen und Werte des Unternehmens als auch seine Kunden zu schützen.

Aus diesen Gründen steigt der dringende Bedarf einer allgemeinen Sprache, um Sicherheitsrichtlinien zu definieren. Bei einer unternehmensweiten Implementierung einer solchen Sprache ist es möglich, die Sicherheitsrichtlinien in allen Komponenten der IT-Systeme zentral zu verwalten.



### Anforderungen

Einige grundlegenden Anforderungen an eine Richtlinienprache für Informationssystem-Sicherheit sind:

- Zur Verfügung stellen eines Verfahrens um individuelle Regeln und Richtlinien in einem einzelnen Set, welches für eine bestimmte Entscheidungsanfrage gilt, zu kombinieren.
- Anbieten einer Methode zur Kombination von mehreren Richtlinien und Regeln.
- Möglichkeit mit mehreren Subjekten umgehen zu können.
- Fällen eines Autorisierungsentscheides anhand von Subjekt- und Ressource-Attributen.
- Ein Set von logischen und mathematischen Operationen, welche auf Subjekt-, Ressource-, und Umgebungs-Attribute anwendbar sind, zu Verfügung zu stellen.
- Handhabung von verteilten Sets von Richtlinien. Dabei wird die Art und Weise, wie diese aufgefunden und abgerufen werden abstrahiert.
- Eine Methode zur schnellen Identifizierung der Richtlinien, welche für eine bestimmte Aktion, basierend auf den Werten von Subjekt, Ressource und Aktion, zu Verfügung gestellt wird.
- Eine Abstraktionsschicht, welche das Erstellen der Richtlinien von den Applikationsumgebungen isoliert.



## 3.2 Aufbau

### *Allgemein*

In diesem Abschnitt wird auf den Aufbau von XACML 2.0 eingegangen. Dabei werden die wichtigsten Komponenten und deren Zusammenspiel beschrieben. Ebenfalls werden essentiellen Elemente zum definieren von Richtlinien in XACML 2.0 aufgezeigt.

### *Komponenten Modell*

Abbildung 1 zeigt den Datenfluss, wie er von der XACML 2.0 Spezifikation [XACML2.0] definiert wird. Folgende logische Elemente sind zu unterscheiden, wobei diese nicht als physikalisch eigenständige Komponenten implementiert werden müssen:

#### **Policy Administration Point (PAP)**

Der PAP ist die Administrationskomponente. Mit Hilfe des PAP können Richtlinien erstellt, verwaltet und dem PDP zur Verfügung gestellt werden.

#### **Policy Enforcement Point (PEP)**

Der PEP ist die Durchsetzende Einheit. Er unterbricht Zugriffe auf geschützte Ressourcen und leitet eine Autorisierungsanfrage an den Context Handler weiter. Abhängig von den Autorisierungsentscheidungen des PDP muss der PEP Zugriffe auf Ressourcen erlauben oder verweigern. Sind zusätzliche Obligationen in den Entscheidungen enthalten, so muss er sicherstellen, dass diese erfüllt werden.

#### **Policy Decision Point (PDP)**

Der PDP ist der Entscheider. Er erhält eine Entscheidungsanfrage und berechnet abhängig von den anwendbaren Richtlinien einen Entscheid, ob der Zugriff auf die geschützte Ressource erlaubt oder verweigert wird. Beinhalten die Richtlinien Obligationen, so muss der PDP diese mit dem Entscheid mitschicken.

#### **Policy Information Point (PIP)**

Der PIP ist die Informationskomponente. Er stellt zusätzliche Informationen über Subjekte, Ressourcen, Umfeld oder Aktionen zur Verfügung, welche von einer Richtlinie referenziert werden können.

#### **Context Handler (CH)**

Der Context Handler hat zwei Aufgaben. Zum einen ist er dafür verantwortlich, dass er Entscheidungsanfragen von PEP, in eine XACML Entscheidungsanfrage umformt. Zum anderen stellt er für den PDP zusätzliche Informationen zu einem Request bereit.

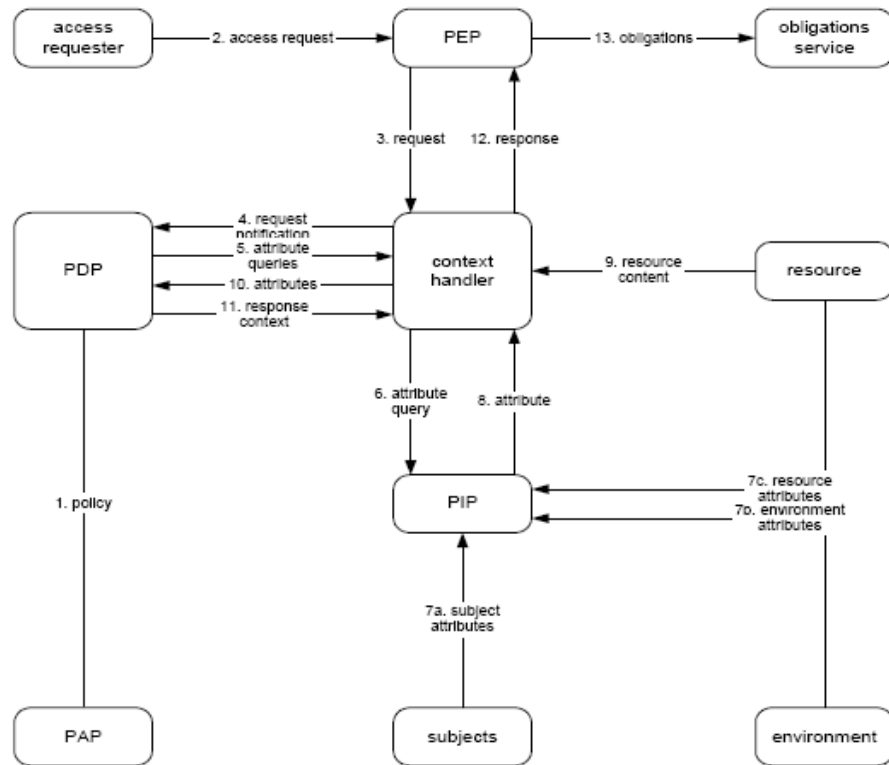
Datenfluss-  
diagramm

Abbildung 1 Datenflussdiagramm der XACML 2.0 Spezifikation

## Beschreibung

1. Der PAP definiert Richtlinien und stellt diese dem PDP zur Verfügung.
2. Der access requester (Benutzer oder System) schickt eine Zugriffsanfrage auf eine durch einen PEP geschützte Ressource.
3. Der PEP schickt eine Entscheidungsanfrage in XACML oder nativen Form an den Context Handler. Optional kann die Anfrage zusätzliche Attributinformationen über Subjekte, Ressourcen, das Umfeld oder die Aktion beinhalten.
4. Ist die Anfrage vom PEP nicht in XACML formuliert, wandelt er diese in eine XACML konforme Anfrage um. Zudem wird für jede angefragte Resource eine eigene Anfrage erstellt.
5. Wenn in der Anfrage Subjekt-, Resource-, Umfeld- oder Aktionsinformationen fehlen, fordert der PDP diese vom Context Handler an.
6. Der Context Handler erfragt die entsprechenden Attributinformationen bei einem PIP.
7. Der PIP holt die Informationen ein.
8. Der PIP sendet die Informationen an den Context Handler zurück.
9. Optional kann der Context Handler die Resource in die Anfrage einfügen.
10. Der Context Handler schickt die angeforderten Attribute und optional die Resource an den PDP zurück. Der PDP evaluiert die Anfrage mit den vorhandenen Richtlinien.



11. Der PDP schickt eine Antwort an den Context Handler zurück, welche den Autorisierungsentscheid beinhaltet. Sind in den zutreffenden Richtlinien Obligationen enthalten, so müssen diese zusätzlich in die Antwort übernommen werden.
12. Der Context Handler übersetzt die Antwort in eine Form, die für den PEP verständlich ist. Versteht der PEP XACML, so wird die Antwort unverändert weitergeleitet.
13. Enthält eine positive Antwort Obligationen, so muss der PEP diese ausführen bevor der Zugriff gestattet wird. Sind die Obligationen dem PEP unbekannt, wird der Zugriff wie auch bei einer negativen Antwort verweigert.[Graf06]

### Aufbau von XACML 2.0 Richtlinien

Für das Schützen von Ressourcen werden oft mehrere Richtlinien definiert. So kann der Zugriff auf Daten zum Beispiel durch zwei Richtlinien geschützt werden. Eine Richtlinie beschreibt allgemeine Datenschutzbestimmungen, eine weitere realisiert firmenbezogene Sicherheitsrichtlinien. Dieses Szenario kann mit XACML abgebildet werden, in dem Richtlinien in Sets von Richtlinien gruppiert werden. Weitere Komponenten aus dem XACML-Modell und deren Zusammenspiel zeigt das UML-Diagramm in der Abbildung 2.

### Modell der Richtliniensprache

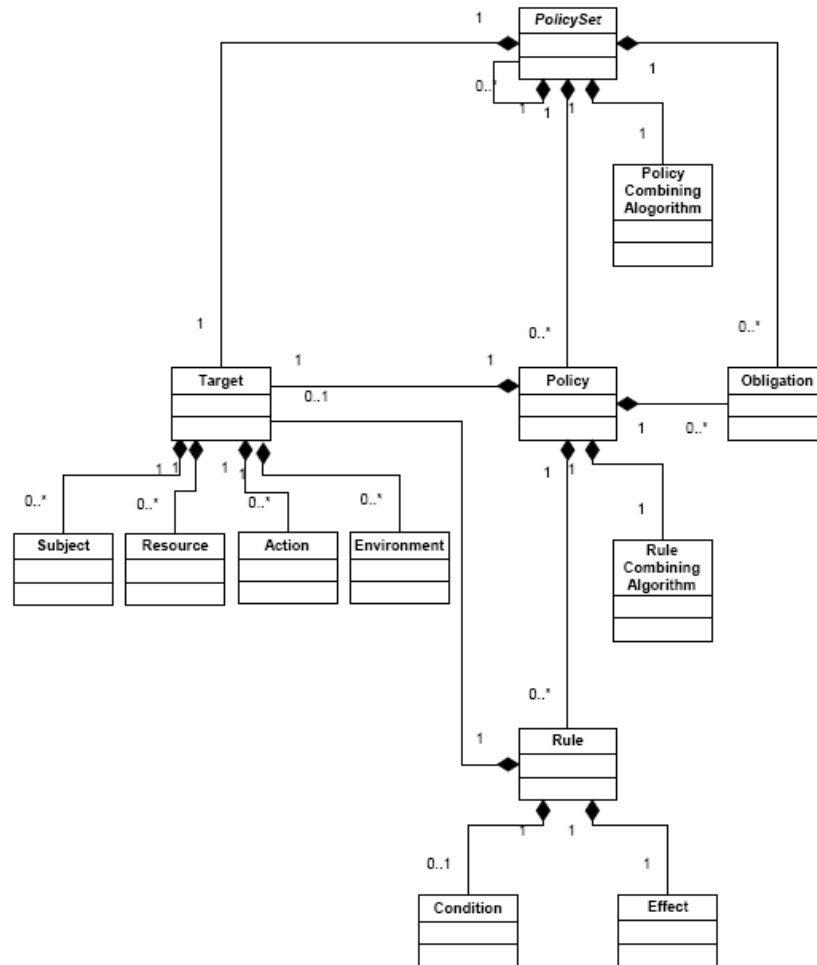


Abbildung 2 Policy Language Modell der XACML 2.0 Spezifikation



### Beschreibung

#### Rule

Eine Rule kann nicht selbstständig existieren, sondern muss in einer Policy enthalten sein. Die Rule ist dabei die grundlegendste Einheit einer Policy und setzt sich aus den folgenden Elementen zusammen:

- Target (optional)
- Condition (optional)
- Effect

Das Target-Element definiert, ob eine Rule für eine Autorisierungsanfrage anwendbar ist. Fehlt das Target-Element, so gelten die Target-Elemente der umschliessenden Policy. Das Condition-Element kann die Anwendbarkeit auf das Target-Element verfeinern. Auf diese Weise kann eine Rule zum Beispiel abhängig von der Tageszeit anwendbar sein.

Ist eine Rule anwendbar, so wird ihr Effect-Element wirksam. Das Effect-Element kann die Werte "permit" oder "deny" enthalten, welche sich entsprechend auf die Zugriffserlaubnis auswirken. [XACML2.0]

#### Policy

Eine Policy setzt sich aus den folgenden vier Elementen zusammen:

- Target
- eine beliebige Anzahl Rules
- rule-combining Algorithmus
- eine beliebige Anzahl Obligations

Das Target-Element definiert die Anwendbarkeit der Policy auf eine Autorisierungsanfrage. Damit die Effekte der enthaltenen Rules zu einem einzelnen Entscheid kombiniert werden können, wird ein entsprechender rule-combining Algorithmus verwendet. XACML definiert bereits einige Algorithmen, zum Beispiel deny-overrides oder first-applicable. Eigene Kombinalgorithmen können ebenfalls verwendet werden. Eine Policy kann Obligations einschliessen. Diese beinhalten Aktionen, welche vom PEP vor der Umsetzung des Autorisierungsentscheides des PDPs erfüllt werden müssen. [XACML2.0]



### PolicySet

Ein PolicySet ist eine Vereinigung einer Menge von Policies. Ähnlich wie eine Policy enthält ein PolicySet:

- Target
- eine beliebige Anzahl Policies oder PolicySets
- policy-combining Algorithmus
- eine beliebige Anzahl Obligations

Das Target-Element sowie die Obligations werden in der gleichen Weise verwendet, wie bei einer Policy. Ein PolicySet kann eine Menge von Policies oder PolicySets enthalten. Das PolicySet verwendet einen policy-combining Algorithmus zum Kombinieren von Entscheidungen der enthaltenen Policies. XACML bietet zum Kombinieren von Policies dieselben Algorithmen an, wie zum Vereinen von Rules, wie z. B. deny-overrides oder first-applicable.[XACML2.0]

### Target

Policies, PolicySets oder Rules enthalten ein Target-Element, welches aus einer Menge der folgenden Elemente bestehen kann:

- Subject-Elemente
- Resource-Elemente
- Action-Elemente
- Environment-Elemente

Das Target-Element ist das wichtigste Element, um die Anwendbarkeit der Rules, Policies und PolicySets zu bestimmen.[XACML2.0]

## 3.3 Vor- und Nachteile von XACML

### *Wichtigste Vorteile*

#### **XACML ist ein Standard**

Zumal XACML eine ratifizierte Sprache ist, können Entwickler aus den Erfahrungen einer grossen Gemeinschaft profitieren. Daher müssen sie keine eigenen Systeme entwickeln und sich Gedanken über die verwendete Autorisierungssprache machen. Darüber hinaus vereinfacht sich die Integration verschiedener Systeme die XACML implementieren mit zunehmender Verbreitung.

#### **XACML ist generisch**

In Folge dessen kann XACML für jedes Umfeld und jede Ressource verwendet werden. Richtlinien können mehrfach verwendet werden. Dies erleichtert die Administration von Richtlinien.



### **XACML ist verteilt**

Richtlinien können verteilt und an verschiedenen Orten gespeichert werden. Riesige und monolithische Richtlinien können aufgeteilt und durch verschiedene Personen oder Gruppen verwaltet werden. Auf diese Weise werden Richtlinien von Experten erstellt und können so stets aktuell gehalten werden.

### **XACML ist leistungsfähig**

Es gibt viele Möglichkeiten, die Sprache zu erweitern, auch wenn das für die meisten Fälle nicht notwendig ist. Die Standardversion von XACML unterstützt bereits viele Datentypen, Funktionen und Regeln für das Kombinieren von Resultaten der Richtlinien. Zusätzlich arbeiten Standardisierungsgruppen an Anbindungsprofilen und Erweiterungen, damit XACML mit anderen Standards (zum Beispiel SAML, XML Digital Signature) verwendet werden kann. [Sun05]

#### *Nachteile*

Ein Nachteil bei der Verwendung einer derart flexiblen Sprache ist der Mehraufwand, den man betreiben muss, um die Metadaten zu verarbeiten, welche in dieser Sprache stecken.

Dieser Nachteil lässt sich anhand eines typischen Anwendungsfalls von XACML aufzeigen. Verarbeitet man die Kommunikation zwischen PEP und PDP mit SOAP, SAML und XACML, wird der Anteil der Nutzdaten im Verhältnis zu den Metainformationen sehr klein. [Egg06]

## **3.4 Zukunft von XACML**

#### *Zukunft von XACML*

Das Ansehen von XACML wächst stetig. In einer kürzlich verfassten Medienmitteilung von OASIS [PressInterop] wurde bekannt gegeben, dass acht namhafte Firmen, darunter BEA Systems, IBM, Oracle, Red Hat usw., zusammen arbeiten um die Interoperabilität von XACML zu demonstrieren.

Dies und die Tatsache, dass Zugriffskontrolle in der heutigen Zeit eine Anforderung von fast jeder Applikation ist, könnten den Ausschlag dafür geben, dass sich XACML in Zukunft durchsetzt.



## 4 HERAS<sup>AF</sup>

### *Inhalt*

In diesem Abschnitt wird auf den Hintergrund, sowie den Aufbau von HERAS<sup>AF</sup> eingegangen. Dabei werden die Ziele von HERAS<sup>AF</sup> auf den Punkt gebracht, sowie der Aufbau der einzelnen Module und ihr Zusammenwirken genauer erläutert. Zum Schluss dieses Kapitels wird eine Einführung in die, innerhalb dieser Diplomarbeit entwickelte, HERAS<sup>AF</sup> XACML 2.0 Implementierung gegeben. Die HERAS<sup>AF</sup> XACML 2.0 Implementierung bildet den Kern von HERAS<sup>AF</sup>.

### 4.1 Hintergrund

#### *Allgemein*

HERAS<sup>AF</sup> ist ein Open Source Projekt in der Entstehungsphase. Initiiert wurde das Projekt im Januar 2006, nach ca. achtmonatiger Ideenentwicklung, Konzeption und Projektion. Die Idee zu HERAS<sup>AF</sup> ist eine Gemeinschaftsidee von Wolfgang Giersche, René Eggenschwiler und Yan Graf.

Die Module PEP und PAP wurden von Sandro Strebel und Massimo Cerqui entwickelt.

Der PDP Web Service Endpoint sowie die Implementierung von XACML 2.0 wurde von Stefan Oberholzer, Florian Huonder und Sacha Dolski entwickelt.

#### *Motivation*

In der heutigen Zeit entstehen immer mehr so genannte virtuelle Unternehmen, welche durch Vernetzungen mit anderen Firmen ihre Dienstleistungen anbieten. Dies zwingt Firmen, den Zugriff auf kritische Ressourcen, Informationen und Applikationen zu öffnen.

Produkte und Dienstleistungen können so effizienter geliefert werden, was zu tieferen Kosten und höherer Produktivität führt.

Oft sind Benutzerinformationen über mehrere Systeme in einem Unternehmen verteilt. Jede Applikation hat seine eigene Benutzerverwaltung und Zugriffskontrolle. Dadurch wird das Durchsetzen von allgemeinen und firmenbezogenen Sicherheitsrichtlinien erschwert, was eine Autorisierung über Unternehmensgrenzen fast unmöglich macht.

Darüber hinaus wird von immer mehr Benutzern mit unterschiedlichen Rechten auf Applikationen zugegriffen. Die Verantwortung und Funktion von einzelnen Benutzern sowie die Firmenstruktur sind von ständigen Änderungen betroffen, was zu widersprüchlichen Herausforderungen führt. Unternehmen müssen sich öffnen und gleichzeitig die Sicherheit erhöhen.

HERAS<sup>AF</sup> nimmt diese Herausforderung an und setzt sich zum Ziel, verteilte Autorisierung unternehmenstauglich zu machen. Dies wird durch den Einsatz von einer offenen und technologieunabhängigen Sprache wie XACML ermöglicht. Dabei setzt HERAS<sup>AF</sup> ausschliesslich auf frei verfügbare und quelloffene Komponenten. [Graf06]



## Ziele

Um HERAS<sup>AF</sup> im Open Source Markt positionieren zu können ist es wichtig, dass einige grundlegende Ziele erfüllt werden. Einige der wichtigsten Ziele sind nachfolgend definiert.

### Ganzheitlicher Ansatz

- HERAS<sup>AF</sup> unterstützt den gesamten Autorisierungsprozess.
- Im technischen Sinne bedeutet dies, dass jegliche Zugriffe auf geschützte Ressourcen von verteilten Agenten (PEPs) unterbrochen und an einen PDP umgeleitet werden, wo sie, basierend auf anwendbare Richtlinien evaluiert werden. Nur wenn eine positive Antwort vom PDP zurückkommt, wird der PEP Zugriff auf die geschützte Ressource gewähren.
- Mit „ganzheitlich“ ist aber auch gemeint, dass alle Aspekte des Entwerfens und der Wartung von Policies durch nichttechnisches Personal im Frameworkdesign beachtet werden, speziell im PAP.

### Unternehmenstauglichkeit

- HERAS<sup>AF</sup> in einer Unternehmung zu integrieren, soll nur geringe Änderungen in der existierenden Infrastruktur zur Folge haben. Existierende Authentifizierungslösungen können weiterhin verwendet oder integriert werden.
- HERAS<sup>AF</sup> wird mit explizitem Hinblick auf Erweiterbarkeit und Anpassbarkeit entworfen. Durch die konsistente Verwendung des Spring IoC-Containers ist die Austauschbarkeit von Komponenten, jederzeit garantiert. Eigene Komponenten können durch Integration der beschriebenen API realisiert werden.
- Die API von HERAS<sup>AF</sup> kann verwendet werden um eine unternehmensspezifische Einbindungen zu realisieren. Die spezifischen Komponenten brauchen lediglich die Erweiterungspunkte von HERAS<sup>AF</sup> zu verwenden. Beispielsweise können Verbindungen zu ERP-Systemen implementiert werden, welche zusätzliche Informationen für die Zugriffsentcheidung liefern. Dies können Betriebs-, Logistik-, Personal- oder sonstige Geschäftsdaten in der Autorisierung sein.
- HERAS<sup>AF</sup> verwendet existierende, etablierte Standards. Damit steht das Framework auf einer breiten, offenen Basis für zukünftige Erweiterungen. Dies steigert die Interoperabilität und vereinfacht die Einbindung von HERAS<sup>AF</sup> in bestehende und zukünftige Infrastrukturen.
- Der PAP bietet geschäftstypische Ansichten und Sprachmittel, so dass ein Richtlinienadministrator nicht zu verstehen braucht, was eine URL oder eine Klassenmethode ist. Vorlagen (Templates) stellen die Funktionalitäten zur Verfügung, die es einem Administrator ermöglichen, technische Richtlinien mit Geschäftssyntax anzureichern (technische Sicht). Ein Berechtigungsverantwortlicher kann mit diesen Vorlagen neue Richtlinien definieren. Dabei kann er die Richtlinie in geschäftsüblicher Sprache ausfüllen (geschäftliche Sicht), wodurch er seine Arbeit effektiver erledigen kann.



### **Application Security**

- Nicht mehr jede Applikation muss den Zugriff auf ihre Ressourcen schützen. Diese Verantwortung kann an HERAS<sup>AF</sup> delegiert werden. Das Framework bietet Agenten (PEPs), die als Unterbrecher in bestehende und neue Anwendungen eingebettet oder vorgeschaltet werden können.

### **Architecture Framework**

- HERAS<sup>AF</sup> ist ein Architekturframework und verbindet bestehende Open Source Komponenten zu einer ganzheitlichen Autorisierungslösung. Alle Komponenten sind zukunftsweisende Technologien wie Java 5.0, Tomcat 5.5 usw.

Sobald der Code den Alpha-Status erreicht, wird HERAS<sup>AF</sup> unter der LGP-Lizenz veröffentlicht. [Graf06]



## 4.2 Aufbau

### Übersicht

HERAS<sup>AF</sup> realisiert die, durch die XACML 2.0 Spezifikation definierten Komponenten, welche den Autorisierungsprozess unterstützen. Der in XACML 2.0 definierte ContextHandler wird in HERAS<sup>AF</sup> nicht explizit als Modul implementiert sondern in die Module PEP und PDP verteilt.

HERAS<sup>AF</sup> realisiert alle Instanzen, welche für einen durchgehenden Autorisierungsprozess benötigt werden. Jede Instanz ist einzeln verteil- und installierbar. Zudem können sie unabhängig voneinander betrieben und gewartet werden.

HERAS<sup>AF</sup> unterstützt die Spezifikation von XACML 2.0 und vereinfacht die Anwendung von Zugriffskontrolle in Unternehmensbereichen.

Abbildung 3 zeigt das Komponentenschema von HERAS<sup>AF</sup>.

### HERAS<sup>AF</sup> Komponenten- schema

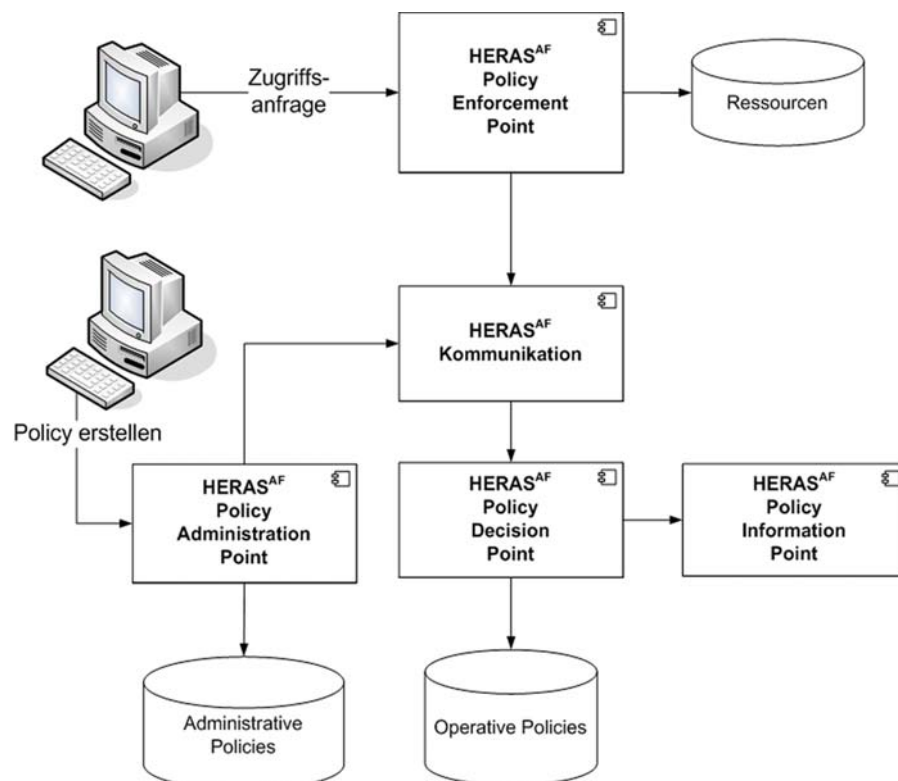


Abbildung 3 HERAS<sup>AF</sup> Komponentenschema

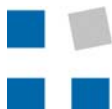
### HERAS<sup>AF</sup> Komponenten- beschreibung

#### HERAS<sup>AF</sup> Policy Enforcement Point

Der HERAS<sup>AF</sup> PEP stellt die Strukturen und Mechanismen zur Verfügung um Agents zu realisieren, welche die Zugriffskontrolle innerhalb einer schützenswerten Applikation durchführen. Dazu wird im Allgemeinen das Pattern der unterbrechenden Filter oder der unterbrechenden Web-Agenten angewandt. [Egg06]

Applikationsspezifische PEPs können daher auf diverse Arten realisiert werden, weil verschiedene Unterbrechungsmöglichkeiten zur Verfügung stehen.

Zur Logik eines HERAS<sup>AF</sup> PEP gehören vor allem die Erstellung von



Autorisierungsanfragen, die Kommunikation mit dem PDP, sowie die kontextspezifische Interpretation einer Autorisierungsantwort. In der Arbeit HERAS<sup>AF</sup>: Interzeptoren für Spring AOP und AspectJ [CerStr07PEP] wird näher auf einen HERAS<sup>AF</sup> PEP eingegangen.

### HERAS<sup>AF</sup> Policy Decision Point

Der HERAS<sup>AF</sup> PDP realisiert die komplette Logik der Zugriffsentscheidungsfindung. Dazu gehören das performante Auffinden von anwendbaren Richtlinien, die Evaluierung einer Anfrage gegen die gefundenen Richtlinien, wie auch die Kombination der erhaltenen Resultate.

Da das Hauptaugenmerk beim PDP vor allem in der Performanz liegt, wurden einige leistungsfördernde Mechanismen, wie Indexierung oder schnelle Vergleichsalgorithmen eingebaut. Ebenfalls wurde der HERAS<sup>AF</sup> PDP so entwickelt, dass nur während der Initialisierung auf die Datenhaltung zugegriffen werden muss. Danach werden alle Richtlinien im RAM gehalten, was die Entscheidungsfindung des HERAS<sup>AF</sup> PDP entscheidend beschleunigt.

Um das Auffinden der Anwendbaren Richtlinien möglichst performant zu lösen wurden Indexierungsalgorithmen entwickelt, welche nur die Richtlinien zurückgeben, welche für die Entscheidungsfindung in betracht gezogen werden. Dies beschleunigt die Entscheidungsfindung. Um die Benutzbarkeit dieser Indexe möglichst offen zu halten, sind diese konfigurierbar.

Der HERAS<sup>AF</sup> PDP wird üblicherweise als zentrale Einheit eingesetzt, die mehrere PEP's bedient. Physikalisch lässt sich der HERAS<sup>AF</sup> PDP auf verschiedene virtuelle Maschinen und Rechner verteilen. So können zum Beispiel Lastverteilungen oder Ausfallregelungen angewendet werden.

Die Anforderungen an einen PDP sind in der Arbeit [Graf06] definiert worden, sowie auch durch die XACML Spezifikation [XACML2.0] vorgegeben.

### HERAS<sup>AF</sup> PEP <-> PDP Kommunikation

Die Kommunikation zwischen einem HERAS<sup>AF</sup> PEP und einem HERAS<sup>AF</sup> PDP wird über einen Web Service erstellt. Um sicherzustellen dass die HERAS<sup>AF</sup> Komponenten auch mit Komponenten von anderen Anbietern zusammenarbeiten können, ist eine Standardkonforme Übertragung essentiell. Um die Integrität der Übertragung zu sichern wird die XACML Nachricht in SAML eingepackt, welches zum Versenden über den Web Service in SOAP gekapselt wird. Die Datenübermittlung erfolgt anschliessend mit dem http Protokoll.

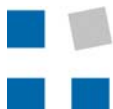
Alle verwendeten Standards sind offen und in den Anwendungsbereichen der Sicherheit und der Datenübermittlung etabliert.

Die Umsetzung der PEP <-> PDP Kommunikation wurde in der Arbeit HERAS<sup>AF</sup>: XACML PDP Web Service Endpoint [DOH07SA] beschrieben.

### HERAS<sup>AF</sup> PAP <-> PDP Kommunikation

Die Kommunikation zwischen einem HERAS<sup>AF</sup> PAP und einem HERAS<sup>AF</sup> PDP wird wie bei der HERAS<sup>AF</sup> PEP <-> PDP Kommunikation über einen Web Service erstellt und durch die gleichen Standards, wie in HERAS<sup>AF</sup> PEP <-> PDP Kommunikation beschrieben, gesichert.

### HERAS<sup>AF</sup> PDP <-> PIP Kommunikation



Der HERAS<sup>AF</sup> PIP ist so abstrahiert, dass dieser direkt in den PDP eingebaut, oder über einen Web Service angesprochen werden kann. Wie dies realisiert wird, ist im Moment noch offen.

### **HERAS<sup>AF</sup> Policy Information Point**

Falls eine Anfrage nicht alle benötigten Attribute enthält, liefert der HERAS<sup>AF</sup> PIP dem HERAS<sup>AF</sup> PDP die fehlenden Attribute.

Ein PIP stellt die Logik und Struktur zur Verfügung, um spezifische Adapter und Bindeglieder zwischen der Autorisierungslösung und der bestehenden Infrastruktur zu realisieren. Somit können Entwickler und Betreuer Daten aus Legacy-Systemen in die Zugriffskontrolle einbinden, indem sie nur den benötigten Adapter programmieren.

Wie ein PIP in HERAS<sup>AF</sup> eingebunden werden soll, wurde noch nicht festgelegt.

### **Policy Store**

Die definierten Richtlinien und Sets von Richtlinien werden in verschiedenen Datenbanken gehalten.

Auf dem HERAS<sup>AF</sup> PAP müssen die Richtlinien in einer lokalen Datenbank gespeichert werden. Ebenfalls muss geloggt werden, welche Richtlinien zurzeit auf dem PDP deployt sind.

Auf dem HERAS<sup>AF</sup> PDP werden die persistierten Richtlinien ebenfalls gespeichert und beim initialisieren ins RAM geladen. Wurde der HERAS<sup>AF</sup> PDP initialisiert, wird für die Evaluierung nicht mehr auf die Datenbank zugegriffen.

### **HERAS<sup>AF</sup> Policy Administration Point**

Der HERAS<sup>AF</sup> PAP erstellt und verwaltet die Richtlinien. Zudem ist er dafür zuständig, dass die PDPs die benötigten Richtlinien kennen.

Um das Erstellen von den teilweise komplexen Richtlinien zu erleichtern, wurde mit JSF eine benutzerfreundliche Oberfläche entwickelt, welche diese anhand von vordefinierten Templates erstellt. Genauerer zu diesem Modul kann in der Arbeit HERAS<sup>AF</sup> PAP [CerStr07PAP] nachgelesen werden.

#### *Besonderheiten*

HERAS<sup>AF</sup> besitzt folgende Besonderheiten die von der XACML 2.0 Spezifikation [XACML 2.0] abweichen.

### **HERAS<sup>AF</sup> hat kein ContextHandler Modul**

In HERAS<sup>AF</sup> wurde die Funktionalität des ContextHandlers auf die Module HERAS<sup>AF</sup> PDP und HERAS<sup>AF</sup> PEP verteilt. Ein HERAS<sup>AF</sup> PEP generiert eine XACML konforme Anfrage an den PDP und der HERAS<sup>AF</sup> PDP holt die benötigten Informationen direkt beim HERAS<sup>AF</sup> PIP.



## 4.3 HERAS<sup>AF</sup> XACML 2.0 Implementierung

### Hintergrund

Als die Idee entstand HERAS<sup>AF</sup> zu entwickeln, wurde die SunXACML Referenzimplementierung als XACML Implementierung verwendet.

Während der Entwicklung von HERAS<sup>AF</sup> zeigte sich, dass diese in gewissen Punkten nicht dem XACML 2.0 Standard entspricht und einzelne Features fehlerhaft oder noch nicht implementiert sind, was die Weiterentwicklung von HERAS<sup>AF</sup> erschwerte. [DOH07SA]. Dies bedeutet, dass SunXACML für eine mögliche Nutzung in HERAS<sup>AF</sup> erweitert werden müsste.

So bestand z.B. kein Indexierungsmechanismus und die Persistierung von Richtlinien wurde nur für Testzwecke implementiert.

Wenn HERAS<sup>AF</sup> auf Basis von SunXACML weiterentwickelt worden wäre, müssten diese Mängel und Schwachstellen der SunXACML Implementierung behoben werden.

### Motivation

Anhand der teilweise negativen Erfahrungen mit Sun XACML (beschrieben in [DOH07SA]) und der Tatsache, dass seit Juni 2006 an der Sun XACML Implementierung keine Änderungen mehr vorgenommen wurde, wurde entschieden, den Aufwand einzugehen, eine eigenen XACML 2.0 Implementierung zu entwickeln.

Eine zusätzliche Motivation bildet die Tatsache, dass namhafte Firmen, wie BEA Systems, IBM, Oracle usw. sich dazu entschieden haben, die Interopabilität von XACML aufzuzeigen [PressInterop] und dazu eine Zusammenarbeit eingehen.

Mit einer Implementierung von XACML 2.0 könnte das Interesse einer dieser Firmen für das Projekt HERAS<sup>AF</sup> geweckt werden.

### Ziele

Für die XACML 2.0 Implementierung wurden folgende Ziele definiert:

#### „Must Elemente“ abdecken

Die XACML 2.0 Spezifikation [XACML 2.0] spezifiziert viele Funktionen und Elemente, bei denen jeweils angegeben wird ob diese pflichtmässig oder optional zu implementieren sind. Damit die HERAS<sup>AF</sup> XACML 2.0 Implementierung standardkonform ist, sind alle pflichtmässigen Elemente und Funktionen zu implementieren.

#### Flaschenhals bei Datenbankzugriff vermeiden

Um wiederkehrende Zugriffszeiten auf die Datenhaltung zu vermeiden, sollen die Richtlinien beim Initialisieren des PDPs ins Memory geladen werden.

#### Schnelles Auffinden der anwendbaren Richtlinien

Um die Richtlinien möglichst performant aufzufinden, sind Indexierungsalgorithmen notwendig. Damit diese auf die individuellen Bedürfnisse der Firmen optimiert werden können, sollen diese konfigurierbar sein.



### **Abstrahieren der einzelnen Module**

Die Austauschbarkeit der einzelnen Module wie ReferenceLoader, Locator oder PDP soll durch eine Abstraktionsschicht gewährleistet werden.

### **Integration in Applikationen mit Spring**

Die Integration der HERAS<sup>AF</sup> XACML 2.0 Implementierung in eine Applikation, soll über Spring ermöglicht werden. Zudem soll die Konfiguration der Implementierung ebenfalls mit Spring umgesetzt werden.

### **Generieren und Anpassen der Policy und Context Klassen**

Die Policy und Context Klassen sollen mit JAXB 2.1 generiert und anschliessend angepasst werden. Dies soll den Vorteil bieten, dass sich für Marshalling und Unmarshalling JAXB 2.1 verwenden lässt.

### **ReferenceLoader und AttributeFinder als Mock**

Die Module ReferenceLoader und das Interface AttributeFinder sollen in den Ablauf der Entscheidungsfindung integriert, können aber gemockt werden. Dies soll über vordefinierte Interfaces geschehen.

### **Eigene Funktionen, Datentypen und Kombinationsalgorithmen**

Die Architektur der HERAS<sup>AF</sup> XACML 2.0 Implementierung soll erlauben, dass eigene Funktionen, Datentypen und Kombinationsalgorithmen geschrieben und eingebunden werden können.

### **Validierung ein-/ ausschaltbar**

Die Validierung von eingehenden Anfragen und zu speichernde Richtlinien soll ein- und ausschaltbar sein. Dies soll über ein Konfigurationsfile geschehen.

### **ContextHandler ist Teil von PDP und PEP**

Der in der XACML 2.0 Spezifikation [XACML 2.0] definierte ContextHandler soll nicht als eigenes Modul implementiert werden, sondern soll ein Teil vom HERAS<sup>AF</sup> PDP und vom HERAS<sup>AF</sup> PEP sein. So soll der HERAS<sup>AF</sup> PEP die HERAS<sup>AF</sup> XACML 2.0 Implementierung verwenden um eine standardkonforme Anfrage, direkt an den PDP absetzen. Der PDP wiederum soll eine Antwort direkt an den PEP senden, welche wiederum XACML 2.0 konform ist.



# Teil II: Technischer Bericht HERAS<sup>AF</sup> XACML 2.0 Implementierung



# 1 Analyse

## Inhalt

In diesem Kapitel werden einzelne funktionale und nichtfunktionale Anforderungen an die HERAS<sup>AF</sup> XACML 2.0 Implementierung gestellt und Schwerpunkte der Arbeit festgehalten. Dabei werden verschiedene Varianten aufgezeigt wie allfällige Realisierungen aussehen könnten, sowie deren Vor- und Nachteile diskutiert. Welche der Varianten dann effektiv zum Einsatz gelangen wird in der Herausforderung der einzelnen Module im Kapitel 3 des Technischen Berichtes festgehalten.

## 1.1 Anforderungen

### Funktionale Anforderungen

#### Anforderungen an den PDP

---

Remote Referenzen	Aus Integritätsgründen sollen Referenzen auf Remote Policies bei jedem eingehenden Request neu aufgelöst werden. Wenn viele Remote Referenzen aufgelöst werden müssen, drückt dies auf die Performanz, weshalb ein Mechanismus zum Cachen der Remote Policies denkbar ist.
ReferenceLoader Modul	Das ReferenceLoader Modul, zum Auflösen von Policyreferenzen, soll in den Ablauf der Entscheidungsfindung integriert werden, kann aber momentan noch gemockt werden.
PDP als Hauptmodul	Der PDP soll das Hauptmodul für die Entscheidungsfindung und hat Referenzen auf alle anderen Module der HERAS <sup>AF</sup> XACML 2.0 Implementierung.
Initialisieren	Beim Initialisieren des PDP sollen die in einem Policy Store gehaltenen Policies direkt ins RAM geladen und dort gehalten werden.
Evaluieren	Das Evaluieren eines Requests soll anhand der anwendbaren Policies im RAM und den Combining Algorithmen geschehen.

#### Anforderungen an den Locator

---

Indexierung	Der Locator soll die auf den PDP deployten Policies indexieren um die Performanz beim Auffinden der anwendbaren Policies zu steigern.
Indexierung konfigurieren	Der Locator soll die Möglichkeit bieten, die Indexierung anhand eines Konfigurationsfiles an die eigenen Bedürfnisse anzupassen.



### Anforderungen an die Persistenz

---

Policy Speicherung	Die Policies werden als BLOB in die Datenbank gespeichert. Dies soll die Geschwindigkeit beim Initialisieren des PDP, welcher die Policies aus der Datenbank lädt, erhöhen.
Unabhängigkeit	Die Persistierung soll, unabhängig von der darunter liegenden Datenhaltung, geschehen.

### Anforderungen an den ReferenceLoader

---

Referenzen einmal pro Anfrage	Um die Integrität der Policies zu gewährleisten, sollen Policyreferenzen bei jeder Anfrage neu aufgelöst werden.
Referenzen als Set	Um Performanzverluste beim Auflösen von Policyreferenzen zu vermeiden, soll dem ReferenceLoader falls notwendig ein Set, das alle benötigten Policyreferenzen beinhaltet, übergeben werden. Der ReferenceLoader löst dann die lokalen- und die remote Referenzen auf. Die Verbindung zu einem remote PDP muss dadurch nur an einem Ort, einmal gemacht werden.

### Anforderungen an den Core

---

Erweiterbarkeit	Der Core soll so gestaltet werden, dass Funktionen, Datentypen und Combining Algorithmen erweitert und im Konfigurationsfile eingebunden werden können.
-----------------	---

### Nichtfunktionale Anforderungen

#### Zuverlässigkeit

---

Fehlertoleranz	Ist der PDP einmal hochgefahren, darf dieser keine RuntimeException werfen. Treten Fehler auf, müssen diese abgefangen werden und entsprechende Exceptions geworfen werden, welche bei einer Anfrage in einem Indeterminate mit entsprechendem ErrorCode resultieren.
----------------	---

#### Benutzbarkeit

---

Verständlichkeit	Die API der HERAS <sup>AF</sup> XACML 2.0 Implementierung soll komplett in Englisch als Javadoc dokumentiert werden. Die Dokumentation soll für Entwickler verständlich geschrieben werden.
Bedienbarkeit	Die Bedienbarkeit soll einfach gehalten werden. Einstellungen sollen in Konfigurationsfiles angepasst werden können. Die Konfigurationsfiles sollen dokumentiert werden.



### Leistung und Effizienz

---

Antwortzeiten	Die Antwortzeiten für die Entscheidungsfindung eines PDP sollen sehr gering sein. Sie kann aber variieren, je nach Anzahl von Policies im Speicher und wenn weitere Ressourcen aufgelöst werden müssen.
Ressourcebedarf	Der PDP soll auf einem sehr leistungsstarken Server installiert werden. Dies ist notwendig, da die ganzen Policies im RAM gehalten werden.

### Wartbarkeit, Änderbarkeit

---

Analysierbarkeit	Die HERAS <sup>AF</sup> XACML 2.0 Implementierung soll so dokumentiert sein, dass sie analysierbar ist um allfällige Änderungen und Erweiterungen durchzuführen.
Prüfbarkeit	Es sollen Tests durchgeführt werden, welche die Richtigkeit von Änderungen oder Wartungen belegen.

### Sicherheit

---

Verfügbarkeit	Der PDP muss eine hohe Verfügbarkeit gewährleisten. Im laufenden Betrieb darf dieser nicht durch RuntimeException oder OutOfMemory Exception ausfallen.
---------------	---



## 1.2 Schwerpunkte und Herausforderungen

**Allgemein** In diesem Abschnitt wird auf verschiedene Schwerpunkte und Herausforderungen der HERAS<sup>AF</sup> XACML 2.0 Implementierung eingegangen, sowie Vor- und Nachteile einzelner Ansätze diskutiert.

### 1.2.1 Context und Policy Klassen

**Allgemein** Es gibt verschiedene Möglichkeiten um die Policy und Context Klassen der XACML 2.0 Spezifikation [XACML 2.0] zu implementieren. In diesem Abschnitt wird dabei näher auf die Möglichkeiten der einzelnen Varianten eingegangen.

**Varianten** **Variante 1 selber implementieren**

Für die in der XACML 2.0 Spezifikation definierten Context und Policy Elemente werden vom Entwickler selbst Klassen geschrieben, welche deren Eigenschaften unterstützen. Ebenfalls wird ein Unmarshaller, welcher XML-Dokumente parst und in Java-Objekte umwandelt, sowie ein Marshaller, welcher aus Java-Objekte ein XML-File generiert, realisiert. Diese sind notwendig für das Auswerten eines Request, sowie für das Generieren einer dazugehörigen Response. Auch benötigt werden diese beim Erstellen eines Request durch den PEP.

#### Vorteile

- Selbst geschriebene Klassen sind übersichtlicher und verständlicher.
- Je nach Wahl und Umsetzung der Parsing Technologie sehr schnell.

#### Nachteile

- Unmarshaller und Marshaller müssen selbst entwickelt werden, was einen erheblichen zeitlichen Mehraufwand mit sich bringt.

#### Variante 2 von JAXB generieren lassen

Die XACML 2.0 Spezifikation enthält XML Schema Files, welche Eigenschaften und Einschränkungen der einzelnen Context und Policy Elemente definieren. Aus diesen Schema Files kann man nun mit JAXB die zu den Elementen dazugehörigen Klassen generieren. Ebenfalls bietet JAXB anhand dieser Schema Files Marshaller und Unmarshaller an.

#### Vorteile

- Kein Aufwand zum Erstellen der Context und Policy Klassen.
- Marshaller und Unmarshaller werden von JAXB angeboten, dadurch entfällt das Entwickeln eigener Parser.

#### Nachteile

- JAXB generiert teilweise unschönen Code.
- Generierte Klassen müssen angepasst werden um eigene Interfaces zu implementieren.



## 1.2.2 Indexierung

### Allgemein

Je nach Firmengrösse kann es möglich sein, dass tausende Policies bestehen um einzelne Ressourcen zu schützen. Werden diese Policies nun auf einen PDP deployt, muss dieser, ohne eine Indexierung jede einzelne Policy durchgehen und nach einer oder mehreren, auf den Request passende Policies durchsuchen. Dies kann bei einer hohen Anzahl von Policies so lange dauern, dass dies unbrauchbar wird. Ziel muss es sein, dass ein Request vom PDP möglichst schnell ausgewertet wird, damit keine Wartezeiten bei Programmaufrufen entstehen oder diese minimal gehalten werden können. Es wurden mehrere Varianten analysiert um eine geeignete Indexierung zu finden.

### Varianten

#### Variante 1 Hash Map

Für das Target jeder Policy und jedes PolicySets wird aus gültigen Kombinationen der akzeptierten Datentypen von Subject, Resource und Environment ein Hashwert gebildet. Weil die Action nur den Datentyp String beinhalten kann, kann diese vernachlässigt werden. Diese dienen als Schlüssel in der Hash-Map um die Policies aufzufinden.

Für jeden Hashwert wird ein Set erstellt, bestehend aus Referenzen auf Policies und PolicySets welche über den Hashwert aufgefunden werden sollen.

Um die entsprechende Policy für einen Request zu finden, wird ein Hashwert über die Datentypen von Subject, Resource und Environment berechnet und danach in der Hash-Map gesucht. Im erhaltenen Set werden die Policies iterativ durchsucht. Wird eine passende Policy gefunden, wird abgebrochen. Wird keine gefunden, wird der Hash mit fehlenden Subject-, Resource- oder Environment-Elementen gebildet um Policies zu finden, welche für alle Elemente gelten. [DOH07SA]

#### Vorteile

- Im optimalen Fall sehr schneller Zugriff,  $O(c)$ , auf eine Liste von evtl. anwendbaren Policies bzw. PolicySets.

#### Nachteile

- Da ca. 90 Prozent der Anfragen auf den Datentyp String gehen werden, ist es marginal schneller als wenn man die ganze Liste durchgeht.
- Die Liste von anwendbaren Policies, welche über den Datentyp String gefunden wird, ist sehr gross.

#### Variante 2 Baum

Die Policies werden in einer Baumstruktur abgelegt. Gesucht wird im Baum jeweils nach den Datentypen der Elemente im Request. Subject, Resource und Environment bilden eine eigene Ebene im Baum. Die Reihenfolge der Elemente in den Ebenen wird so gewählt, dass die Anzahl der Äste für einen Knoten möglichst klein sind. Die Blätter des Baumes enthalten jeweils ein Set mit Referenzen auf Policies, welche zu den Datentypen passen. Dieses Set wird anschliessend iterativ nach dem passenden Target durchsucht. [DOH07SA]



### Vorteile

- Sehr schneller Zugriff, auf eine Liste von evtl. anwendbaren Policies bzw. Policy Sets.
- Priorisierung der Knotenanordnung möglich, so dass das Auffinden einer Liste von evtl. anwendbaren Policies bzw. PolicySets verkürzt werden kann.

### Nachteile

- Da ca. 90 Prozent der Anfragen auf den Datentyp String gehen werden, ist es nur marginal schneller als wenn man die ganze Liste durchgeht.
- Die Liste von anwendbaren Policies, welche über den Datentyp String gefunden wird, ist sehr gross.

### Variante 3 Konfigurierbar

Es gibt zwei verschiedene Typen von Index, den StringIndex und den ComparableIndex. Der StringIndex erstellt dabei aus den einzelnen Characters des String einen Baum, während der ComparableIndex einen Baum aufbaut, in welchem nach "grösser" und "kleiner" geordnet wird.

Grundsätzlich sind alle Subject, Resource, Action und Environment die String als Datentyp besitzen mit dem StringIndex indexierbar. Alle, welche einen Datentyp, der Comparable implementiert, besitzen (z.B. Integer, String, Double), sind mit dem ComparableIndex indexierbar.

Ob eine Policy indexierbar ist, wird in verschiedenen Schritten geprüft:

- Schritt 1: Im Konfigurationsfile kann festgelegt werden, auf was indexiert wird. Man kann für Subject, Resource, Action und Environment angeben auf welche Werte der AttributeID (z.B. "Subject-role"), welche Datentypen (z.B. String oder Integer) und optional, auf welchen Issuer (z.B. "Firma XY") indexiert werden soll. Ebenfalls muss der Indexierungstyp (StringIndex oder ComparableIndex) angegeben werden.
- Schritt 2: Die Policies werden anhand der im Konfigurationsfile angegebenen Einstellungen geprüft. Passt eine Policy auf diese Einstellungen, wird diese in den Indexierungsbaum aufgenommen. Passt die Policy nicht darauf, wird diese an alle Knoten im Baum angehängt, so dass diese immer auffindbar ist.

### Vorteile

- Variable Nutzung der Indexierung dank anpassbarem Konfigurationsfile.
- Indexierung kann genauer spezifiziert werden.
- Verschiedene Index möglich (ComparableIndex und StringIndex).

### Nachteile

- Indexierung greift nur auf dem Target von PolicySet und PolicyType, nicht aber auf dem Target des RuleTypes.



### Variante 4 LDAP

Die ganzen Policies sollen mit LDAP verwaltet werden. Auf der OASIS Webseite gibt es einen Draft [LDAPDraft], in welchem angesprochen wird, wie man dies realisieren könnte. Das Dokument dazu ist aber nicht sehr informativ und Rücksprachen mit Seth Proctor (Entwickler von SunXACML) haben auch keine weiteren Ergebnisse geliefert.

#### Vorteile

- Mit LDAP könnte ein existierender Standard für das Indexieren von Policies verwendet werden.

#### Nachteile

- Keine ausgereifte Dokumentation.
- Draft von OASIS wurde nie weiterverfolgt.

## 1.2.3 Entscheidungsfindung

### Allgemein

Der Prozess der Entscheidungsfindung ist essentiell für den PDP. Dieser muss der XACML 2.0 Spezifikation [XACML 2.0] folgen. Wichtig dabei ist, wann welche Antwort zurückgegeben werden soll, was im Fehlerfall passieren soll, wo die Logik, welche entscheidet ob eine Policy auf einen Request matched, sein soll. Vor allem an welcher Stelle die Logik greifen soll, ist entscheidend. Dabei gibt es mehrere Möglichkeiten.

### Varianten

#### Variante 1 Logik in den Policies

Die ganze Logik für das Matchen einer Policy auf den Request passiert in einer evaluate Methode in der Policy. Ein Ablauf einer Entscheidungsfindung sieht folgendermassen aus:

Ablauf: Der PDP besitzt einen Root Combining Algorithmus. Ein eingehender Request wird zusammen mit den, vom Locator zurückgegebenen "möglichen" Policies dem Combining Algorithmus des PDP's übergeben. Dieser Combining Algorithmus ruft die evaluate Methode der einzelnen Policies auf und übergibt den Request. Das Matchen der Policy auf den Request findet in der evaluate Methode der Policy statt. Die Entscheidung der einzelnen Policies werden zurückgegeben und vom Combining Algorithmus des PDP verarbeitet. Zum Schluss gibt es genau eine Entscheidung, welche "Permit", "Deny", "Indeterminate" oder "NotApplicable" ist.

#### Vorteile

- Performant, da die Entscheidungsfindung direkt in den Policies stattfindet.

#### Nachteile

- Architektonisch nicht ganz sauber, da Datenklassen mit Logik angereichert werden.
- Bei einer Auswechslung der Datenklassen, müssen Anpassungen an mehreren Teilen vorgenommen werden.



### Variante 2 Logik in einer Targetmatcher Klasse

Die Logik für das Matchen von Policies auf einen Request ist in eine Targetmatcher Klasse ausgelagert. Diese Klasse wird von den Combining Algorithmen referenziert. Ein Ablauf einer Entscheidungsfindung sieht folgendermassen aus:

**Ablauf:** Der PDP besitzt einen Root Combining Algorithmus. Ein eingehender Request wird zusammen mit den, vom Locator zurückgegebenen, "möglichen" Policies dem Combining Algorithmus des PDP's übergeben. Dieser Combining Algorithmus benutzt die Targetmatcher Klasse um zu evaluieren, welche Policies auf den Request passen. Dabei besitzen die einzelnen Policies wieder einen Combining Algorithmus für die enthaltenen Policies oder Rules. Diese Combining Algorithmen referenzieren wiederum die Targetmatcher Klasse. Über die einzelnen Combining Algorithmen werden die Entscheidungen des Targetmatcher für die verschiedenen Policies zusammengefasst und weitergegeben. Zum Schluss gibt es genau eine Entscheidung, welche "Permit", "Deny", "Indeterminate" oder "NotApplicable" ist.

#### Vorteile

- Bessere Architektur, da die Entscheidungsfindung in einer Targetmatcher Klasse ausgelagert ist.
- Bei einer allfälligen Auswechslung der Datenklassen schneller anpassbar.

#### Nachteile

- Leichte Performanzeinbusse im Vergleich zur ersten Variante.

## 1.2.4 Auflösen der Policyreferenzen

### Allgemein

Einzelne Policies können lokale oder remote Policyreferenzen enthalten. Diese Referenzen müssen zu einem bestimmten Zeitpunkt aufgelöst werden. Es gibt mehrere denkbare Varianten.

### Varianten

#### Variante 1 Änderungsnachricht bei ändern von Policies auf Remote PDP

Die Remote Policies eines PDP werden, wie auch die lokalen Referenzen, während des Initialisieren des PDP aufgelöst. Falls Änderungen am Remote PDP vorgenommen werden, sendet dieser eine Änderungsnachricht, worauf der lokale PDP alle Remotepolicies noch einmal auflösen muss.



### Vorteile

- Während der Entscheidungsfindung müssen keine Remote und lokale Policies aufgelöst werden. Dadurch verbessert sich die Performanz des PDP's.
- Durch die Änderungsnachricht treten keine Inkonsistenzen auf.

### Nachteile

- Ein PDP muss wissen, welche anderen PDP's auf ihn referenzieren, um die Änderungsnachricht zu verschicken. Dadurch müsste ein Dienst zu Verfügung gestellt werden, welchen von PDP's abonniert werden kann.
- Trifft eine Änderungsnachricht auf dem PDP ein, muss dieser den laufenden Betrieb unterbrechen, damit die Remote Policies aufgelöst werden können und keine Inkonsistenzen auftreten.

### Variante 2 Jede Policyreferenz wird sofort aufgelöst

Für jede Policy, welche auf einen Request passen könnte, werden die enthaltenen Policyreferenzen sofort aufgelöst.

### Vorteile

- Es treten keine Inkonsistenzen auf, da lokale und Remote Referenzen während der Entscheidungsfindung sofort aufgelöst werden.

### Nachteile

- Grosser Performanzverlust beim Auflösen von remote Referenzen, da für jede einzelne Referenz eine Verbindung zum Remote PDP erstellt werden muss.

### Variante 3 Alle Policyreferenzen werden gemeinsam aufgelöst

Enthalten die vom Locator zurückgegebenen Policies Policyreferenzen werden diese dem Modul ReferenceLoader übergeben, welches die Referenzen auflöst und zurückgibt.

### Vorteile

- Es treten keine Inkonsistenzen auf, da die Policyreferenzen während der Entscheidungsfindung aufgelöst werden.
- Eine allfällige Verbindung zum Remote PDP muss nur einmal pro eingehenden Request erstellt werden.

### Nachteile

- Mässiger Performanzverlust für das Auflösen der Remote Referenzen, da die Verbindung wenn notwendig einmal pro eingehenden Request aufgebaut werden muss.



## 1.2.5 Verhalten bei fehlenden Attributen

**Allgemein** Tritt während der Entscheidungsfindung der StatusCode "MissingAttribute" auf muss der AttributeFinder angesprochen und weitere Attribute angefordert werden. Kann der AttributeFinder weitere Attribute liefern, muss der PDP diese in den Evaluierungsprozess einbeziehen. Falls nicht, wird die AuthorizationDecision an den PEP zurückgeschickt. Es gibt mehrere Möglichkeiten, um den AttributeFinder anzusprechen.

### Varianten

#### Variante 1 Fehlende Attribute sofort auflösen

Bei einem Auftreten eines fehlenden Attributes in einer Policy wird der AttributeFinder sofort angesprochen. Der AttributeFinder gibt, wenn möglich, das fehlende Attribut zurück und der PDP kann weiter evaluieren.

##### Vorteile

- Der Evaluierungsprozess wird nur einmal durchgeführt.

##### Nachteile

- Wenn bei jedem fehlenden Attribut der AttributeFinder angesprochen wird, kann dies zu Performanzverlusten führen, da eine allfällige Verbindung vom AttributeFinder zu einem PIP mehrmals aufgebaut werden müsste.

#### Variante 2 Fehlende Attribute zwischenspeichern

Während des Evaluierungsprozess auftretende "MissingAttributes" werden zwischengespeichert und am Ende des Evaluierungsprozesses, wenn notwendig, dem AttributeFinder übergeben. Der AttributeFinder gibt dem PDP einen neuen Request zurück, welcher nochmals evaluiert wird.

##### Vorteile

- Einfach zu implementieren, da die Evaluierung mit dem neuen Request nochmals neu gestartet wird.

##### Nachteile

- Falls fehlende Attribute auftreten, wird der Evaluierungsprozess nochmals von neuem gestartet, was die Performanz des PDP verlangsamt.
- Allenfalls müssen sehr viel mehr fehlende Attribute aufgelöst werden, als benötigt.



## 1.2.6 Abstraktion der Datenhaltung

*Allgemein* Der Zugriff auf die Datenhaltung sollte unabhängig von der spezifischen Datenbank realisiert werden. Ein relationales Schema für die Datenbank ist nicht notwendig, da lediglich BLOB's der einzelnen Policies gespeichert werden sollen. Um dies zu bewerkstelligen, gibt es verschiedene Möglichkeiten.

*Varianten*

### Variante 1 Spring JDBC-Templates

Für das Abstrahieren der Datenbank werden Spring JDBC-Templates verwendet.

#### Vorteile

- Transaktionsmanagement wird von Spring gemacht.
- Auf- und Abbau der Verbindung wird von Spring erledigt.
- Übersichtlicherer Code.

#### Nachteile

- Abhängig von der Datenbankschnittstelle.

### Variante 2 JDBC

Die Abstrahierung der Datenbank wird mit JDBC gemacht.

#### Vorteile

- Sehr performant

#### Nachteile

- Transaktionsmanagement muss selbst gemacht werden.
- Unübersichtlicherer Code.

### Variante 3 JPA

Die Abstrahierung der Datenbank wird mit JPA gemacht.

#### Vorteile

- Unabhängig von der Datenbankschnittstelle.

#### Nachteile

- Zu umfangreiche Technologie, für die effektiv benötigten Funktionalitäten.
- Grosser Konfigurationsaufwand



## 1.2.7 Möglichkeiten von Konfigurationsfiles

*Allgemein* In der HERAS<sup>AF</sup> XACML 2.0 Implementierung gibt es die Möglichkeit mehrere Module zu konfigurieren. So können zum Beispiel SQL Abfragen, die Indexierung und die JAXB Eigenschaften konfiguriert werden. Um dies zu ermöglichen gibt es mehrere Ansätze.

*Varianten*

### Variante 1 Eigenes XML-Schema und Konfigurationsfile

Es werden für die verschiedenen Konfigurationsmöglichkeiten eigene XML-Schema und dazugehörige Konfigurationsfiles geschrieben. Um diese Files auszuwerten, wird ein eigener Parser geschrieben.

#### Vorteile

- Einfach umzusetzen

#### Nachteile

- Verschiedene Konfigurationsfiles notwendig
- Konfiguration an verschiedenen Orten

### Variante 2 Spring Namensraum Erweiterung

Es können eigene Tags für Spring definiert werden. Dafür braucht es BeanDefinitionParser Klassen, welche diese erweiterten Tags parsen und Handler Klassen welche dafür zuständig sind, den BeanDefinitionParser beim Spring zu registrieren.

#### Vorteile

- Konfiguration komplett in Spring integrierbar
- Konfiguration an einem Ort

#### Nachteile

- Nur mit Spring zusammen verwendbar



## 1.2.8 Fehler in der XACML 2.0 Spezifikation

*Allgemein* In der XACML 2.0 Spezifikation [XACML 2.0] treten Fehler auf. In diesem Abschnitt wird auf diese Fehler eingegangen.

*Falsche Abhängigkeiten von Funktionen* In der XACML 2.0 Spezifikation [XACML 2.0] führt eine Abhängigkeit zwischen einzelnen Funktionen zu einer falschen Auswertung im PDP.

Das Problem tritt dann auf, wenn mehrere Higher-order bag Funktionen verschachtelt sind und zum Schluss eine Regex-Funktion aufgerufen wird.

Anhand drei pseudocode Beispielen, wird diese Problematik aufgezeigt:

### Pseudocode 1 AnyOf Funktion laut XACML 2.0 Spezifikation

Nach XACML 2.0 Spezifikation muss der folgende Pseudocode eingehalten werden:

```
anyOf.handle(p1, p2, p3){
    for( e : p3 ){
        p1.handle(p2, e)
    }
}
```

Abbildung 4 Pseudocode 1 AnyOf Funktion

### Pseudocode 2 AllOfAny Funktion

Der folgende Pseudocode hält die AllOfAny Funktion der XACML 2.0 Spezifikation:

```
AllOfAnyFunction.handle( string-regex-match, p2 , p3){
    for(e1 : p2){
        AllOfFunction.handle( string-regex-match, e , p3 ){
            for( e2 : p3 ){
                string-regex-match.handle(e1, e2)
            }
        }
    }
}
```

Abbildung 5 Pseudocode 2 AllOfAny Funktion



### Pseudocode 3 AnyOfAll Funktion

Der folgende Pseudocode hält die AnyOfAll Funktion der XACML 2.0 Spezifikation ein:

```

AnyOfAllFunction.handle( string-regex-match, p2 , p3){
    for(e1 : p3){
        AllOfFunction.handle( string-regex-match, e1, p2){
            for( e2 : p2){
                string-regex-match.handle(e2, e1)
            }
        }
    }
}

```

Abbildung 6 Pseudocode 3 AnyOfAll Funktion

### Der Fehler in der Spezifikation

Der oben erwähnte Pseudocode 1 (Abbildung 4) beschreibt nach XACML 2.0 Spezifikation, wie die AnyOf Funktion aussehen muss. Diese Funktion wird laut XACML 2.0 Spezifikation in den beiden Funktion AllOfAny und AnyOfAll verwendet.

Der grüne Teil des Pseudocode 2 (Abbildung 5) und der blaue Teil des Pseudocode 3 (Abbildung 6) sind Teil der AnyOf Funktion. An diesen Stellen zeigt sich das Problem, da die string-regex-match Funktion fest vorgibt, welcher Parameter der reguläre Ausdruck sein soll.

Wie ist es nun möglich, dass einmal der Element aus der übergebenen Liste an zweiter Stelle (grüner Pseudocode Ausschnitt) und das andere Mal an erster Stelle (blauer Pseudocode Ausschnitt) ist? In den Pseudocode Beispielen ist beim Aufruf der string-regex-match Funktion der Parameter e1 einmal an erster Stelle und einmal an zweiter Stelle.



## 1.3 Problematik von Statistiken

### *Evaluierungs- dauer*

Sinnvolle, aussagekräftige Statistiken über die Evaluierungsdauer der HERAS<sup>AF</sup> XACML 2.0 Implementierung zu erheben, ist sehr aufwändig. Dies darum, weil dazu eine grosse Anzahl an unternehmensnahen Policies definiert und auf den PDP deployt werden müssen. Beim Erstellen dieser Policies muss ausserdem darauf geachtet werden, dass sich diese Policies nicht gegenseitig stören. Dies erledigt ansonsten der PAP, welcher momentan noch nicht existiert. Ebenfalls muss die Indexierung so konfiguriert werden, dass die meisten der deployten Policies indexiert werden können. Die ganzen Rahmenbedingungen würden einen enormen Zeitaufwand bedeuten und den Rahmen dieser Arbeit sprengen.

Es wäre möglich, Annahmen zu treffen und so einfache Statistiken zu erstellen. Das Problem dabei ist aber, dass diese Statistiken sehr unrealistisch wären, weil diese auf eine Einfachheit herunter gebrochen würden, in welcher der Bezug zur Praxis verloren ginge.

### *RAM Verbrauch*

Ebenfalls problematisch stellt sich eine Erhebung von Statistiken für den RAM Verbrauch heraus. Das Problem hierbei liegt vor allem am Indexierungsalgorithmus und an der schwer vorherzusagenden Komplexität der Policies.

Das Problem des Indexierungsalgorithmus liegt daran, dass man nicht vorhersagen kann, wie gross ein Index wird. Dieser hängt sehr stark von der Komplexität und Grösse der deployten Policies ab.

Die Komplexität einer Policy setzt sich dabei aus folgenden Punkten zusammen:

- Anzahl Policysets in einem Policyset
- Anzahl Policies in einem PolicySet
- Anzahl Rules in einer Policy
- Anzahl Subject, Resource, Action und Environment Elemente in einem Target
- Komplexität der Condition in einer Rule

Um also den durchschnittlichen Verbrauch einer Policy zu berechnen, müssten viele unternehmensnahe Policies verwendet werden. Zum Verbrauch jeder dieser Policies käme dann noch der RAM Verbrauch für die Indexierung der einzelnen Policies dazu. Es ist möglich daraus eine repräsentative Statistik zu erstellen. Der Zeitrahmen dieser Arbeit ist allerdings zu knapp dafür.



## 1.4 Analyse der Indexierungsalgorithmen

<i>Allgemein</i>	In diesem Abschnitt wird anhand der Big Oh Notation aufgezeigt, welchen Nutzen für die Performanz die realisierten Indexierungsalgorithmen beim Auffinden der anwendbaren Policies haben. Es ist zu beachten, dass diese Statistiken nur für ideale Fälle gelten. Dies bedeutet, dass die Konfiguration des Indexes so gewählt ist, dass alle Policies auch indexierbar sind und eine Anfrage auf den Index wenige potentielle Policies zurückliefert.
<i>Vergleichswert</i>	Als Vergleichswert zum Auffinden der Policies dient $O(n)$ . Dies erreicht man wenn alle Policies in eine Liste gespeichert sind und durchgegangen werden müssen.
<i>Anzahl Kombinationen des Request</i>	Um anwendbare Policies aufzufinden, wird aus den im Request enthaltenen Subject, Resource, Action und Environment alle möglichen Kombinationen gebildet und diese auf dem Index gesucht. Die Anzahl der Kombinationen wird als $O(k)$ bezeichnet.
<i>String Index</i>	Das Auffinden im String Index ist gleichzusetzen mit dem Auffinden in einer Map, da jeder Knoten im String Index ein Charakter bildet. Damit ist das Auffinden mit $O(c)$ zu bewerten. Zusammen mit den möglichen Kombinationen gilt für das Auffinden in einem String Index: $O(k) * O(c)$
<i>Comparable Index</i>	Das Auffinden im Comparable Index bildet sich aus dem bekannten Wert $O(\log n)$ , welches für das Aufsuchen in einem AVLTree benötigt wird. Zusammen mit den möglichen Kombinationen gilt für das Auffinden in einem Comparable Index: $O(k) * O(\log n)$
<i>Auswertung</i>	Für die Indexierungsalgorithmen gilt also: <ul style="list-style-type: none"> <li>• Wenn <math>O(k) * O(c) &lt; O(n)</math>, lohnt sich das Auffinden der Policies über einen String Index.</li> <li>• Wenn <math>O(k) * O(\log n) &lt; O(n)</math>, lohnt sich das Auffinden der Policies über einen Comparable Index.</li> </ul> <p>Da es sehr wahrscheinlich ist, dass ein Request mehrere Subject, Resource, Action und Environment Elemente enthält, bieten die Indexierungsalgorithmen erst dann ein Geschwindigkeitsvorteil, wenn der PDP viele Policies beinhaltet.</p>



## 1.5 Analyse SunXACML

<i>Allgemein</i>	<p>In diesem Abschnitt wird die HERAS<sup>AF</sup> XACML 2.0 Implementierung mit der Referenzimplementierung SunXACML verglichen. Dabei wird auf einzelne Erweiterungen und Einschränkungen, sowie die Performanz einzelner Teile eingegangen.</p>
<i>Unmarshall</i>	<p><b>Implementierung</b></p> <p>XML Input wird mithilfe eines DOM Parsers in ein DOM Baum umgewandelt. Anschliessend werden die Informationen aus diesem ausgelesen und daraus die Objekte erstellt</p> <p><b>Bewertung</b></p> <p>Das Parsen mit einem DOM Parser ist langsam, deshalb wurde diese Lösung als ungeeignet eingestuft.</p>
<i>Marshal</i>	<p><b>Implementierung</b></p> <p>Jedes Objekt kann sich selbst parsen. Dazu schreibt es die eigene XML Representation auf einen Stream und ruft an entsprechender Stelle die Kindelemente auf. Um das XML leserlich auszugeben, wird immer mitgegeben wie viele Tabulatoren dem Element vorgeschoben werden müssen. Namespace Informationen werden nicht ausgegeben.</p> <p><b>Bewertung</b></p> <p>Die Analyse ergab, dass dieses Vorgehen sehr performant ist, jedoch schlecht konfigurierbar und aus der Design Sicht nicht optimal. Die Implementierung lässt sich nicht konfigurieren. Da keine Namespace Informationen ausgegeben werden, lässt sich das ausgegebene XML nur dann in ein anderes XML Dokument einbinden, wenn nachträglich Namespaces eingefügt werden.</p>
<i>Indexierung und Persistierung</i>	<p><b>Implementierung</b></p> <p>Das Indexieren und Persistieren der Policies geschieht in Policy Locator Modulen. Diese können beim Erstellen des PDPs übergeben werden. Während dem Evaluieren werden die Policy Locator Module nacheinander aufgerufen. Falls ein Policy Locator Modul eine, zum Request passende, Policy enthält, wird diese vom Locator Modul zurückgegeben.</p> <p><b>Bewertung</b></p> <p>Die Analyse ergab, dass ein Einbinden von mehreren Policy Locator Modulen die Performanz des PDPs erheblich beeinträchtigt, da dadurch keine Indexierung über alle Policies mehr möglich ist.</p>



### *Attribute Auflösung*

#### **Implementierung**

Attribute werden mithilfe von AttributeFinder Modulen aufgefunden. Sobald ein passendes Attribut gefunden wurde, wird dieses zurückgegeben. In einer Liste können mehrere Attribut Finder Module definiert werden, die nacheinander aufgerufen werden.

#### **Bewertung**

Die Analyse ergab, dass die AttributFinder Logik, die den AttributDesignator repräsentiert, auch direkt in der jeweiligen Klasse implementiert werden kann. Es wird lediglich ein AttributFinder benötigt um fehlende Attribute beim PIP anzufragen.

### *Policy/PolicySet Referenzen Auflösung*

#### **Implementierung**

Referenzen auf Policies oder PolicySets werden wie normale Policies angesprochen und in der evaluate Methode aufgelöst.

#### **Bewertung**

Die Analyse ergab, dass diese Lösung zu inkonsistenten PolicySets führen kann. Wenn ein PolicySet ein weiteres PolicySet enthält und beide auf die gleiche Policy referenzieren, kann es vorkommen, dass sich die referenzierte Policy ändert nachdem ein PolicySet die Referenz aufgelöst hat, das zweite PolicySet jedoch noch nicht.

### *Mehrere Resources in Request verwenden*

#### **Implementierung**

Die Logik des Profiles für mehrere Resources ist direkt im PDP implementiert und nicht wie im Profile angegeben beim ContextHandler.

#### **Bewertung**

Dieses Vorgehen vereinfacht die Logik des ContextHandlers, verschlechtert jedoch die Performanz falls der ContextHandler die Policies vorher schon aufgeteilt hat.

### *Entscheidungs- findung*

#### **Implementierung**

Die Entscheide werden weitestgehend so implementiert, dass jedes Element die eigene Logik enthält und die Kind Elemente aufrufen kann.

#### **Bewertung**

Die Analyse ergab, dass diese Implementierung aus der Design Sicht nicht ganz sauber ist, da die Datenklassen Logik enthalten.

### *Environment Zeitfelder*

#### **Implementierung**

Als Zeit der Anfrage wird die Zeit genommen, zu der die Anfrage im PDP evaluiert wird. Wahlweise kann auch ein Caching derselben verwendet werden.

#### **Bewertung**

Die Analyse ergab, dass dies eine Abweichung von der Spezifikation ist. Laut Spezifikation wird der Zeitpunkt einer Anfrage im Context Handler bestimmt und nicht im PDP.



*Konfiguration*

**Implementierung**

Der PDP wird mithilfe eines PDPCConfig Objektes, welches dem PDP Konstruktor übergeben wird, erstellt. Dieses legt die konfigurierten Module fest.

**Bewertung**

Die Bewertung hat ergeben, dass diese Lösung schnell unübersichtlich werden kann, wenn sehr viele Konfigurationsmöglichkeiten bestehen.



## 2 Architektur

### Allgemein

Der Aufbau der XACML 2.0 Implementierung ist Modular gehalten. Dies ist ein wichtiger Ansatz um allfällige Module auszutauschen. In der Abbildung 7 werden die Abhängigkeiten der einzelnen Module aufgezeigt.

### Modul Design

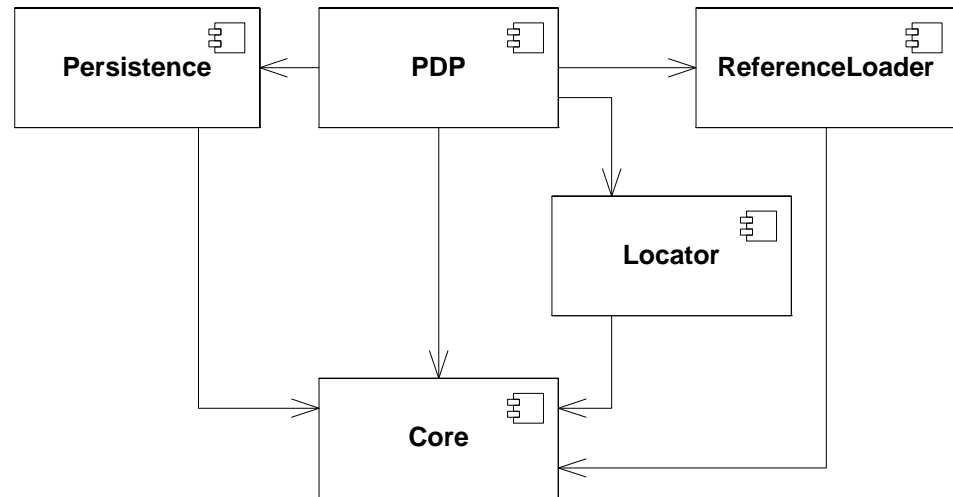


Abbildung 7 Modul Design HERAS<sup>AF</sup> XACML 2.0 Implementierung

### PDP

Der PDP enthält die Schnittstelle zum Ansprechen des HERAS<sup>AF</sup> PDP und steuert den Ablauf zwischen den einzelnen Modulen.

### ReferenceLoader

Der ReferenceLoader ist dafür zuständig, die übergebenen Policyreferenzen aufzulösen und zurückzugeben. Um Remote Referenzen aufzulösen, muss der ReferenceLoader eine Verbindung mit einem externen PDP aufbauen. Für lokale Referenzen ist dies nicht notwendig. Aus zeitlichen Gründen wurde der ReferenceLoader in dieser Implementierung nur gemockt. Das Modul wurde aber erstellt um zu einem späteren Zeitpunkt einen funktionsfähigen ReferenceLoader in die bestehende Struktur zu integrieren.

### Locator

Der Locator enthält die entwickelten Indexierungsalgorithmen um die Policies möglichst performant aufzufinden.

### Persistence

Das Modul Persistence wird vom PDP benötigt um die vom PAP deployten Policies zu persistieren, sowie die Policies beim Initialisieren des PDP's ins RAM zu laden.



## Core

Das Core Modul enthält neben den von JAXB generierten Policy- und Context-Klassen auch die implementierten Funktionen, Datentypen und Combining-Algorithmen.

Zusätzlich beinhaltet es die Logik für das Auffinden von Attributen und das Matchen eines Targets einer Policy.

### Datenfluss

### Instantiierung und Initialisierung des HERAS<sup>AF</sup> PDP

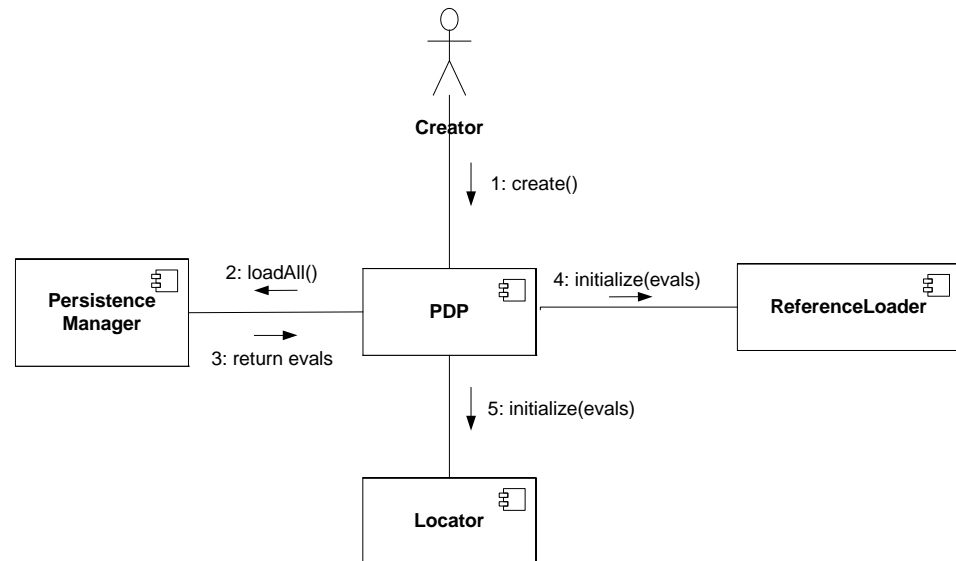


Abbildung 8 Initialisierung HERAS<sup>AF</sup> PDP

### Beschreibung

1. Der HERAS<sup>AF</sup> PDP wird erstellt.
2. Die loadAll Methode des PersistenceManager wird aufgerufen.
3. Der PersistenceManager liefert die auf der Datenbank gehaltenen Policies zurück.
4. Der HERAS<sup>AF</sup> PDP ruft die initialize Methode des ReferenceLoaders auf und übergibt ihm die erhaltenen Policies.
5. Der HERAS<sup>AF</sup> PDP ruft die initialize Methode des Locators auf und übergibt die vom PersistenceManger erhaltenen Policies. Die Indexierung der Policies wird erstellt.



## Evaluieren eines Request

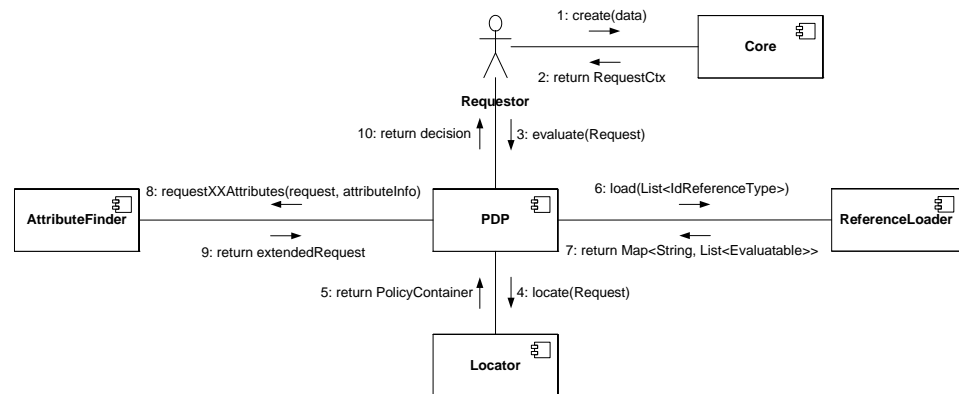


Abbildung 9 Evaluierung HERAS<sup>AF</sup> PDP (Requestor = Applikation mit der Absicht ein Request auf dem PDP zu evaluieren)

## Beschreibung

1. Der Requestor ruft die create Methode der RequestCtxFactory im Core auf. Dabei werden als Parameter die Daten für den zu erstellenden Request übergeben.
2. Die RequestCtxFactory gibt einen RequestCtx zurück.
3. der Requester ruft die evalute Methode des PDP auf und übergibt einen RequestCtx als Parameter.
4. Der PDP ruft die locate Methode des Locators auf und übergibt den Request als Parameter.
5. Der Locator holt aus dem Index eine, auf den Request passende Liste von Policies, erstellt einen PolicyContainer, welcher eine Liste von Policies und eine Liste von Remotereferenzen enthält und gibt diesen zurück.
6. Der PDP ruft die load Methode des ReferenceLoaders auf und übergibt als Parameter eine Liste von IdReferenceTypes.
7. Der ReferenceLoader löst die übergebenen IdReferenceType auf, speichert diese in eine Map und gibt die Map zurück. Danach evaluiert der PDP den Request anhand der geladenen lokalen und Remote Policies.
8. Fehlt im Request ein gesuchtes Attribut, wird dieses mithilfe des AttributeFinders beim PIP angefragt.
9. Der AttributeFinder fügt die gesuchten AttributeValues dem Request hinzu und gibt die Werte zurück.
10. Der PDP gibt die Authorization Decision zurück.

### Konfigurationsdateien

Für die einzelnen Module sind eigene Konfigurationsdateien geschrieben worden.

### Verweis

Der genaue Aufbau, sowie die Funktionalitäten der einzelnen Module und Konfigurationsfiles sind im Developer's Guide zu dieser Arbeit beschrieben. [DOH07DADev]



## 3 Design

### Inhalt

Dieses Kapitel behandelt die einzelnen Module der HERAS<sup>AF</sup> XACML 2.0 Implementierung. Es wird vor allem auf die einzelnen Herausforderungen und Lösungen in den Modulen eingegangen, sowie ein grober Überblick des Designs gegeben. Das Design im Detail wird im Developer's Guide dieser Arbeit [DOH07DADev] beschrieben.

### 3.1 Modul Core

#### Allgemein

Das Core Modul ist der Kern der HERAS<sup>AF</sup> XACML 2.0 Implementierung. Es bietet alle benötigten Datenklassen, um XACML 2.0 Requests, Responses und Policies zu erstellen. Ausserdem enthält es die in der XACML 2.0 Spezifikation [XACML 2.0] definierten Funktionen, Combining Algorithmen und Datentypen.

Ebenfalls im Core Modul enthalten sind die Entscheidungsfindung, sowie der gemockte AttributeFinder.

#### Package Diagramm

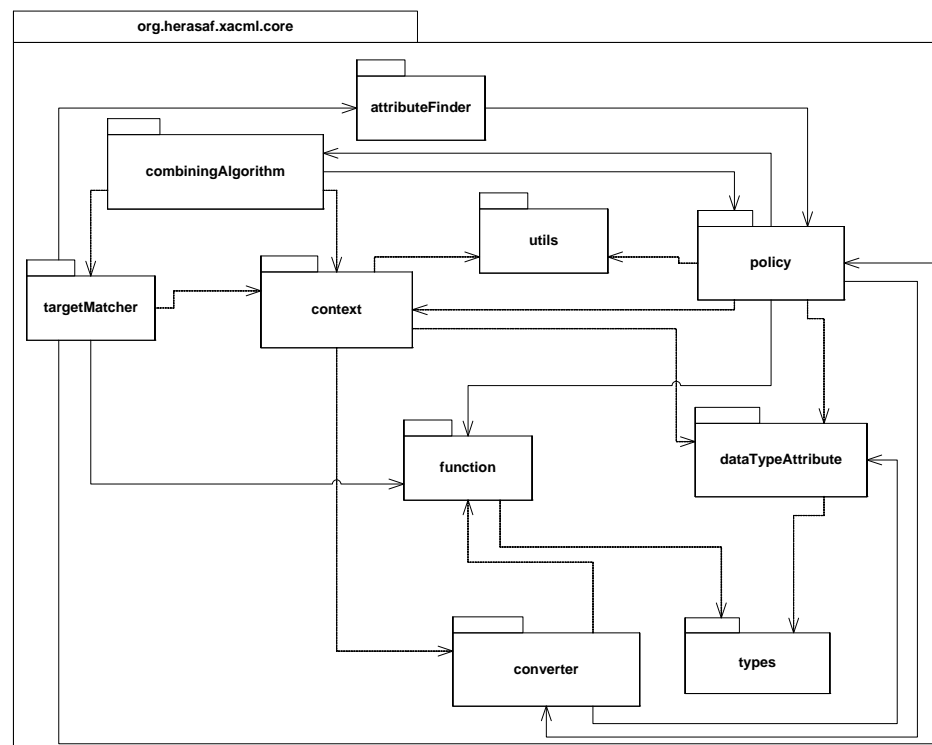


Abbildung 10 Diagramm Core Modul

#### Package Beschreibung

##### policy

Das Package policy enthält all die Datenklassen um ein XACML 2.0 Policy oder PolicySet zu erstellen. Ebenfalls werden die enthaltenen Klassen für das Marshalling und Unmarshalling verwendet.

##### attributeFinder

Das Package attributeFinder enthält das Interface für den AttributeFinder, sowie dessen Implementierungen.

**context**

Das Package context enthält all die Datenklassen um ein XACML 2.0 Request oder Response zu erstellen. Ebenfalls werden die enthaltenen Klassen für das Marshalling und Unmarshalling verwendet.

**converter**

Das Package converter enthält alle wichtigen Konverterklassen, welche während dem Marshalling und Unmarshalling Prozess benötigt werden.

**combiningAlgorithm**

Das Package Combining Algorithm enthält, die in der XACML 2.0 Spezifikation definierten Rule- und Policy Combining Algorithmen, um zu entscheiden wie mehrere Rules oder Policies kombiniert werden.

**function**

Das Package function enthält, die in der XACML 2.0 Spezifikation [XACML 2.0] definierten Funktionen.

**dataTypeAttribute**

Das Package dataTypeAttribute enthält alle die dataTypeAttribute Klassen, welche dafür zuständig sind, die String Representationen eines Datentypen in den entsprechenden Datentyp umzuwandeln.

**types**

Das Package types enthält die in der XACML 2.0 Spezifikation [XACML 2.0] definierten Datentypen.

**targetMatcher**

Das Package targetMatcher enthält die Klassen, welche dafür zuständig sind um die Targets der einzelnen Evaluatables und Rules mit einem Request zu matchen.

**utils**

Das Package utils enthält die verschiedenen Utilityklassen welche im Core verwendet werden.



### 3.1.1 Herausforderungen

**Allgemein** In diesem Abschnitt wird auf die einzelnen Herausforderungen, die während dem Entwickeln des Core Moduls auftraten, eingegangen. Ebenfalls wird auch, anhand der im Teil II Abschnitt 1.2 besprochenen Schwerpunkte, eine geeignete Lösung gefällt und diskutiert.

**Context und Policy Klassen** Im Teil II Abschnitt 1.2.1 werden zwei Varianten besprochen, wie die Context- und Policy-Klassen realisiert werden könnten. Zum einen besteht die Möglichkeit diese Klassen komplett selber zu implementieren und passende Funktionalitäten für Unmarshalling und Marshalling zu entwickeln, zum Andern können die Context- und Policy-Klassen mit JAXB 2.1 zu generiert werden, wobei Marshalling und Unmarshalling Methoden von JAXB zur Verfügung gestellt werden.

#### Diskussion der gewählten Lösung

Als Lösung wurde die Variante mit dem generieren der Context- und Policy-Klassen gewählt. Dies ist vor allem mit dem zeitlichen Aspekt zu erklären. Der Aufwand um diese Klassen selbst zu schreiben und geeignete Unmarshalling und Marshalling Methoden zu entwickeln, hätte erheblich mehr Zeit in Anspruch genommen und andere wichtigere Funktionalitäten wären entfallen. Denkbar aber wäre, dass man zu einem späteren Zeitpunkt diese Klassen selbst implementieren könnte.

**Entscheidungsfindung** Im Teil II Abschnitt 1.2.3 werden zwei Varianten besprochen, wie die Entscheidungsfindung realisiert werden könnte. Zum einen besteht die Möglichkeit der Logik, um das Matchen in der Policy zu realisieren, zum anderen könnte die Logik in eine Targetmatcher Klasse ausgelagert werden.

#### Diskussion der gewählten Lösung

Als Lösung wurde die Variante mit dem Auslagern der Logik in eine Targetmatcher Klasse gewählt. Diese wurde gewählt, weil damit die Architektur der HERAS<sup>AF</sup> XACML 2.0 Implementierung schöner ist und bei einer allfälligen Änderung oder Auswechslung der Datenklassen weniger Anpassungen gemacht werden müssen.

**Auflösen der Policyreferenzen** Im Teil II Abschnitt 1.2.4 werden drei Varianten besprochen, wie das Auflösen der Policyreferenzen aussehen könnte. Diese könnten nach einer Änderungsnachricht des externen PDP's neu aufgelöst werden, bei Auftreten jeder Policyreferenz neu aufgelöst werden oder pro Request alle auftretenden Referenzen zusammen auflösen.



### Diskussion der gewählten Lösung

Die dritte Variante wurde als Lösung umgesetzt. Dies ist damit zu begründen, dass sie, um die Referenzen aufzulösen, nur eine Verbindung zum externen PDP erstellt um Remote Referenzen aufzulösen und deshalb wesentlich performanter ist, als wenn für jede Remote Referenz eine Verbindung erstellt wird. Die dritte Variante wäre dann am besten, wenn ein externer PDP kaum Änderungen hat. Dann müssten die Remote Referenzen nur beim Initialisieren des eigenen PDP's aufgelöst werden. Hat aber ein externer PDP viele Änderungen, ist der sich im Betrieb befindende PDP für jede Änderung eine gewisse Zeit nicht erreichbar. Dies könnte zur Unbrauchbarkeit des PDP's führen.

*Verhalten bei  
fehlenden  
Attributen*

Im Teil II Abschnitt 1.2.5 werden zwei Varianten besprochen, wie innerhalb des Evaluierungsprozesses beim Auftreten von fehlenden Attributen reagiert werden kann. Es gibt die Möglichkeit bei jedem Auftreten, dieses sofort über den AttributeFinder aufzulösen oder man kann alle fehlenden Attribute zwischenspeichern und zum Schluss der Evaluierung diese, wenn notwendig, zusammen auflösen.

### Diskussion der gewählten Lösung

Die erste Variante wurde als die bessere erachtet und wurde implementiert. Die Hauptgründe sind darin zu finden, der Evaluierungsprozess nur einmal durchlaufen wird und dass keine fehlenden Attribute auf Vorrat aufgelöst werden. Bei der zweiten Variante werden alle fehlenden Attribute zwischengespeichert und dem AttributeFinder als Liste übergeben. Dies führt dazu, dass Attribute aufgelöst werden, welche für den Entscheidungsfindungsprozess nicht mehr notwendig sind.

*Grenzen von  
JAXB*

### Annotierungen werden nicht vererbt

Anfangs der Arbeit wurde die Überlegung gemacht, dass die XACML Datenklassen mit JAXB generiert werden. An den generierten Klassen sollen dabei keine Anpassungen gemacht werden. Die effektiven im Core verwendeten Klassen sollten von diesen generierten Klassen ableiten und allfällige Anpassungen sollten in den Abgeleiteten Klassen gemacht werden. Bei einer späteren Umstellung auf XACML 3.0, welches momentan von OASIS erarbeitet wird, wären so Anpassungen an der HERAS<sup>AF</sup> XACML 2.0 Implementierung einfacher möglich.

Um dies zu ermöglichen, müssten aber alle Annotierungen vererbt werden, damit das Marshalling und Unmarshalling mit den abgeleiteten Klassen auch möglich ist. Theoretisch ist es möglich, dass Annotierungen vererbt werden. Dies müsste allerdings in jeder Annotierung angegeben werden. In JAXB 2.1 wird dies nicht gemacht.

Aufgrund dessen wurde entschieden, dass direkt mit den generierten Klassen gearbeitet wird, diese aber nach Möglichkeit nicht angepasst werden müssen.



### Interfaces für einzelne generierte Klassen nicht möglich

Um einzelne der generierten Context- und Policy-Klassen besser nutzen zu können macht es Sinn, dass diese ein zusätzliches Interface implementieren. Dies betrifft vor allem die Klassen Policy SetType und PolicyType, da diese nahezu identisch behandelt werden können.

Im JAXB Bindingfile ist es möglich anzugeben, dass alle generierten Klassen ein bestimmtes Interface implementieren. Es ist aber nicht möglich, dass nur einzelne Klassen ein Interface implementieren.

Deshalb war es nicht mehr möglich, Anpassungen an den generierten Klassen zu vermeiden. Darum wurde entschieden, dass die Context- und Policy-Klassen einmal generiert und danach angepasst werden.

## 3.2 Modul PDP

### Allgemein

Das Modul PDP bietet die Schnittstelle an um den HERAS<sup>AF</sup> PDP zu nutzen. Ausserdem steuert es den Ablauf zwischen den einzelnen Modulen der HERAS<sup>AF</sup> XACML 2.0 Implementierung.

### Diagramm

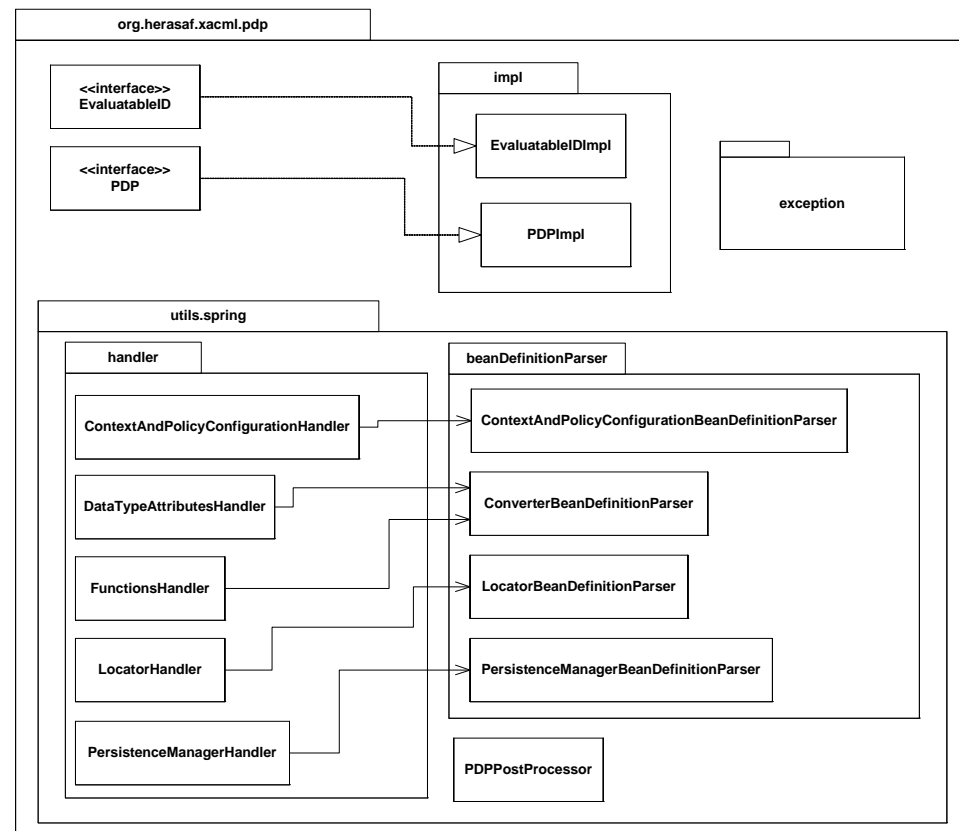


Abbildung 11 Diagramm PDP Modul

*Beschreibung***Impl**

Das Package impl enthält die PDPIml Klasse, welche die evaluate() Methode anbietet und den ganzen Ablauf des Evaluierungsprozesses steuert.

**handler**

Das Package handler enthält diejenigen Klassen welche für die Spring Namensraum Erweiterungen die passenden beanDefinitionParser bei Spring registriert.

**beanDefinitionParser**

Das Package beanDefinitionParser enthält die Klassen, welche zuständig sind bei Spring Namensraum Erweiterungen im ApplicationContext integrierte Tags zu parsen und entsprechende Werte zu setzen.



### 3.2.1 Herausforderungen

*Allgemein* In diesem Abschnitt wird auf die einzelnen Herausforderungen, die während dem Entwickeln des PDP Moduls auftraten, eingegangen. Ebenfalls wird auch, anhand der im Teil II Abschnitt 1.2 besprochenen Schwerpunkte, eine geeignete Entscheidung gefällt und diskutiert.

*Möglichkeiten von Konfigurationsfiles* Im Teil II Abschnitt 1.2.7 wird in zwei Varianten besprochen, wie eigene Konfigurationsfiles geschrieben und verwendet werden. Dabei gibt es die Möglichkeit ein eigenes Konfigurationsfile mit einem dazu gehörigen XML-Schema zu erstellen oder die Spring Namensraum Erweiterungen zu nutzen, um alle Konfigurationen zentral im ApplicationContext zu halten.

#### Diskussion der gewählten Lösung

Die Wahl der einzusetzenden Lösung ist auf die zweite Variante gefallen. Dies ist damit zu begründen, dass die Konfiguration an einem zentralen Ort durchgeführt werden kann und komplett in das ohnehin zu verwendende Spring integrierbar ist. Die zweite Variante würde einzig den Vorteil bieten, dass die Konfiguration nicht nur mit Spring verwendet werden können. Da Spring aber ein fester Bestandteil der HERAS<sup>AF</sup> XACML 2.0 Implementierung ist, ist die Notwendigkeit eines Spring unabhängigen Konfigurationsfile fraglich.

## 3.3 Modul Persistence

*Allgemein* Das Modul Persistence abstrahiert die Datenhaltung. In der HERAS<sup>AF</sup> XACML 2.0 Implementierung ist die Persistierung sehr einfach gehalten, da die darunter liegende Datenhaltung nur BLOB's der einzelnen Policies persistieren muss. Das heisst auf ein relationales Schema wird komplett verzichtet werden.

*Klassen Diagramm*

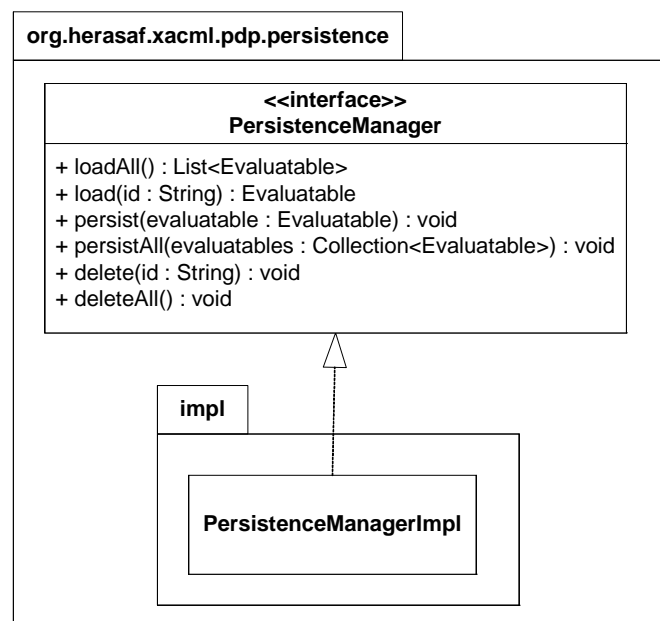


Abbildung 12 Diagramm Persistence Modul



*Beschreibung  
Packages und  
Intrafaces*

### **Interface PersistenceManager**

Das Interface PersistenceManager abstrahiert die Datenhaltung und bietet Methoden zum Persistieren, Löschen und Laden von Policies an.

### **Package impl**

Das Package impl enthält die Implementierung des PersistenceManager.

## **3.3.1 Herausforderungen**

*Allgemein*

In diesem Abschnitt wird auf die einzelnen Herausforderungen, die während dem Entwickeln des Persistence Moduls auftraten, eingegangen. Ebenfalls wird auch, anhand der im Teil II Abschnitt 1.2 besprochenen Schwerpunkte, eine geeignete Entscheidung gefällt und diskutiert.

*Abstraktion der  
Datenhaltung*

Im Teil II Abschnitt 1.2.6 werden drei Varianten besprochen, wie die Datenhaltung abstrahiert werden könnte. Eine solche Abstraktion kann mit Spring JDBC-Templates, mit JDBC oder mit JPA realisiert werden.

### **Diskussion der gewählten Lösung**

Als beste Variante wurde die erste, mit den Spring JDBC-Templates erachtet, welche auch umgesetzt wurde. Ausschlaggebend dafür waren das standardmässig zur Verfügung gestellte Auf- und Abbauen der Verbindung, sowie das Transaktionsmanagement. In der Variante mit dem reinen JDBC hätte man dies selbst realisieren müssen. Die dritte Variante wurde nicht berücksichtigt, weil sie zu schwergewichtig für eine derartige triviale Datenhaltung ist.



## 3.4 Modul Locator

### Allgemein

Das Modul Locator enthält die Logik um potentiell anwendbare Policies aufzufinden. Um dies zu erreichen, wurden verschiedene Indexierungsalgorithmen entwickelt. Diese werden in diesem Abschnitt etwas genauer erläutert.

### Klassen- diagramm

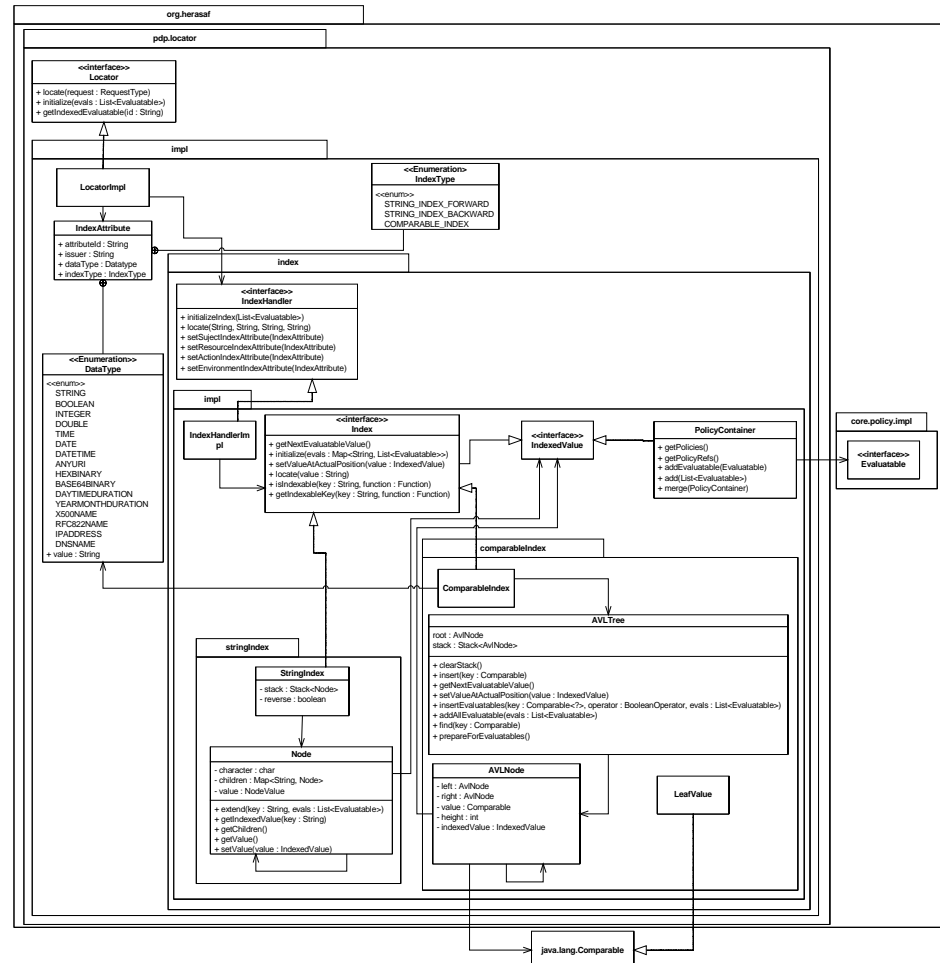


Abbildung 13 Diagramm Locator Modul

### Beschreibung

#### `org.herasaf.pdp.locator`

Das Package `org.herasaf.pdp.locator` enthält das Interface für den zu implementierenden Locator.

#### `org.herasaf.pdp.locator.impl`

Das Package `org.herasaf.pdp.locator.impl` enthält die Implementierung des Locators, sowie eine Datenklasse, welche für die Konfiguration des Index verwendet wird.

#### `org.herasaf.pdp.locator.impl.index`

Das Package `org.herasaf.pdp.locator.impl.index` enthält das Interface für den zu implementierenden IndexHandler. Dieser ist zuständig um die einzelnen



Indexe zu initialisieren, sowie Policies darauf zu suchen.

#### **org.herasaf.pdp.locator.impl.index.impl**

Das Package org.herasaf.pdp.locator.impl.index.impl enthält die Interfaces für die zu implementierenden Indexe, die Implementierung des IndexHandlers, sowie den PolicyContainer.

#### **org.herasaf.pdp.locator.impl.index.impl.stringIndex**

Das Package org.herasaf.pdp.locator.impl.index.impl.stringIndex implementiert einen Index welcher auf String-Equals-Funktionen, sowie auf String-Regex-Funktionen indexieren kann.

#### **org.herasaf.pdp.locator.impl.index.impl.comparableIndex**

Das Package org.herasaf.pdp.locator.impl.index.impl.comparableIndex implementiert einen Index welcher auf alle Typen, welche Comparable implementieren die Funktionen less, less-than, greater, greater-than und equals indexieren kann. Ebenfalls enthält dieses Package den AVLTree, mit welchem dieser Index aufgebaut wird.

### 3.4.1 Herausforderungen

#### *Allgemein*

In diesem Abschnitt wird auf die einzelnen Herausforderungen, die während dem Entwickeln des Locator Moduls auftraten, eingegangen. Ebenfalls wird, anhand der im Teil II Abschnitt 1.2 besprochenen Schwerpunkte, eine geeignete Entscheidung gefällt und diskutiert.

#### *Indexierung*

Im Teil II Abschnitt 1.2.2 werden vier Varianten besprochen, wie die Indexierung für die Policies aussehen könnte. So wäre es möglich aus Kombinationen der einzelnen Datentypen, von Subject, Ressource, Environment und Action, einen Hashwert zu bilden. Aus diesen Kombinationen einen Baum aufbauen oder die Indexierung konfigurierbar zu gestalten. Ebenfalls könnte man den Standard LDAP zur Verwaltung der Policies einsetzen.

#### **Diskussion der gewählten Lösung**

Als Lösung für die HERAS<sup>AF</sup> XACML 2.0 Implementierung wurde die dritte Variante gewählt. Dies vor allem aus den Gründen, dass über die Konfiguration die Indexierungseinstellungen angepasst werden können und so offener für eine Verwendung in verschiedenen Unternehmen ist. Um LDAP zur Verwaltung von Policies zu verwenden, wären mehr Informationen notwendig gewesen, als der LDAP profile draft [LDAPDraft] liefert.



### 3.4.1.1 Indexierungsalgorithmen

#### Allgemein

Die HERAS<sup>AF</sup> XACML 2.0 Implementierung implementiert zwei Policy Indexierungsalgorithmen. In diesem Abschnitt wird besprochen, wie diese Algorithmen funktionieren, welche Möglichkeiten sie bieten und wie diese implementiert wurden.

#### Schwierigkeit

Die Schwierigkeit beim Entwickeln eines Indexierungsalgorithmus ist, dass im Gegensatz zur normalen Indexierung, die Rollen vertauscht sind. So enthält z.B. eine Policy die Regular Expression, während diese sonst in einer Anfrage vorkommen würde.

Ein Beispiel zur Veranschaulichung:

**Normale Indexierung** "Um alle Worddokumente auf dem eigenen PC zu suchen, wird das Windows Suchfenster geöffnet und eine Suchanfrage mit "\*.doc" erstellt. Als Ergebnis werden alle Dokumente mit der Endung ".doc" gefunden."

**Indexierung in unserem Falle** "Um den Zugriff auf irgendein Dokument mit der Endung ".doc" zu bekommen, muss eine Policy erstellt werden, welche den Regex "\*.doc" enthält. Eine Anfrage ist genau auf das Dokument spezifiziert, auf welches zugegriffen werden soll."

Ein Target einer Policy kann auf viele verschiedene Arten matchen. ODER, sowie UND Beziehungen zwischen den Attributen sind möglich. Ebenfalls können die Datentypen der einzelnen Attribute verschieden sein. Alle diese Punkte machen es sehr schwer, einen schnellen Indexierungsalgorithmus zu finden.

#### Realisierung

Um die Schwierigkeiten zu lösen, geeignete Indexierungsalgorithmen zu entwickeln, wird die Indexierung der Policy in Teile zerlegt. Die einzelnen Elemente des Targets (Subject, Resource, Action und Environment) werden einzeln indexiert und zusammengesetzt (Abbildung 14). Dies ermöglicht es, für die einzelnen Elemente verschiedene Indexierungsalgorithmen zu verwenden. Um zu erlauben, dass verschiedene Unternehmen, die Indexierung auf die unternehmensspezifischen Daten anpassen, kann die Indexierung konfiguriert werden.

Um sicher zu stellen, dass Policies welche nicht indexiert werden können dennoch als potentiell anwendbare Policies zurückgegeben werden, sind diese jedem Knoten im Index angehängt.

Um auf die Möglichkeit verschiedener Datentypen und Matchfunktionen reagieren zu können, wurden zwei verschiedene Index entwickelt. Diese werden in den Abschnitten 3.4.1.1.1 und 3.4.1.1.2 behandelt.

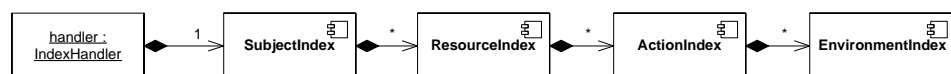


Abbildung 14 Indexierungskette



### 3.4.1.1.1 StringIndex

#### *Allgemein*

Der StringIndex wird verwendet für Policies, welche im Subject-, Ressource-, Action- und EnvironmentMatch Element als Matchfunktion eine string-equals oder eine string-regexp Funktion enthalten.

Reguläre Ausdrücke sind sehr schwierig zu indexieren. Aus diesem Grund werden nur diejenigen regulären Ausdrücke indexiert, welche einen Stern am Anfang oder am Ende des Ausdruckes enthalten. Für diese zwei Möglichkeiten gibt es verschiedene Algorithmen, den Forward StringIndex und den Reverse StringIndex.

Um einen Index anzubieten, in welchem der Stern an jeder erdenklichen Stelle des Ausdruckes vorkommen darf, müssten weitere Forschungsarbeiten unternommen werden, wobei fraglich ist, ob es überhaupt möglich wäre, diese in einem Index zu vereinen. Dies würde allerdings den Rahmen dieser Diplomarbeit sprengen.

#### *Forward StringIndex*

Der Forward StringIndex ist eine Spezialisierung des StringIndex. Dieser Index kann reguläre Ausdrücke mit einem Stern an der letzten Stelle behandeln.

#### *Backward StringIndex*

Der Reverse StringIndex ist eine Spezialisierung des StringIndex. Dieser Index kann reguläre Ausdrücke mit einem Stern an der ersten Stelle behandeln.

#### *Algorithmus*

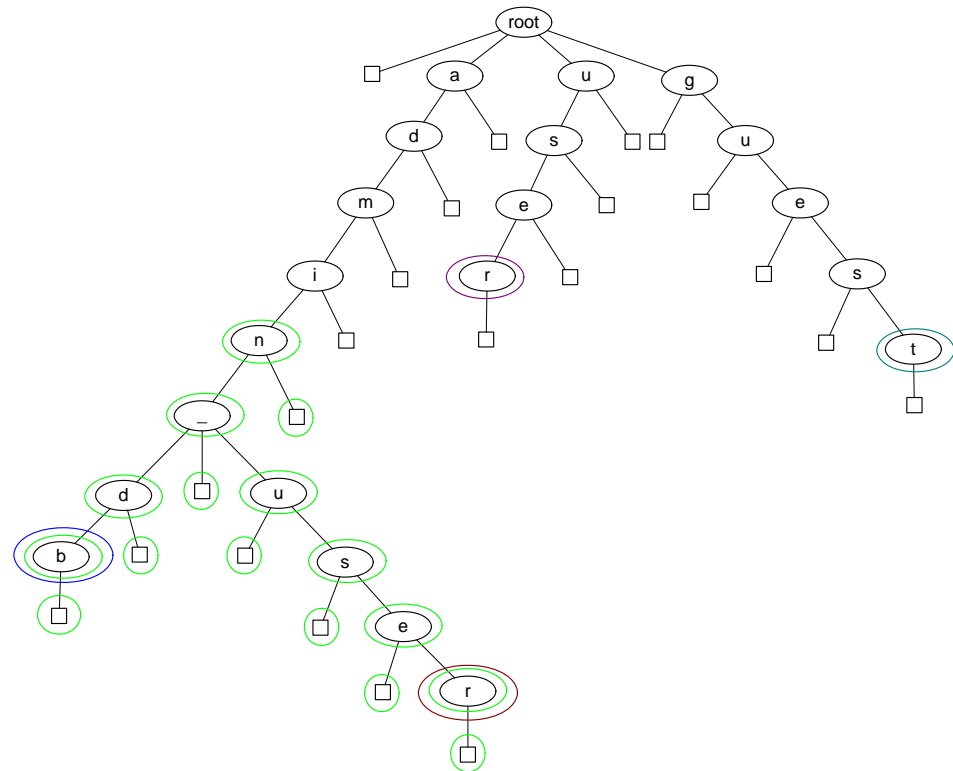
Der StringIndex ist ein Baum mit mehreren Kindelementen pro Knoten. Jeder Knoten des Baumes repräsentiert einen Charakter des zu indexierenden Wertes oder eine Wildcard. Die zusammengehängten Knoten, angefangen beim Wurzelknoten, ergeben den gesuchten Wert. Um sicher zu stellen, dass der Index immer einen Wert zurückgibt, enthält jeder Knoten einen Wildcardknoten. Dieser Knoten wird zurückgegeben, wenn der Index keinen weiteren Character des Suchstring enthält. Die nicht indexierbaren Policies werden allen Knoten des Index angehängt, damit diese immer auffindbar sind.

#### **Beispiel eines Forward StringIndex**

Der Index matched folgende Strings:

- admin.\*
- admin\_db
- admin\_user
- user
- guest

Die Knoten in welchen die Policies aufgefunden werden, sind in der Legende angegeben.



- = admin.\*
- = admin\_user
- = admin\_db
- = user
- = guest

Abbildung 15 Beispiel eines Forward String Index

Verweis

Im Detail wird der StringIndex im Developer's Guide zu dieser Arbeit [DOH07DADev] beschrieben.

### 3.4.1.1.2 ComparableIndex

Allgemein

Der ComparableIndex wird verwendet für Policies, welche im Subject-, Resource-, Action- und Environment Match Element als Matchfunktion eine less-than, less-or-equal, greater-than, greater-or-equal oder equal Funktion enthalten und deren Datentype das Interface Comparable implementiert.

AVLTree

Der Kern eines ComparableIndex bildet ein AVLTree. Mit diesem AVLTree ist der Indexierungsbaum komplett ausbalanciert und bietet Geschwindigkeitsvorteile beim Aufsuchen der Policies im laufenden Betrieb.



### Algorithmus

Der Index ist ein AVLTree mit mehreren Kindelementen pro Knoten. Jeder Attributwert des zu indexierenden Target Elements wird als Knoten im Baum aufgenommen. Der Baum balanciert sich beim Einfügen von Werten selbst aus. Subject-, Resource-, Action- und EnvironmentMatch Elemente mit einer Matchfunktion greater-than, springt im Baum zum angegebenen Wert und fügt die Policy allen Knoten, bei welchen der Wert grösser als der angegebene Wert ist, hinzu. Bei less-than die Policy wird aber allen kleineren Knoten hinzugefügt. Nicht indexierbare Policies werden allen Knoten hinzugefügt, damit diese immer auffindbar sind.

### Beispiel eines ComparableIndex

Es werden Policies mit folgenden Werten indexiert:

- >100
- <=75
- 102
- <50
- >=160

Die Abbildung 16 zeigt über welche Werte die indexierten Policies auffindbar sind.

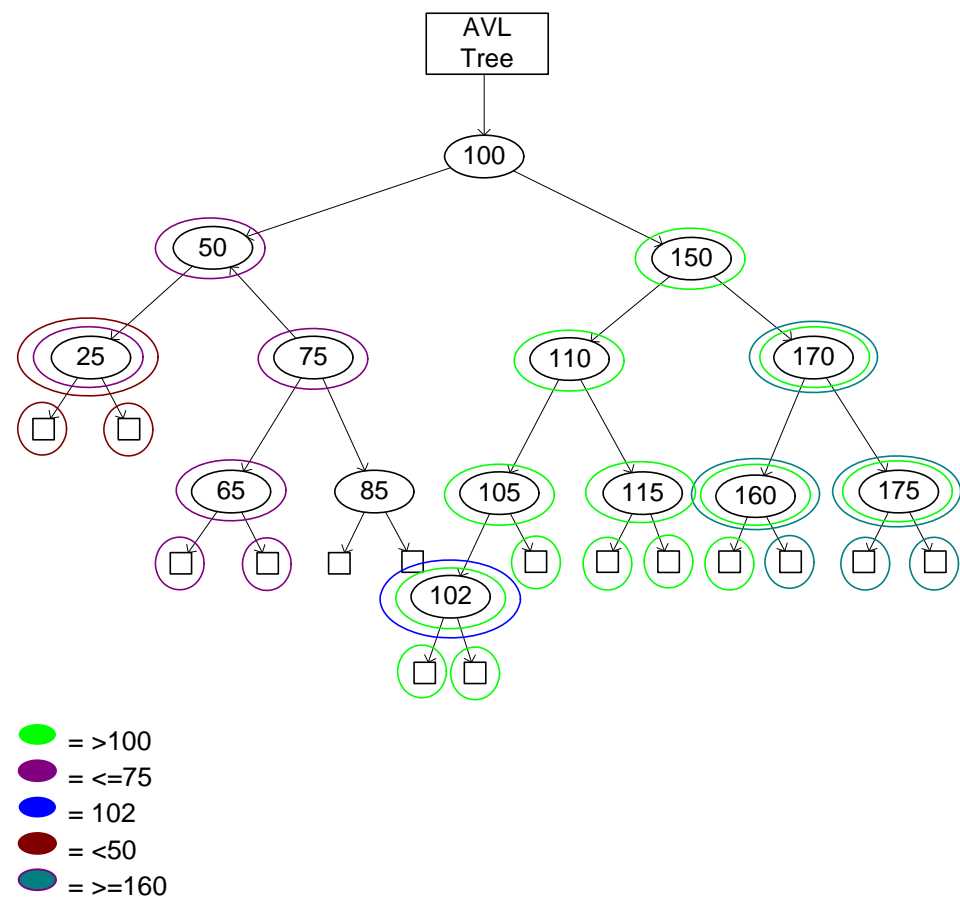


Abbildung 16 Beispiel eines Comparable Index



*Verweis* Wie ein ComparableIndex initialisiert und darauf Policies gesucht werden, sind im Developer's Guide zu dieser Arbeit [DOH07DADev] detailliert aufgeführt.

## 3.5 Modul ReferenceLoader

*Allgemein* Das Modul ReferenceLoader ist dafür zuständig um Policyreferenzen aufzulösen. Es ist in die HERAS<sup>AF</sup> XACML 2.0 Implementierung eingeplant und in den Ablauf eingebaut, wird aber momentan nur gemockt.

*Interface*

<<interface>> ReferenceLoader
+initialize(List<Evaluable>) : void +load(List<IdReferenceType> : Map<String, Evaluable> +get(String) : Evaluable

Abbildung 17 Interface ReferenceLoader

*Beschreibung*

### Interface ReferenceLoader

Das Interface ReferenceLoader abstrahiert den ReferenceLoader und bietet die Methode load() an welche eine Liste von IdReferenceType entgegennimmt um diese aufzulösen.



## 4 Tests

*Inhalt* In diesem Kapitel wird ausführlich auf das Testing der HERAS<sup>AF</sup> XACML 2.0 Implementierung eingegangen. Um die Funktionalität der HERAS<sup>AF</sup> XACML 2.0 Implementierung zu belegen, wurde auf das Testing besonderen Wert gelegt. Es wird in zwei Testbereiche gegliedert Integrations-Tests, welche Konformitäts-Tests und Last-Tests enthält und die Funktionalität der HERAS<sup>AF</sup> XACML 2.0 Implementierung belegt, sowie die Unit-Tests, welche das Funktionieren einzelner Klassen testet.

### 4.1 Integrations-Tests

*Allgemein* Um die Funktionalität der HERAS<sup>AF</sup> XACML 2.0 Implementierung zu belegen, werden zwei Arten von Tests durchgeführt. Einerseits werden Konformitäts-Tests durchgeführt, welche die pflichtmässig zu implementierenden Anforderungen der XACML 2.0 Spezifikation [XACML 2.0] abdecken und andererseits werden Last-Tests durchgeführt, welche belegen, dass die HERAS<sup>AF</sup> XACML 2.0 Implementierung Threadsafe arbeitet. Ebenfalls werden die Funktionalität einzelner Module separat durch Module Integrations-Tests abgedeckt.

#### 4.1.1 Konformitäts-Tests

*Allgemein* Um die Konformität der HERAS<sup>AF</sup> XACML 2.0 Implementierung zu belegen, bietet OASIS eine Reihe von Konformitäts-Tests [XACMLKonf] an. Diese Tests sind in einzelne Untergruppen unterteilt. Da die HERAS<sup>AF</sup> XACML 2.0 Implementierung nur die pflichtmässigen Teile der XACML 2.0 Spezifikation [XACML 2.0] implementieren, wurden auch nur diejenigen Teile getestet.

*Aufbau* Für jeden Test gibt es drei XML-Files. Eine Policy, ein Request und die darauf passende Response. Für einzelne Testfälle gibt es mehrere Policies.

*Ablauf*

1. Über den application context wird der HERAS<sup>AF</sup> PDP initialisiert und im Test geholt. Dabei werden dem HERAS<sup>AF</sup> PDP alle notwendigen Module gesetzt.
2. Die Policies werden mit dem PolicyConverter in Java Objektstrukturen konvertiert und auf dem HERAS<sup>AF</sup> PDP deployt. Dieser speichert die Policy in die Datenbank.
3. Der HERAS<sup>AF</sup> PDP wird über den application context nochmals neu initialisiert. Dabei werden die nun in der Datenbank vorhandenen Policies indiziert und ins RAM geladen.
4. Über die RequestCtxFactory wird aus dem Request.xml File ein RequestCtx erstellt.
5. Die evaluate Methode des HERAS<sup>AF</sup> PDP wird aufgerufen und der erstellte RequestCtx übergeben.
6. Der vom HERAS<sup>AF</sup> PDP zurückgegebene Response Ctx wird in ein XML-File gemarshalled und mit dem Response.xml File verglichen.



*Erfolgs  
Szenario*

Der Test gilt als erfolgreich, wenn der XML-Stream der erwarteten Antwort mit dem XML-Stream der effektiven Antwort übereinstimmt.

*Attribute  
Referenz Tests*

Die Attribute Referenz Tests testen das Referenzieren von AttributeValues eines Request durch eine Policy. Dabei werden insbesondere die verschiedenen AttributeDesignators getestet.

**Testresultat**

Bis auf zwei Ausnahmen waren alle Tests erfolgreich. Die nicht erfolgreichen Tests werden im nachfolgenden Abschnitt beschrieben.

**Abweichungen**

Folgende Tests konnten nicht erfüllt werden:

IIA002 Für den Test IIA002 muss ein fehlendes Attribute, welches im ursprünglichen Request nicht enthalten ist, von einem externen PIP zurückgegeben werden. Da der AttributeFinder in unserer HERAS<sup>AF</sup> XACML 2.0 Implementierung momentan nur gemockt wird, ist dieser Test nicht möglich.

*Target Matching  
Tests*

Die Target Matching Tests testen mehrere Variationen des Target matching.

**Testresultat**

Alle Tests waren erfolgreich.

*Funktions  
Evaluierungs  
Tests*

Die Funktions Evaluierungs Tests testen alle pflichtmässig zu implementierenden Funktionen.

**Testresultat**

Nach den im nächsten Abschnitt beschriebenen Änderungen waren alle Tests erfolgreich.



## Änderungen

In den Konformitäts Tests wurden die neusten Änderungen im Errata der XACML 2.0 Spezifikation [XACML 2.0] nicht nachgetragen. Dies betrifft vor allem die XML String Representation von DateTimeDuration und YearMonthDuration. Diese Anpassungen wurden in den folgenden Files vorgenommen:

- |                    |   |
|--------------------|---|
| IIC102 -<br>IIC107 | In den Policies wurde die XML String Representation des DataType von DaytimeDuration und YearMonthDuration angepasst.   |
| IIC150 -<br>IIC157 | In den Policies und den Requests wurde die XML String Representation des DataType von DaytimeDuration und YearMonthDuration angepasst.  |
| IIC165,<br>IIC166  | In den Policies wurden die AttributeValues der stringp-regex angepasst:<br><br><div style="margin-left: 40px;">altes AttributeValue = “*This is.* IT!”</div> <div style="margin-left: 40px;">neues AttributeValue = “.*This is.* IT!”</div> |
| IIC231,<br>IIC232  | In den Policies und den Requests wurde die XML String Representation des DataType von daytimeDuration und yearMonthduration angepasst.  |

### *Combining Algorithmen Tests*

Die Combining Algorithmen Tests testen die Funktionalität der pflichtmässig zu implementierenden Combining Algorithmen.

### **Testresultat**

Alle Tests waren erfolgreich.

### *Schema Komponenten Tests*

Die Schema Komponenten Tests testen die Elemente PolicySetIdReference und PolicyIdReference. Das Module ReferenceLoader wird momentan gemockt. Darum wurden diese Tests nicht durchgeführt.



## 4.1.2 Last-Tests

*Allgemein* Um zu belegen dass die HERAS<sup>AF</sup> XACML 2.0 Implementierung auch bei einem längeren Einsatz, keine Probleme mit der Thread-Safety oder Memory-Leaks auftreten, wurde ein Last-Test entwickelt. Mit diesem Last-Test wurden Verschiedene Szenarien mit unterschiedlicher Dauer und Anzahl Threads durchgeführt und dokumentiert.

### 4.1.2.1 Test Szenario 1

*Aufbau* Für den Test gibt es acht Policy XML-Files, und neun Request XML-Files, sowie die darauf passenden Responses. Um Abhängigkeiten von Eclipse auszuschliessen wird der Test in der Windows Konsole gestartet. Der Speicherverbrauch wird über den Windows Taskmanager angezeigt.

Testdauer: 6,5 Stunden

Anzahl Threads: 10

*Ablauf*

1. Über den application context wird der HERAS<sup>AF</sup> PDP initialisiert und im Test geholt. Dabei werden dem HERAS<sup>AF</sup> PDP alle notwendigen Module gesetzt.
2. Die acht Policies werden mit dem PolicyConverter in Java Objektstrukturen konvertiert und auf dem HERAS<sup>AF</sup> PDP deployt. Dieser speichert die Policy in die Datenbank.
3. Der HERAS<sup>AF</sup> PDP wird über den application context nochmals neu initialisiert. Dabei werden die nun in der Datenbank vorhandenen Policies indiziert und ins RAM geladen.
4. Es werden 10 Threads erstellt und gestartet.
5. Jeder Thread erstellt über die RequestCtxFactory für jedes Request.xml File ein RequestCtx und ruft die evaluate Methode des HERAS<sup>AF</sup> PDP, mit dem RequestCtx als Parameter auf.
6. Jeder, vom HERAS<sup>AF</sup> PDP zurückgegebene ResponseCtx wird in ein XML-File gemarshalled und mit dem dazugehörigen Response.xml File verglichen.

*Mögliche  
Fehlerszenarien*

#### **Falsche Entscheidung**

Der HERAS<sup>AF</sup> PDP liefert eine falsche Entscheidung, weil dieser nicht Threadsafe ist. In diesem Falle wirft der Test eine Exception und bricht ab. Der Test ist fehlgeschlagen.

#### **Memory leak**

Mit Fortlaufen des Tests erhöht sich der Speicherverbrauch des Prozesses im Taskmanager. Über eine lange Dauer führt dies zu einer OutOfMemoryException und der Test wird abgebrochen. Der Test ist fehlgeschlagen.



*Erfolgsszenario* Der Test gilt als erfolgreich, wenn der Test über die ganze Testdauer läuft.

*Ergebnis* Der Test lief erfolgreich. Der Speicherverbrauch war konstant.

#### 4.1.2.2 Test Szenario 2

*Aufbau* Für den Test gibt es acht Policy XML-Files, und neun Request XML-Files, sowie die darauf passenden Responses. Um Abhängigkeiten von Eclipse auszuschliessen wird der Test in der Windows Konsole gestartet. Der Speicherverbrauch wird über den Windows Taskmanager angezeigt.

Testdauer: 4,5 Stunden

Anzahl Threads: 50

*Ablauf*

1. Über den application context wird der HERAS<sup>AF</sup> PDP initialisiert und im Test geholt. Dabei werden dem HERAS<sup>AF</sup> PDP alle notwendigen Module gesetzt.
2. Die acht Policies werden mit dem PolicyConverter in Java Objektstrukturen konvertiert und auf dem HERAS<sup>AF</sup> PDP deployt. Dieser speichert die Policy in die Datenbank.
3. Der HERAS<sup>AF</sup> PDP wird über den application context nochmals neu initialisiert. Dabei werden die nun in der Datenbank vorhandenen Policies indexiert und ins RAM geladen.
4. Es werden 50 Threads erstellt und gestartet.
5. Jeder Thread erstellt über die RequestCtxFactory für jedes Request.xml File ein RequestCtx und ruft die evaluate Methode des HERAS<sup>AF</sup> PDP, mit dem RequestCtx als Parameter auf.
6. Jeder, vom HERAS<sup>AF</sup> PDP zurückgegebene ResponseCtx wird in ein XML-File gemarshalled und mit dem dazugehörigen Response.xml File verglichen.

*Mögliche  
Fehlerszenarien*

##### **Falsche Entscheidung**

Der HERAS<sup>AF</sup> PDP liefert eine falsche Entscheidung, weil dieser nicht Threadsafe ist. In diesem Falle wirft der Test eine Exception und bricht ab. Der Test ist fehlgeschlagen.

##### **Memory leak**

Mit Fortlaufen des Tests erhöht sich der Speicherverbrauch des Prozesses im Taskmanager. Über eine lange Dauer führt dies zu einer OutOfMemoryException und der Test wird abgebrochen. Der Test ist fehlgeschlagen.

*Erfolgsszenario* Der Test gilt als erfolgreich, wenn der Test über die ganze Testdauer läuft.

*Ergebnis* Der Test lief erfolgreich. Der Speicherverbrauch war während des Tests konstant.



### 4.1.2.3 Test Szenario 3

<i>Aufbau</i>	<p>Für den Test gibt es acht Policy XML-Files, und neun Request XML-Files, sowie die darauf passenden Responses. Um Abhängigkeiten von Eclipse auszuschliessen wird der Test in der Windows Konsole gestartet. Der Speicherverbrauch wird über den Windows Taskmanager angezeigt.</p> <p>Testdauer:                    7 Stunden</p> <p>Anzahl Threads:            100</p>
<i>Ablauf</i>	<ol style="list-style-type: none"> <li>1. Über den application context wird der HERAS<sup>AF</sup> PDP initialisiert und im Test geholt. Dabei werden dem HERAS<sup>AF</sup> PDP alle notwendigen Module gesetzt.</li> <li>2. Die acht Policies werden mit dem PolicyConverter in Java Objektstrukturen konvertiert und auf dem HERAS<sup>AF</sup> PDP deployt. Dieser speichert die Policy in die Datenbank.</li> <li>3. Der HERAS<sup>AF</sup> PDP wird über den application context nochmals neu initialisiert. Dabei werden die nun in der Datenbank vorhandenen Policies indiziert und ins RAM geladen.</li> <li>4. Es werden 100 Threads erstellt und gestartet.</li> <li>5. Jeder Thread erstellt über die RequestCtxFactory für jedes Request.xml File ein RequestCtx und ruft die evaluate Methode des HERAS<sup>AF</sup> PDP, mit dem RequestCtx als Parameter auf.</li> <li>6. Jeder, vom HERAS<sup>AF</sup> PDP zurückgegebene Response Ctx wird in ein XML-File gemarshalled und mit dem dazugehörigen Response.xml File verglichen.</li> </ol>
<i>Mögliche Fehlerszenarien</i>	<p><b>Falsche Entscheidung</b></p> <p>Der HERAS<sup>AF</sup> PDP liefert eine falsche Entscheidung, weil dieser nicht Threadsafe ist. In diesem Falle wirft der Test eine Exception und bricht ab. Der Test ist fehlgeschlagen.</p> <p><b>Memory leak</b></p> <p>Mit Fortlaufen des Tests erhöht sich der Speicherverbrauch des Prozesses im Taskmanager. Über eine lange Dauer führt dies zu einer OutOfMemory Exception und der Test wird abgebrochen. Der Test ist fehlgeschlagen.</p>
<i>Erfolgsszenario</i>	<p>Der Test gilt als erfolgreich, wenn der Test über die ganze Testdauer läuft.</p>
<i>Ergebnis</i>	<p>Der Test lief erfolgreich. Der Speicherverbrauch war während des Tests konstant.</p>



#### 4.1.2.4 Test Szenario 4

<i>Aufbau</i>	<p>Für den Test gibt es acht Policy XML-Files, und neun Request XML-Files, sowie die darauf passenden Responses. Um Abhängigkeiten von Eclipse auszuschliessen wird der Test in der Windows Konsole gestartet. Der Speicherverbrauch wird über den Windows Taskmanager angezeigt.</p> <p>Testdauer: 28.5 Stunden</p> <p>Anzahl Threads: 50</p>
<i>Ablauf</i>	<ol style="list-style-type: none"> <li>1. Über den application context wird der HERAS<sup>AF</sup> PDP initialisiert und im Test geholt. Dabei werden dem HERAS<sup>AF</sup> PDP alle notwendigen Module gesetzt.</li> <li>2. Die acht Policies werden mit dem PolicyConverter in Java Objektstrukturen konvertiert und auf dem HERAS<sup>AF</sup> PDP deployt. Dieser speichert die Policy in die Datenbank.</li> <li>3. Der HERAS<sup>AF</sup> PDP wird über den application context nochmals neu initialisiert. Dabei werden die nun in der Datenbank vorhandenen Policies indiziert und ins RAM geladen.</li> <li>4. Es werden 50 Threads erstellt und gestartet.</li> <li>5. Jeder Thread erstellt über die RequestCtxFactory für jedes Request.xml File ein RequestCtx und ruft die evaluate Methode des HERAS<sup>AF</sup> PDP, mit dem RequestCtx als Parameter auf.</li> <li>6. Jeder, vom HERAS<sup>AF</sup> PDP zurückgegebene ResponseCtx wird in ein XML-File gemarshalled und mit dem dazugehörigen Response.xml File verglichen.</li> </ol>
<i>Mögliche Fehlerszenarien</i>	<p><b>Falsche Entscheidung</b></p> <p>Der HERAS<sup>AF</sup> PDP liefert eine falsche Entscheidung, weil dieser nicht Threadsafe ist. In diesem Falle wirft der Test eine Exception und bricht ab. Der Test ist fehlgeschlagen.</p> <p><b>Memory leak</b></p> <p>Mit Fortlaufen des Tests erhöht sich der Speicherverbrauch des Prozesses im Taskmanager. Über eine lange Dauer führt dies zu einer OutOfMemoryException und der Test wird abgebrochen. Der Test ist fehlgeschlagen.</p>
<i>Erfolgsszenario</i>	<p>Der Test gilt als erfolgreich, wenn der Test über die ganze Testdauer läuft.</p>
<i>Ergebnis</i>	<p>Der Test lief erfolgreich. Der Speicherverbrauch war während des Tests konstant.</p>



### 4.1.3 Modul Integration-Tests

<i>Allgemein</i>	Um die Funktionsfähigkeit der einzelnen Module zu belegen, wurden pro Modul Integration-Tests entwickelt. Diese Tests wurden für die Module Core, Locator und Persistence realisiert. Das Modul PDP steuert im eigentlichen Sinne nur das Zusammenspiel der einzelnen Module und wird deshalb nicht mit einem Modul Integration-Test abgedeckt. Ausserdem wird die Integration des PDP's schon zur Genüge mit den Konformitäts-Tests und den Last-Tests belegt.
<i>Core</i>	Die Core Integration-Tests umfassen das Marshalling und Unmarshalling von PolicySetType, PolicyType, RequestType und ResponseType, sowie die Funktionalität des TargetMatcher.
<i>Locator</i>	Die Locator Integration-Tests umfassen den kompletten Aufbau des Locators, sowie das Auffinden von anwendbaren Policies über die initialisierten Indexe.
<i>Persistence</i>	Die Persistence Integration-Tests umfassen das Speichern, Löschen und Laden von Policies auf der Datenbank.

## 4.2 Unit-Tests

<i>Allgemein</i>	Um die Funktionalität einzelner Klassen der HERAS <sup>AF</sup> XACML 2.0 Implementierung zu belegen werden mit dem TestNG Framework Unit-Tests durchgeführt.
<i>Aufbau der Tests</i>	Um die einzelnen Klassen separat von andern zu testen, wurden Referenzen auf andere Klassen, wenn notwendig gemockt. Um gleiche Tests mit verschiedenen Werten durchführen zu können, wurde das Feature Dataprovider von TestNG genutzt.
<i>Tests</i>	Es wurden zwei Arten von Tests durchgeführt: <ul style="list-style-type: none"> <li>• Tests, welche den Erfolgsfall prüfen</li> <li>• Tests, welche den Fehlerfall simulieren</li> </ul> Jeder der Testarten wurde mit verschiedenen Werten durchgeführt.
<i>Code Abdeckung</i>	Um die Code Abdeckung der zu prüfenden Klassen zu ermitteln, wurde das Eclipse Plug-in EclEmma verwendet.



## 5 Schlussbericht

### 5.1 Zusammenfassung

Als Vorarbeit auf die Diplomarbeit arbeiteten wir uns nochmals in die XACML 2.0 Spezifikation [XACML 2.0] ein. Diese Spezifikation ist die Grundlage der zu erstellenden HERAS<sup>AF</sup> XACML 2.0 Implementierung.

In der ersten, zwei Wochen andauernden Entwicklungsphase befassten wir uns mit dem Design der Implementierung, mit den Möglichkeiten von JAXB 2.1 um die Datenstruktur des Core's zu erstellen und zu verwenden, der Persistierung der deployten Policies, sowie dem Locator Modul für das Indexieren und Auffinden der Policies.

Die Erstellung des Overall Design verlief ohne Probleme. Dieses wurde Modular aufgebaut um die Erweiterbarkeit, sowie die Änderbarkeit möglichst gut zu unterstützen.

Die Analyse von JAXB 2.1 gestaltete sich etwas schwieriger. Die Grundfunktionalitäten von JAXB konnten ziemlich schnell umgesetzt werden, aber mit den von uns definierten Anforderungen stiessen wir schnell an den Grenzen von JAXB an. Die Möglichkeit die Datenstruktur von XACML 2.0 zu generieren und daran keine Anpassungen vornehmen zu müssen, musste verworfen werden. Dies hatte zu Folge, dass das Design nochmals ein wenig anzupassen war.

Weit weniger Probleme bereitete die Abstrahierung der Datenhaltung. Einzige Anforderung an die Datenhaltung war die Performanz, weswegen entschieden wurde, die Daten nicht in ein Schema, sondern serialisiert zu persistieren.

Die weitaus anspruchsvollste Arbeit in der ersten Phase war das Entwickeln der Indexierungsalgorithmen im Locator Modul. Für verschiedene Datentypen und Matchfunktionen musste es möglich sein, die Policies geeignet zu indexieren. Aufgrund dessen wurden zwei Indexierungsalgorithmen entwickelt. Der eine kann den Datentyp String, sowie Gleichheitsfunktionen und reguläre Ausdrücke indexieren und der andere kann diejenigen Datentypen indexieren, welche mit kleiner, grösser usw. verglichen werden. Um die entwickelten Indexierungsalgorithmen unternehmenstauglich zu gestalten, wird angeboten, dass die zu indexierenden Werte konfigurierbar sind.

Nach der ersten Entwicklungsphase galt es in einer einwöchigen Dokumentationsphase die geleistete Arbeit zu dokumentieren. Zu diesem Zeitpunkt konnte der Zeitplan noch mühelos eingehalten werden.

In der zweiten Entwicklungsphase, welche ebenfalls zwei Wochen dauerte, ging es darum, die Entscheidungsfindung des PDP's zu implementieren, sowie mit Conformance-tests die Funktionalität der HERAS<sup>AF</sup> XACML 2.0 Implementierung zu belegen.

Die grössten Probleme beim Implementieren der Entscheidungsfindung waren die teilweise unterschiedlich aufgefassten Beschreibungen der XACML 2.0 Spezifikation. Des Weiteren wurden in dieser Phase die über zweihundert Funktionen, sowie die Combining Algorithmen, welche nach XACML 2.0 Spezifikation angeboten werden müssen, implementiert.

Um die Konformität HERAS<sup>AF</sup> XACML 2.0 Implementierung zu belegen, bietet OASIS über 300 Test-Requests, -Responses und -Policies an. Diese wurden mit Erfolg getestet. Damit belegen wir, dass unsere Implementierung funktionsfähig und standardkonform ist.

Danach galt es in einer zweiwöchigen Dokumentationsphase die in der zweiten Entwicklungsphase geleistete Arbeit zu dokumentieren, sowie die gesamte Dokumentation der Arbeit fertig zu stellen.

Die letzte Woche wurde in der Planung freigehalten um der Dokumentation den letzten Feinschliff zu geben, diese zu binden, das Plakat für die HSR zu erstellen, sowie als Reserve für Unvorhergesehenes.

Da wir im Ungewissen waren, wie viel Zeit die Implementierung der XACML 2.0 Spezifikation effektiv brauchen wird, haben wir von Anfang an 10 bis 11 Stunden pro Tag gearbeitet. Dank dieser Massnahme, ist die Zeitplanung ziemlich gut aufgegangen. Damit war es uns sogar möglich, die Wochenendarbeit auf ein Minimum zu beschränken.



## 5.2 Umfang

Im Rahmen der Diplomarbeit wurde eine lauffähige und zur XACML 2.0 Spezifikation [XACML 2.0] konforme XACML 2.0 Implementierung entwickelt. Die Architektur der HERAS<sup>AF</sup> XACML 2.0 Implementierung ist modular aufgebaut. Jedes der Module kann zu einem späteren Zeitpunkt ausgetauscht werden. Ebenfalls ist die ganze HERAS<sup>AF</sup> XACML 2.0 Implementierung über Spring konfigurierbar.

Die HERAS<sup>AF</sup> XACML 2.0 Implementierung beinhaltet alle von der XACML 2.0 Spezifikation pflichtmässig vorgegebenen, sowie einige der optionalen Elemente.

Die Datenstruktur des Core's wurde aus den Schema-Files der XACML 2.0 Spezifikation mit JAXB 2.1 generiert. Das Parsen von XACML 2.0 Request, -Response und -Policies wird mit JAXB 2.1 realisiert. Die Datenstruktur enthält für eine spätere Verwendung auch die optionalen Elemente. Für die Entscheidungsfindung werden aber die meisten optionalen Elemente nicht berücksichtigt.

Speziellen Wert wurde dem Locator Modul beigemessen. Um eine möglichst schnelle Auffindung von potentiell möglichen Richtlinien zu realisieren wurden Indexierungsalgorithmen entwickelt. Um den Locator für die Bedürfnisse einzelner Unternehmen anpassbar zu gestalten, können die indexierbaren Werte, sowie die zu verwendenden Indexierungsalgorithmen konfiguriert werden.

Das nicht geforderte Modul zur Auflösung von Policyreferenzen und der AttributeFinder zur auflösung von fehlenden Attributen sind in die Entscheidungsfindung des HERAS<sup>AF</sup> PDP eingebunden, werden aber gemockt.

Um die Funktionalität einer XACML 2.0 Implementierung zu belegen, stellt OASIS Konformitätstests zur Verfügung, welche erfüllt werden müssen. Diese Tests werden von der HERAS<sup>AF</sup> XACML 2.0 Implementierung abgedeckt. Ebenfalls wurden Integrations-Tests auf Ebene der einzelnen Module, sowie Unit-Tests entwickelt, welche die Funktionalität einzelner Module und Klassen gewährleisten.

## 5.3 Abweichungen zur XACML 2.0 Spezifikation

<i>Allgemein</i>	Die HERAS <sup>AF</sup> XACML 2.0 Implementierung weicht in ein paar wenigen Punkten von der XACML 2.0 Spezifikation [XACML 2.0] ab. In diesem Abschnitt werden diese Abweichungen aufgelistet und begründet.
<i>Unique ID über Policy und PolicySet</i>	<p>Laut XACML 2.0 Spezifikation soll jede Policy unter allen Policies, sowie jedes PolicySet unter allen Policy Sets, eine einzigartige ID haben.</p> <p>Da Policy und Policy Set sich in ihrem Verhalten nicht wesentlich unterscheiden, implementieren in der HERAS<sup>AF</sup> XACML 2.0 Implementierung beide das Interface Evaluatable. So kann die Speicherung der einzelnen Policies und PolicySets zusammengefasst werden. Dies zieht aber die Einschränkung mit sich, dass eine ID über alle Evaluatables eindeutig sein muss.</p>
<i>Zeitzone Funktionen</i>	<p>Die Funktionen welche eine Zeitzone beinhalten, können in Grenzfällen falsche Ergebnisse liefern, die Zeit könnte so z.B. um eine Stunde verschoben sein.</p> <p>Die Zeitzone werden in der HERAS<sup>AF</sup> XACML 2.0 Implementierung ignoriert. Um die Zeitzone zu unterstützen, müsste man sich ausführlicher mit der Zeitzone-Problematik befassen. Dies konnte aus zeitlichen Gründen nicht mehr erledigt werden.</p>



*Optionales Element bei Generierung*

Für die Generierung der Datenstruktur mit JAXB 2.1 wurde im <xs:complexType name="PolicyType" > im Sequence Block das optionale Element CombiningParameters auskommentiert. Dies war notwendig, weil im Nachfolgenden Choice Block das Element CombiningParameters nochmals vorkommt und JAXB 2.1 damit nicht umgehen kann.

## 5.4 Ausblick

*Allgemein*

In diesem Abschnitt wird ein Ausblick auf mögliche Erweiterungen und Änderungen der HERAS<sup>AF</sup> XACML 2.0 Implementierung gegeben.

*Optionale Elemente in der Entscheidungsfindung*

Die optionalen Elemente werden in der bestehenden Datenstruktur der HERAS<sup>AF</sup> XACML 2.0 Implementierung angeboten. Diese müssten aber in der Entscheidungsfindung noch berücksichtigt werden. Die Liste der in der HERAS<sup>AF</sup> XACML 2.0 Implementierung nicht implementierten optionalen Elemente wird im folgenden abgebildet:

- Obligations (Context)
- StatusMessage (Context)
- Obligations (Policy)
- Obligation
- AttributeAssignment
- AttributeSelector
- CombinerParameters
- CombinerParameter
- PolicyCombinerParameters
- PolicyDefaults
- PolicySetDefaults
- RuleCombinerParameters
- XPathVersion

*Optionale Funktionen*

Die optional zu implementierenden Funktionen enthalten XPath Funktionalitäten. Diese Funktionen müssten nach XACML 2.0 Spezifikation entwickelt werden.

*Mocks ersetzen*

Das in der HERAS<sup>AF</sup> XACML 2.0 Implementierung gemockte Modul ReferenceLoader kann man zu einem späteren Zeitpunkt durch seine Implementierung ersetzen. Ebenfalls müsste entschieden werden, ob der PIP intern im PDP oder extern angesprochen würde. Bei einer allfälligen Implementierung des ReferenceLoaders müsste die Kommunikation mit einem externen PDP entwickelt werden.

*Verbesserte Indexierungsalgorithmen*

Die Indexierung einzelner Policies könnte man noch viel umfangreicher und komplexer lösen, als dies die HERAS<sup>AF</sup> XACML 2.0 Implementierung macht. So wurden in dieser Implementierung nur Targets der Toplevel Policies Indexiert. Ebenfalls könnte man für eine Indexierung auch die Targets der einzelnen Rules berücksichtigen. Die Tiefgründigkeit einer solchen Indexierung würde aber rasant ansteigen. Die Entwicklung neuer Indexierungsalgorithmen könnte man in einer weiteren Arbeit abdecken.



# Anhang I : Projektmanagement



# 1 Projektplanung

## 1.1 Einführung

*Zweck* Die Projektplanung soll als Leitfaden für das Projekt "HERAS<sup>AF</sup>: XACML 2.0 Implementierung" 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

#### Florian Huonder

Architekt

#### Verantwortung

- Kontrolle der Module
- Kontrolle der technischen Analyse
- Kontrolle des Designs

#### Stefan Oberholzer

Projektleiter

#### Verantwortung

- Kontrolle Projektfortschritt
- Sicherung der Kommunikation und Korrespondenz

#### Sacha Dolski

Chefentwickler

#### Verantwortung

- Kontrolle des Codes
- Koordination bei Refactoring
- Sicherung der Infrastruktur

### Externe Schnittstellen

#### Wolfgang Giersche

Verantwortlicher Dozent

#### Kontakt

Email: [wgiersch@hsr.ch](mailto:wgiersch@hsr.ch)  
Tel: +41 (0)79 769 98 12

Verantwortlicher Dozent der Projektarbeit und Ansprechperson für allgemeine Architekturfragen die das HERAS<sup>AF</sup> als Ganzes betreffen.

#### René Eggenschwiler

Projektbetreuer

#### Kontakt

Email: [reggensc@hsr.ch](mailto:reggensc@hsr.ch)  
[rene.eggenschwiler@hispeed.ch](mailto:rene.eggenschwiler@hispeed.ch)  
Tel: +41 (0)78 629 08 09

Betreut das Projektteam, Anlaufstelle bei Unklarheiten oder Problemen.

*Termine*

Folgende Termine sind von der HSR vorgegeben:

- Projektstart 02.10.2007
- Abgabe 23.11.2007

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

Nach jeder Iteration wird dem verantwortlichen Dozenten 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>
Implementierung der Module Locator, Persistence, und die Datenklassen des Core Moduls.	01.10.2007	14.10.2007
<b>Iteration 2</b>	<b>Start</b>	<b>Ende</b>
Dokumentation der Iteration 1.	15.10.2007	21.10.2007
<b>Iteration 3</b>	<b>Start</b>	<b>Ende</b>
Implementierung der Entscheidungsfindung.	22.10.2007	04.11.2007
<b>Iteration 4</b>	<b>Start</b>	<b>Ende</b>
Dokumentation der Iteration 3.	05.11.2007	18.11.2007
<b>Iteration 5</b>	<b>Start</b>	<b>Ende</b>
Abschlussarbeiten und Reserve.	19.11.2007	23.11.2007

*Meilensteine*

<b>MS1: Module zum Laden und Persistieren von Policies</b>	<b>Datum</b>
Die Module Locator, Persistence und die Datenstruktur im Core sind implementiert, getestet und dokumentiert.	21.10.2007
<b>MS2: Modul zur Entscheidungsfindung</b>	<b>Datum</b>
Die Entscheidungsfindung, sowie die Kombinationsalgorithmen, die Funktionen und die handle Methoden der ExpressionTypes sind implementiert, dokumentiert und getestet sowie mit den Artefakten von MS1 zusammengeführt.	18.11.2007

**MS3: Abschluss der Diplomarbeit****Datum**

Die Diplomarbeit ist komplett dokumentiert und liegt in gebundener Form vor.

23.11.2007

**1.4 Risikomanagement**

Risiko	Auswirkungen	Massnahmen	Kosten der Massnahmen in Personenstunden	Maximaler Schaden in Personenstunden	Wahrscheinlichkeit des Eintreffens in Prozent	Gewichteter Schaden in Personenstunden	Priorität
R01: Absturz oder Ausfall von Teilen der Infrastruktur	Daten nicht mehr verfügbar	Regelmässiges Backup der Daten	1	48	1	0.5	6
R02: Teammitglied wird krank oder fällt verletzt für eine gewisse Zeit aus	Teammitglieder müssen Arbeiten des Mitglieds übernehmen und betreiben deshalb einen Mehraufwand	Täglich kurzes Meeting. Wer macht was? Wie weit? Usw.	12	360	1	3.5	7
R03: Integration der Entscheidungsfindung mit den anderen Modulen schlägt fehl	Fehlersuche und weiterer Programmieraufwand	Sauberes Design und Codereviews	15	90	10	9	3
R04: JAXB Komplexität unterschätzt	Geplantes ist nicht umsetzbar. Eine andere Lösung muss umgesetzt werden	Alternativlösungen aufzeigen, frühe Analyse von JAXB	9	30	10	3	5
R05: Komplexität eines Modules unterschätzt	Erheblicher Mehraufwand für die Umsetzung des Moduls	Sauberes Design der einzelnen Module, sorgfältiges Studium der Spezifikationen	30	180	15	27	1
R06: Drucker, Laptop fällt aus	Laptop, Drucker für eine gewisse Zeit nicht verfügbar	Abklärungen treffen, ob es Ersatzgeräte gibt	3	50	2	1	8
R07: Dokumentationsaufwand für JavaDoc unterschätzt	Erheblicher Mehraufwand um Javadoc nach zu dokumentieren	Genügend Zeit für die Dokumentation einplanen. Saubere Zeitplanung	3	30	5	1.5	2
R08: Indexierungsalgorithmen schwieriger als erwartet	Erheblicher Mehraufwand um eine Indexierung zu Stande zu bringen	Sauberes Design, Pair Programming	30	100	15	15	4
Total Stunden in Arbeitspaketen enthalten			103				
Total Rückstellungen						60.5	



## 1.5 Qualitätsmassnahmen

- Dokumentation* Die Dokumentation des ganzen Projekts, vom Projektantrag bis zur Code-Dokumentation, hat einen hohen Stellenwert und wird fortlaufend aktualisiert. Das Hauptdokument wird in Deutsch, der Developer's Guide in Englisch geschrieben.
- Das JavaDoc soll ausführlich sein und auf die XACML 2.0 Spezifikation [XACML2.0] verweisen.
- Tests* Die Tests sollen eine komplette Feature- bzw. Modulabdeckung gewährleisten. Mit Hilfe von Mock-Objekten sollen die wichtigen Features getrennt getestet werden können.
- Code* Der Code soll Test-Driven entwickelt werden. Für sämtliche Features und Module wird deshalb zuerst ein Test geschrieben.
- Reviews* Um die Qualität zu steigern, werden sämtliche Dokumente und alle Klassen im Team einem Review unterzogen. Diese Reviews werden in einem Sitzungsprotokoll festgehalten.

## 1.6 Verwendete Technologie

- Spring 2.0.6* Mit Spring wird die Austauschbarkeit der Komponenten in einem Softwaresystem sichergestellt.
- Den Objekten werden die abhängigen Objekte bzw. Ressourcen zugewiesen. Sie müssen sie nicht selbst suchen.
- JAXB 2.1* **J**ava **A**rchitecture for **X**ML **B**inding ist eine API, die es ermöglicht, Daten aus einem XML-Schema automatisiert an annotierte Java Klassen zu binden. JAXB bietet Funktionen für ein Marshalling / Unmarshalling zwischen XML Daten und Java Klassen.
- Dadurch wird es dem Programmierer ermöglicht mit XML-Daten zu arbeiten, ohne auf konkrete Parsing Technologien angewiesen zu sein.
- MySQL 5.0* MySQL ist eine frei verfügbare Datenbank.
- Java 1.5* Objektorientierte Programmiersprache.
- Maven 2.0.7* Maven 2 ist eine Weiterentwicklung des Apache-Ant-Projekts, welches die Erzeugung und Verteilung von Java-Programmen managen soll. Es basiert auf einer Plugin-Architektur, die es ermöglicht, Plugins für verschiedenste Anwendungen auf das Projekt anzuwenden, ohne diese explizit installieren zu müssen.



## 1.7 Verwendete Standards

### *XACML 2.0*

Die eXtensible **A**ccess **C**ontrol **M**arkup **L**anguage ist ein XML basierter Standard, der die Darstellung und Verarbeitung von Autorisierungs-Policies standardisiert.

Es dient dazu, Regeln aufzustellen, durch deren Auswertung der Zugriff von Subjekten auf Ressourcen eines Systems gesteuert werden kann.

## 1.8 Schlussbericht Projektmanagement

### *Planung*

Die vorgegebenen Ziele wurden alle erreicht. Zu Beginn der Arbeit wurde bewusst mehr gearbeitet als eingeplant, um Nacharbeiten in der Schlussphase zu vermeiden. Dabei ist ein beträchtlicher Mehraufwand entstanden.

### *Aufwand- schätzung*

Der Unterschied zwischen geplantem und geleistetem Aufwand ist in etwa eine Woche Mehraufwand pro Person. Die Diskrepanz entstand dadurch, dass von der ersten Woche an zuviel gearbeitet wurde und dies bis zur letzten durchgezogen wurde. Dies wurde gemacht, weil für die Arbeit sehr hohe Ziele gesteckt wurden und diese zeitlich schwer einschätzbar waren.

Total wurden 1080 Stunden geplant. Tatsächlich benötigt wurden 1264 Stunden.



## 2 Auswertung

### 2.1 Architektur und Code Qualität

*Inhalt* In der Auswertung der Architektur und des Codes nach Metriken werden die Module mit produktivem Code, sowie die Module für die Testabdeckung berücksichtigt.

<i>Core Modul</i>	<b>Metriken</b>	
	Anzahl Packages	53
	Anzahl Interfaces	12
	Anzahl Klassen	432
	Anzahl Methoden	1'416
	Anzahl Codezeilen produktiv	10'476
	Anzahl Codezeilen test	10'260
<i>PDP Modul</i>	<b>Metriken</b>	
	Anzahl Packages	5
	Anzahl Interfaces	2
	Anzahl Klassen	13
	Anzahl Methoden	38
	Anzahl Codezeilen produktiv	646
	Anzahl Codezeilen test	0
<i>Locator Modul</i>	<b>Metriken</b>	
	Anzahl Packages	10
	Anzahl Interfaces	4
	Anzahl Klassen	28
	Anzahl Methoden	280
	Anzahl Codezeilen produktiv	3'145
	Anzahl Codezeilen test	1'497
<i>Persistence Modul</i>	<b>Metriken</b>	
	Anzahl Packages	2
	Anzahl Interfaces	1
	Anzahl Klassen	4
	Anzahl Methoden	28
	Anzahl Codezeilen produktiv	227
	Anzahl Codezeilen test	0



*Conformance  
Tests Modul*

**Metriken**

---

Anzahl Klassen	4
Anzahl Methoden	30
Anzahl Codezeilen test	2'626

*Integration  
Tests Modul*

**Metriken**

---

Anzahl Klassen	18
Anzahl Methoden	167
Anzahl Codezeilen test	1'701

*Stress Test  
Modul*

**Metriken**

---

Anzahl Klassen	8
Anzahl Methoden	50
Anzahl Codezeilen test	726

*Projekt-  
übergreifend*

**Metriken**

---

Anzahl Module	10
Anzahl Packages	70 (Ohne Testmodule)
Anzahl Interfaces	19 (Ohne Testmodule)
Anzahl Klassen	507
Anzahl Methoden	2'009
Anzahl Codezeilen produktiv	14'494
Anzahl Codezeilen test	16'810
Anzahl Codezeilen gesamt	31'304



## 2.2 Zeitauswertung

*Allgemein* Das Projekt lief über eine Dauer von acht Wochen Vollzeit.

*Überblick*

	Geplant	Effektive gearbeitet
Personenstunden:	1080	1264
Stunden pro Woche (Durchschnitt):	45	52.5

*Zeit pro Phase*

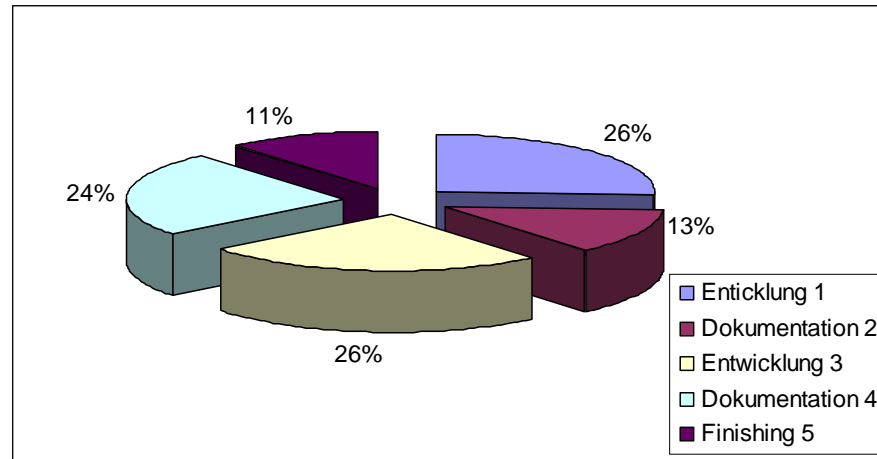


Abbildung 18 Verteilung der verschiedenen Phasen im Projekt

*Zeit pro Teammitglied*

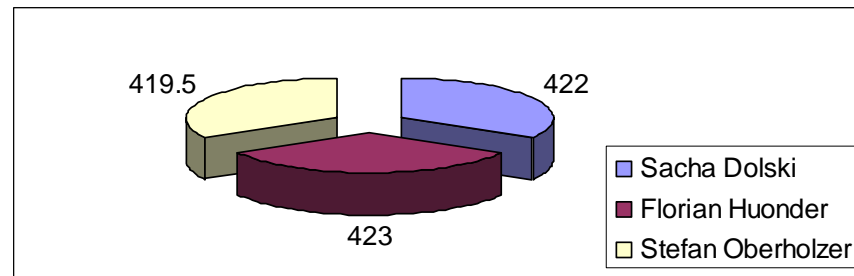


Abbildung 19 Verteilung der gearbeiteten Stunden pro Projektmitglied



## Anhang II : Allgemein



# 1 Persönliche Berichte

## 1.1 Sacha Dolski

Der Krönende Abschluss eines Studiums bedeutet eine Diplomarbeit zu schreiben. Nach der guten Zusammenarbeit in den ersten zwei Studienarbeiten entschied sich Florian Huonder, Stefan Oberholzer und ich dafür diese ebenfalls als Dreiergespann in Angriff zu nehmen.

Im Laufe der zweiten Studienarbeit kristallisierte sich heraus, dass die zu verwendende XACML 2.0 Implementierung von Sun über einige Fehler aufweist. Aufgrund dessen kam die Idee die XACML 2.0 Spezifikation komplett neu zu implementieren.

Die Implementierung dieser Spezifikation wurde von Dozent, Betreuer und von uns als sehr grosse Herausforderung angesehen. Abschätzungen ob der Umfang der Arbeit im Rahmen einer Diplomarbeit überhaupt durchführbar ist waren sehr schwierig zu treffen. Mit dem Vertrauen in die eigenen Fähigkeiten und der Tatsache, dass wir ein eingespieltes Team sind stellten wir uns dennoch dieser Herausforderung.

Meines Erachtens die grösste Herausforderung der Arbeit war das Entwickeln der Indexierungsalgorithmen. Diese erforderten viel Kaffee und teilweise rauchende Köpfe. Probleme traten im eigentlichen Sinne nicht auf. Einzig durch ein spätes Refactoring im Code, wegen eines zuerst fehlerhaft interpretierten Teils der Spezifikation, gerieten wir nochmals ein bisschen ins Schwimmen.

Den Verlauf des Projektes, sowie das entstandene Endprodukt werde ich als grossen Erfolg. Mit der, von Anfang an, eingeplanten Überzeit konnten wir den Zeitplan auch ziemlich gut einhalten. Sogar Wochenendarbeiten konnten fast gänzlich vermieden werden.

Insgesamt gesehen, war diese Arbeit wiederum eine sehr wichtige Erfahrungsquelle. Insbesondere in den Bereichen Softwareentwicklung und Teamarbeit konnte ich deutlich dazulernen.

Abschliessend danke ich dem verantwortlichen Dozent Wolfgang Giersche, sowie dem Betreuer René Eggenschwiler für die Möglichkeit dieser Arbeit und deren Unterstützung. Ebenfalls danke ich meinem Umfeld für die geleistete Arbeit als mentale Motivatoren und Gegenleser.

## 1.2 Florian Huonder

Die Diplomarbeit ist der lang ersehnte Abschluss meines Informatik-Studiums. Darum bin ich auch froh, dass sie ein voller Erfolg war. Es war nicht eine standard Diplomarbeit, wo es darum ging eine Applikation zu designen und anschliessend zu implementieren. Wir implementierten den XACML 2.0 Standard, was ein ganzes Stück mehr Leistung von uns forderte, da wir gezwungen waren, jedes Detail des Standard penibel einzuhalten. Zudem war das Abschätzen des Zeitaufwands im Voraus nur sehr schwer möglich, was offen liess, ob die Implementierung im Rahmen einer Diplomarbeit überhaupt möglich ist.

Da das Produkt dieser Arbeit den Kern des HERAS<sup>AF</sup>-Projektes darstellt, war es für uns ein grosser Ansporn eine top Applikation zu implementieren und die Dokumentation auf einem hohen Niveau zu haben.

Wir haben uns vertieft mit Technologien wie JAXB und dem Spring Framework auseinander gesetzt und sind teilweise auch an deren Grenzen gestossen. Diese Erfahrung war für mich persönlich sehr wertvoll.

Ich war hoch erfreut, dass es problemlos möglich war, wieder mit Sacha Dolski und Stefan Oberholzer zusammenzuarbeiten. Unser Team hatte in den vergangenen Studienarbeiten gezeigt, dass wir uns hervorragend ergänzen und somit auch auf professionellem Level arbeiten können.

Rückblickend auf das ganze Studium an der HSR gesehen habe ich einen riesen Schritt vorwärts gemacht. In den letzten drei Jahren habe ich das ganze Software Engineering Handwerk erlernt und vertieft. Ich fühle mich gerüstet für den Einstieg ins Berufsleben nach der Diplomarbeit.

Hier möchte ich auch noch die Chance ergreifen, allen zu danken, welche mich während meines Studiums finanziell und auch persönlich unterstützt haben.

Besonderen Dank geht Wolfgang Giersche und René Eggenschwiler für die grosse Unterstützung während den Studien- und Diplomarbeiten.



### 1.3 Stefan Oberholzer

Die Note der Diplomarbeit ist in meinen Augen die wichtigste Note des ganzen Studiums. Aus diesem Grund war es für mich wichtig diese mit Leuten zu schreiben von denen ich weiss, dass sie dies ebenso sehen. Deshalb war ich froh als ich die Gelegenheit hatte, die Diplomarbeit zusammen mit Florian Huonder und Sacha Dolski zu schreiben. In den Studienarbeiten ergänzten wir uns hervorragend, deshalb war ich zuversichtlich, dass auch diese Arbeit zu einem guten Resultat führen wird.

Zu Beginn der Arbeit waren wir sehr unsicher, ob wir die gesamte Spezifikation implementieren können. Deshalb begann unsere Arbeit damit, zu ermitteln welche Teile der Arbeit zwingend implementiert werden müssen und welche auch nachträglich noch implementiert werden können. Als nächsten Schritt, erstellen wir eine grobe Architektur sowie den Zeitplan. Dabei kamen wir zum Schluss, dass es möglich ist die notwendigen Teile zu Implementieren oder durch klar definierte Mock Klassen für eine spätere Implementierung vorzubereiten. Dadurch, dass wir jeden Tag über 10 Stunden arbeiteten, konnten wir diesen Zeitplan einhalten.

Der grösste Knackpunkt der Implementierung war der Index, welcher einen grossen Teil der ersten Iteration verschlang. Eine zweite Knacknuss war das Implementieren des Evaluationsprozesses. Für diesen konnten wir jedoch eine gute Lösung erarbeiten, die das Implementieren sehr vereinfachte. Abschliessend bin ich mit unserer Arbeit sehr zufrieden, da ich während der Arbeit viel lernte und ich denke, dass ich besonders während den Studienarbeiten und dieser Diplomarbeit grosse Fortschritte erzielen konnte.

Aus diesem Grund möchte ich auch René Eggenschwiler und Wolfgang Giersche für die Unterstützung während den Arbeiten danken.



## 2 Glossar

<i>AVLTree</i>	Adel'son-Vel'skii and Landis Tree
<i>BLOB</i>	Binary Large Object
<i>HERAS<sup>AF</sup></i>	Holistic Enterprise Ready Application Security <small>Architecture Framework</small>
<i>JAXB</i>	Java API for XML Bindings
<i>JDBC</i>	Java Database Connectivity
<i>LDAP</i>	Lightweight Directory Access Protocol
<i>OASIS</i>	Organization for the Advancement of Structured Information Standards
<i>PEP</i>	Policy Enforcement Point
<i>PDP</i>	Policy Decision Point
<i>PAP</i>	Policy Administration Point
<i>PIP</i>	Policy Information Point
<i>SQL</i>	Structured Query Language
<i>XACML</i>	eXtensible Access Control Markup Language
<i>XML</i>	eXtensible Markup Language



## 3 Literaturverzeichnis

### 3.1 Spezifikationen und Standards

[XACML 2.0] OASIS XACML 2.0 Core Specification  
eXtensible Access Control Markup Language (XACML) v2.0 Errata  
29 June 2006  
[http://www.oasis-open.org/committees/download.php/19135/access\\_control-xacml-2.0-core-spec-os-errata.zip](http://www.oasis-open.org/committees/download.php/19135/access_control-xacml-2.0-core-spec-os-errata.zip)

### 3.2 Artikel und Arbeiten

[CerStr07PAP] Massimo Cerqui, Sandro Strebel  
HERAS<sup>AF</sup> – Holistic Enterprise-Ready Application Security Architecture Framework  
Prototyp einer Richtlinienverwaltung auf Basis von Spring/JSF,  
November 2007  
Diplomarbeit an der Hochschule für Technik, Rapperswil.

[CerStr07PEP] Massimo Cerqui, Sandro Strebel  
HERAS<sup>AF</sup> – Holistic Enterprise-Ready Application Security Architecture Framework  
Interzeptoren für Spring AOP und AspectJ,  
Juli 2007  
Studienarbeit an der Hochschule für Technik, Rapperswil.

[DOH07DADev] Sacha Dolski, Stefan Oberholzer, Florian Huonder  
HERAS<sup>AF</sup> – Holistic Enterprise-Ready Application Security Architecture Framework  
XACML 2.0 Implementation Developer's guide,  
November 2007  
Diplomarbeit an der Hochschule für Technik, Rapperswil.

[DOH07SA] Sacha Dolski, Stefan Oberholzer, Florian Huonder  
HERAS<sup>AF</sup> – Holistic Enterprise-Ready Application Security Architecture Framework  
XACML PDP Web Service Endpoint,  
Juli 2007  
Studienarbeit an der Hochschule für Technik, Rapperswil.



- [Egg06]* René Eggenschwiler  
HERAS<sup>AF</sup> – Holistic Enterprise-Ready Application Security Architecture Framework  
Manageable policy-based access control for J2EE.  
Mai 2006  
Diplomarbeit an der Hochschule für Technik, Rapperswil.
- [Graf06]* Yan Graf  
HERAS<sup>AF</sup> – Holistic Enterprise-Ready Application Security Architecture Framework  
Distributed Access Control Policies.  
Dezember 2006  
Diplomarbeit an der Hochschule für Technik, Rapperswil.
- [LDAPDraft]* OASIS LDAP  
LDAP profile for distribution of XACML policies Working draft 01  
17 October 2003  
<http://lists.oasis-open.org/archives/xacml/200310/bin00000.bin>
- [PressInterop]* OASIS Press Release  
Eight Companies Demonstrate Interoperability of XACML  
28 June 2006  
<http://www.oasis-open.org/news/xacml-interop-2007-press-release.pdf>
- [Sun05]* Sun Microsystems  
XACML: Access Control, Under Control.  
[http://research.sun.com/spotlight/2005\\_11\\_01-XACML.html](http://research.sun.com/spotlight/2005_11_01-XACML.html)

### 3.3 Sonstiges

- [XACMLKonf]* OASIS XACML Konformitätstests  
XACML Version 2.0 Conformance Tests (draft), 10 October 2005  
<http://www.oasis-open.org/committees/download.php/14846/xacml2.0-ct-v.0.4.zip>