



---

Diplomarbeit

---

# HERAS<sup>AF</sup>

Holistic Enterprise-Ready Application Security Architecture Framework

Manageable policy-based access control for J2EE.

---

René Eggenschwiler  
rene.eggenschwiler@bluewin.ch

Abteilung Informatik  
Hochschule für Technik Rapperswil

Mai 2006

Betreuer:  
Josef M. Joller, Hochschule für Technik Rapperswil

Experte:  
Wolfgang Giersche, Zühlke Engineering AG



## Danksagung

An dieser Stelle möchte ich allen danken, die durch ihre fachliche und persönliche Unterstützung zum Gelingen dieser Diplomarbeit beigetragen haben.

Herrn **Prof. Dr. Josef Joller** danke ich sehr für die Ermöglichung und Förderung meiner Diplomarbeit. Er zeigte starkes Interesse an der Arbeit und war stets gewillt, HERAS<sup>AF</sup> weiterzubringen. Die Zusammenarbeit mit Herrn Joller war sehr angenehm. Dank seiner unkomplizierten, flexiblen Art konnten Besprechungen und Beratungen auch spontan durchgeführt werden. Weil er das selbständige, freie Arbeiten befürwortet, konnte ich mich voll auf meine Arbeit konzentrieren und wurde nicht durch unnötige Sitzungen von der Arbeit abgehalten.

**Wolfgang Giersche** möchte ich danken für sein Engagement als Experte und seinen grossen Einsatz für den Showcase von HERAS<sup>AF</sup>. Herr Giersche hat stets intensiv an der technischen Umgebung meiner Diplomarbeit gearbeitet und ist mitverantwortlich für das Gelingen des HERAS<sup>AF</sup> Prototypen. Von seiner Erfahrung als J2EE-Architekt und Sicherheitsspezialist konnte ich sehr viel profitieren.

Yan Graf, der auch ein Träger des Projekts HERAS<sup>AF</sup> ist, möchte ich danken. Er hat Wolfgang Giersche bei der Entwicklung des Showcases unterstützt. Yan Graf wird im Wintersemester 06/07 auch eine Diplomarbeit schreiben, welche weitere Themen von HERAS<sup>AF</sup> behandelt.

Der alabus ag danke ich für die Ausbildung, die es mir ermöglichte, an der HSR aufgenommen zu werden. Die alabus ag und ihre Mitarbeiter unterstützten mich während meines Studiums sehr herzlich, insbesondere bei der Betreuung verschiedener Projektarbeiten.

Ganz besonderer Dank geht an meine Familie. Meine Eltern haben mich moralisch sowie finanziell unterstützt und mich grosszügig durch das Studium begleitet.

Meiner Freundin Caterina, sowie meinem sozialen Umfeld danke ich besonders. Meinen Kollegen und Kolleginnen danke ich für ihre Geduld, ihr Vertrauen und ihr Verständnis, das sie mir entgegenbrachten.



## Inhaltsverzeichnis

<b>1</b>	<b>Abstract</b> .....	<b>5</b>
<b>2</b>	<b>Einführung</b> .....	<b>7</b>
2.1	Ziele der Diplomarbeit .....	7
2.2	Ziele des Prototypen und der Anschauungsstudie .....	7
<b>3</b>	<b>Zugriffskontroll- und Autorisierungsstrategien</b> .....	<b>8</b>
3.1	Zugriffskontrolllisten .....	10
3.2	Rollenbasierte Zugriffskontrolle .....	11
3.3	Richtlinienbasierte Zugriffskontrolle .....	12
3.3.1	eXtensible Access Control Markup Language (XACML) .....	14
3.4	Anforderungen für Unternehmenstauglichkeit .....	19
3.4.1	Unternehmenstaugliche Zugriffskontrolle .....	20
<b>4</b>	<b>HERAS<sup>AF</sup> Hintergrund</b> .....	<b>23</b>
4.1	Motivation .....	23
4.2	Credo.....	23
<b>5</b>	<b>HERAS<sup>AF</sup> Logik</b> .....	<b>25</b>
5.1	Policy Enforcement Point .....	25
5.2	Policy Decision Point.....	26
5.3	Kommunikation PEP↔PDP .....	27
5.4	Policy Information Point.....	28
5.5	Policy Store .....	28
5.6	Policy Administration Point.....	28
<b>6</b>	<b>HERAS<sup>AF</sup> Technik</b> .....	<b>30</b>
6.1	Architektur .....	30
6.2	Kern (heras-core, heras-hib3, heras-sunxacml).....	32
6.3	Policy Enforcement Point (heras-acegi, heras-com) .....	33
6.4	Policy Decision Point (heras-pdp, heras-com) .....	33
6.5	Policy Information Point.....	33
6.6	Policy Store .....	33
6.7	Policy Administration Point (heras-pap) .....	35
6.8	Verwendete Frameworks und Technologien.....	35
6.8.1	Entwicklungstools .....	36
6.8.2	Sun's XACML Implementation.....	36
6.8.3	OpenSAML .....	36
6.8.4	Spring Framework .....	37
6.8.5	Hibernate 3 (ORM) .....	37
6.8.6	Acegi Security.....	38
6.8.7	Tomcat 5.5.....	38
6.8.8	Apache Axis (Apache eXtensible Interaction System).....	38
<b>7</b>	<b>HERAS<sup>AF</sup> Implementierung</b> .....	<b>39</b>
7.1	heras-core .....	41
7.1.1	Package-Design .....	41
7.1.2	Modell (Objektstruktur für XACML 2.0).....	44
7.1.3	Vorlagenmechanismus (Templating).....	46
7.1.4	Erweiterungspunkt: Persistenz und Datenzugriff.....	48
7.1.5	Erweiterungspunkt: XACML-Implementation.....	49
7.2	heras-sunxacml .....	50
7.3	heras-hib3 .....	50



7.3.1	Datenbank .....	52
7.4	Design Qualitätsanalyse.....	53
<b>8</b>	<b>Resultate .....</b>	<b>54</b>
8.1	Diplomarbeit .....	54
8.2	HERAS <sup>AF</sup> Prototype und Anschauungsstudie .....	54
8.3	Fazit.....	54
<b>9</b>	<b>Appendix .....</b>	<b>55</b>
9.1	Definitionen .....	55
9.2	Abkürzungen .....	55
9.3	Literaturverzeichnis .....	55
9.3.1	Spezifikationen, Standards und Definitionen .....	55
9.3.2	Bücher .....	56
9.3.3	Artikel, Arbeiten .....	57
9.3.4	Sonstige.....	59



# 1 Abstract

## Ausgangslage

Praktisch alle grösseren Organisationen, wie etwa öffentliche Einrichtungen oder kommerzielle Unternehmen, betreiben ein komplexes System von IT-Anwendungen. Daher besteht für sie die Notwendigkeit, eine zuverlässige, kundenspezifische Autorisierungslösung zur Verfügung zu stellen. Dies ist eine echte Herausforderung.

Wenn es um die Entscheidung geht, wer, auf welche Art und Weise, zu welchen Ressourcen Zugriff hat, können Richtlinien und Policies kompliziert werden und sind schwierig zu handhaben. Die Meisten kommerziellen und nicht gewerblichen Anwendungen kommen mit ihrem eigenen Benutzerspeicher und ihren proprietären Zugriffssteuerungsmechanismen daher. Quelloffene, standardisierte Lösungen gibt es noch keine.

Mit HERAS<sup>AF</sup> wird ein quelloffenes Projekt gestartet, das sich zum Ziel setzt, eine zentral verwaltbare, unternehmenstaugliche Autorisierungslösung zu realisieren. HERAS<sup>AF</sup> baut auf frei verfügbaren, etablierten und zukunftsträchtigen Technologien und Standards auf. Dabei stehen Interoperabilität, Erweiterbarkeit und Austauschbarkeit der integrierten Komponenten im Vordergrund. Diese Diplomarbeit realisiert den Kern von HERAS<sup>AF</sup> und legt somit die Basis für das Projekt und dessen Prototyp.

## Ziele der Diplomarbeit

Design, Realisierung und Dokumentation...

- ...des logischen Struktur-Modells von HERAS<sup>AF</sup> (XACML 2.0 konform).

- ...der Integration von Sun's XACML Implementation

- ...der Persistenz des Modells mit Hibernate 3

- ...einer API für die Benutzung des Modells und der Logik des Policy Administration Points

Dabei standen die Framework-Architektur und die Aufteilung von Modellen, hinsichtlich ihrer Anforderungen (engl. separation of concerns), im Vordergrund. Komponenten, die unterschiedliche Belange erfüllen, müssen separiert und einzeln austauschbar sein. Beispielsweise müssen die Persistenzkomponente und die XACML-Implementation voneinander völlig unabhängig sein.

Zeitgleich zur Durchführung der Diplomarbeit wurde ein vertikaler Prototyp von HERAS<sup>AF</sup> entwickelt. Die für den Showcase benötigten Komponenten, welche nicht Teil der Diplomarbeit waren, wurden von Drittpersonen entwickelt. Daher war die Integration des Kerns in das gesamte Framework ebenso Teil der Aufgabenstellung dieser Diplomarbeit. Die Kommunikation und Koordination mit dem Projektteam gehörte ebenso dazu.

## Lösung

In der ersten Phase der Arbeit wurde das Struktur-Modell von HERAS<sup>AF</sup> entwickelt. Dieses integriert nahezu alle Definitionen und Funktionen der Richtlinienbeschreibungssprache XACML. Die Aspekte von Persistenz und XACML Integration wurden dabei komplett voneinander getrennt. In der zweiten Phase des Projektes wurden die Anforderungen und Funktionen realisiert, die den Anspruch der Unternehmenstauglichkeit erfüllen.



Dazu gehört die Abstraktion der technischen XACML-Sprache zu einer frei definierbaren Geschäftssprache. Diese Funktionen wurden durch ein konfigurierbares Vorlagensystem (engl. template mechanism) realisiert.

Entstanden ist eine Multi-Projekt-Lösung, mit verschiedenen verteilbaren Komponenten in Java und J2EE. Somit wurde ein Framework realisiert, das in einer Multi-Server-Umgebung bzw. Multi-Anwendungs-Umgebung eingesetzt und gewartet werden kann.

Durch konsequenten Einsatz des Spring IoC-Containers wurden die Abhängigkeiten zwischen Komponenten, Modulen und Paketen minimal gehalten. Dies garantiert die hohe Flexibilität, Erweiterbarkeit und Skalierbarkeit des Frameworks.



## 2 Einführung

Diese Arbeit wurde als Einzelarbeit, im Rahmen einer Diplomarbeit an der Hochschule für Technik in Rapperswil, durchgeführt. Sie wurde zur Erlangung des akademischen Grades eines Diplomingenieurs der Studienrichtung Informatik vorgelegt.

Zusätzlich sollte die Arbeit der Lancierung des quelloffenen Software Projektes HERAS<sup>AF</sup> dienen. Für das Projekt wurde ein vertikaler Prototyp entwickelt, welcher die Ergebnisse der Diplomarbeit verwendet. Die zusätzlichen Module für den Prototyp wurden von externen Projektmitgliedern entwickelt.

Deshalb wurden verschiedene Ziele für die Diplomarbeit und HERAS<sup>AF</sup> definiert.

### 2.1 Ziele der Diplomarbeit

Die konkreten Ziele für die Diplomarbeit lauteten: Design und technische Realisierung und Dokumentation...

- ... des logischen Struktur-Modells von HERAS<sup>AF</sup>
- ... der Integration von Sun's XACML Implementation [6.8.2]
- ... der Persistenz des Modells mit Hibernate 3 [6.8.5]
- ... einer API für die Benutzung des Modells und der Logik des Policy Administration Points

Die in diesem Rahmen entstandene Arbeit realisiert somit den Kern von HERAS<sup>AF</sup>. Alle anderen HERAS<sup>AF</sup>-Module bauen darauf auf oder verwenden den Kern. Daher war auch die Integration des Kernmoduls in HERAS<sup>AF</sup> Teil der Diplomarbeit.

### 2.2 Ziele des Prototypen und der Anschauungsstudie

Für die Lancierung von HERAS<sup>AF</sup> wurde ein vertikaler Prototyp benötigt, bestehend aus:

- HERAS<sup>AF</sup>-PDP, unter Verwendung von Sun's XACML Implementation [6.8.2]
- HERAS<sup>AF</sup>-PAP
- HERAS<sup>AF</sup>-PEP, unter Verwendung von Acegi Security System for Spring [6.8.6]
- Einer Beispiel-Applikation die den PEP verwendet, um die Autorisierung zu demonstrieren

Diese Module werden bei der Bewertung der Diplomarbeit nicht berücksichtigt.



### 3 Zugriffskontroll- und Autorisierungsstrategien

Unter Zugriffskontrolle versteht man die Fähigkeit, die Benutzung eines Objekts (eine passive Entität, Ressource) durch ein Subjekt (eine aktive Entität, eine Person oder Prozess) zu erlauben, oder zu verbieten [WikiAC06].

Zugriffskontrolle beinhaltet Authentisierung, Authentifizierung, Autorisierung und Auditierung.

Die Authentisierung ist das Nachweisen einer Identität, die Authentifizierung deren Überprüfung. Die Autorisierung bezeichnet die Zuweisung und Überprüfung von Zugriffsrechten auf Daten, Dienste und Ressourcen an Nutzer oder Programme. Die Auditierung protokolliert Systemaktivitäten, um eine Analyse der Ereignisse und eine Rekonstruktion ihrer Änderungen zu ermöglichen. HERAS<sup>AF</sup> konzentriert sich speziell auf die Autorisierungsthematik.

In der Autorisierung geht es also darum, eine Antwort auf folgende Frage zu finden:

**„Wer darf was, wie und unter welchen Umständen benutzen?“**

Anhand dieser Fragestellung lässt sich erkennen, dass die Antwortfindung beliebig komplex werden kann. Dazu ein kleines (erfundenes) Beispiel aus einem Krankenhaus:

- Ein betreuender Arzt darf die Krankengeschichte ausschliesslich von seinen Patienten einsehen.
- Assistenzärzte dürfen keine Änderungen an den Patientendaten machen, ausser von 01:00 bis 04:00 Uhr, weil dann kein Chefarzt arbeitet. Die Änderungen müssen dann von einem Chefarzt bestätigt werden.
- Ein Notfallarzt kann alle Daten einsehen, sofern ein ernsthafter Notfall besteht oder er die Erlaubnis des Patienten und Hausarztes hat.
- Ein stellvertretender Arzt hat die gleichen Einsichtsmöglichkeiten wie der Arzt den er vertritt, sofern dieser mehr als drei Tage nicht arbeitet.
- Hilfspersonal darf nur die Medikamentenliste und Rezepte eines Patienten einsehen
- Ein Ober- oder Stationsarzt darf alle Daten der Patienten einsehen, die auf seiner Station liegen.

Die Komplexität der Zugriffsberechtigung ergibt sich aus der Menge von Attributen und deren Beziehung zueinander, die zu berücksichtigen sind. Jedes Objekt, das in den Autorisierungsprozess involviert ist, wird durch verschiedene Attribute beschrieben. Aufgrund der Objektattribute werden Zugriffsentscheidungen getroffen. Zusätzlich können auch Attributbeziehungen den Ausschlag geben, ob Zugriff erlaubt wird. Dies ist besonders bei einer hierarchischen Organisation von Attributen der Fall. Attributhierarchien werden aus den verschiedenen natürlichen Hierarchien in Organisationen und Unternehmen abgeleitet. Ansätze für solche Hierarchien finden sich üblicherweise in jedem „W“ der oben genannten Fragestellung.

„Wer“ oder „Was“ auf eine Ressource zugreifen möchte, wird als Subjekt bezeichnet. Subjekte sind hierarchisch und/oder in parallelen Strukturen organisiert. So organisiert jede grössere Unternehmung oder Organisation ihre Mitarbeiter in Hierarchien, wie zum Beispiel Personalstrukturen, Abteilungsstrukturen und Projektstrukturen. Diese Strukturen haben einen direkten Einfluss darauf, welche Berechtigungen erteilt werden.



Unternehmungen wandeln sich mit Zeit und sind vom Fortschritt betroffen. Firmenübernahmen, Fusionen, Auslagerungen und Neugründungen führen regelmässig zu Änderungen in Unternehmensstrukturen, was auch zu Änderungen in der Zugriffskontrolle führt.

Das Ziel des Zugriffs, also das „Was“, wird als Ressource bezeichnet. Ressourcen sind üblicherweise auch hierarchisch organisiert. So werden beispielsweise Dateisysteme in Ordnerstrukturen verwaltet. Dateien und Akten können aus verschiedenen Dateneinheiten zusammengesetzt werden. Diese Dateneinheiten können relevante Informationen für verschiedene Subjekte beinhalten. Subjekte (Chefarzt, Krankenpflegerin) können daher verschiedene Sichten auf die gleiche Ressource (Patientenakte) haben. Jedes Subjekt interpretiert folglich unterschiedliche Objekte als Ressourcen. Das kann zum Beispiel anhand thematischer, technischer, prozessrelevanter oder sonstiger Kriterien unterschieden werden. Das hängt ganz von der Organisation der Unternehmung, den Geschäftsprozessen, sowie den Geschäftsobjekten ab.

„Wie“ jemand auf eine Ressource zugreift, bezeichnet man als Aktivität. Aktivitäten lassen sich aufgrund ihrer Dringlichkeit, Gefährlichkeit oder Beziehung zu anderen Aktivitäten unterteilen. Auch gesetzliche Rahmenbedingungen wie zum Beispiel Datenschutz haben einen wesentlichen Einfluss darauf, welche Aktivitäten ausgeführt werden dürfen.

Der Zustand der Umwelt ist eine weitere flexible Einflussgrösse. Mit dem Umweltzustand können diverse Informationen einbezogen werden, die für die Zugriffsentscheidung relevant sein können. Dabei geht es oft um Informationen, welche Werte für die Einhaltung von Rahmenbedingungen beschreiben. Beispielsweise dürfen gewisse Aktivitäten nur in beschränkten Zeiträumen ausgeführt werden. Die aktuelle Uhrzeit liefert dabei einen Wert, der den Umweltzustand beschreibt. Umgebungsinformationen werden in vielen Fällen auch einbezogen. Zum Beispiel kann ein Zugriff nur erlaubt werden, wenn andere Bedingungen innerhalb von Geschäftsprozessen oder Organisationen erfüllt sind.

Daraus ergeben sich sehr flexible und komplexe Bedingungen und Berechtigungen in der Zugriffskontrolle. Diese zu automatisieren und wirtschaftlich zu verwalten ist eine grosse Herausforderung für Unternehmungen oder Organisationen.

Eine zuverlässige Autorisierung ist für Unternehmungen aus diversen Gründen überlebenswichtig. Lückenhafte oder fehleranfällige Zugriffskontrollen können einer Organisation massive Schäden zufügen. Diese können sich in Form von Kosten oder aber gesetzlichen Sanktionen ausdrücken. Solche Schäden treten beispielsweise auf bei:

- Datenverlust durch unautorisierte Löschooperationen
- Nichteinhaltung von Datenschutzbestimmungen
- Ausführen von sicherheitskritischen Operationen
- Veröffentlichung von unternehmensstrategischen Informationen
- Unautorisierter Zugriff auf IT-Anwendungen durch Drittpersonen oder externe Programme

Mit Hinblick auf die stetig wachsende (weltweite) Vernetzung und Verteilung von Applikationen, wächst auch die Gefahr eines Fremdzugriffs. Daher ist die Zugriffskontrolle in vernetzten Anwendungen ein zentrales Thema. In folgenden Bereichen wird besonders intensiv an Autorisierungslösungen gearbeitet:

- verteiltes Rechnen (Grid-Computing)
- richtlinienbasierte Netzwerkverwaltung (policy-based network management)
- XML-Dokument-Sicherheit (xml document security)



Autorisierung kann verschieden implementiert werden. Die wohl bekanntesten Strategien sind: Rollenbasierte Zugriffskontrolle, Zugriffskontrolllisten und richtlinienbasierte Zugriffskontrolle.

### 3.1 Zugriffskontrolllisten

Die Strategie der Zugriffskontrolllisten (engl. Access Control Lists, ACL) stammt aus der Betriebssystemensicherheit und wurde im Zusammenhang mit der Entwicklung von Betriebssystemen eingeführt. Zugriffskontrolllisten werden sehr abstrakt modelliert und beschäftigen sich mit dem Schutz von Ressourcen im Allgemeinen. Die heute verbreiteten Betriebssysteme (Windows, Linux, Unix usw.) verwenden Zugriffskontrolllisten um den Zugriff auf Dateien, das Dateisystem (Ordnerstrukturen) und Dienste (zum Beispiel Netzwerkdienste) zu kontrollieren. Grundsätzlich folgt man dabei dem Ansatz, Privilegien verschiedener Benutzer zu separieren.

Die Zugriffskontrollliste ist eine Datenstruktur, üblicherweise eine Tabelle. Jeder Eintrag in dieser Tabelle spezifiziert, welcher Benutzer oder welche Benutzergruppe bestimmte Zugriffsrechte zu einem Objekt (Datei, Dienst, Programm) hat.

Microsoft Windows Betriebssysteme haben die Zugriffskontrolllisten auf die, in Abbildung 1 gezeigte, Art implementiert:

Jedes Objekt (z.B. eine Datei, ein Dateiordner, ein Eintrag im Active Directory, ein Registrierungsschlüssel) besitzt zur Speicherung der Zugriffsrechte eine Sicherheitsbeschreibung (engl. Security Descriptor).

Eine Sicherheitsbeschreibung besteht aus drei Teilen:

- Aus dem Security Identifier (SID) des Besitzers
- Aus einer Discretionary ACL (DACL), die die Zugriffsrechte beschreibt.
- Aus einer System ACL (SACL), die die Überwachungseinstellungen enthält.

Eine Access Control List (ACL) (sowohl DACL als auch SACL) besteht aus Access Control Entries (ACE). Ein ACE wiederum enthält folgende Informationen:

- Treuenehmer (Trustee): der SID des Benutzers bzw. der Gruppe.
- AccessMask: Die AccessMask definiert die Rechte. Für jeden Objekttyp gibt es unterschiedliche Rechte. Jedes Recht ist dabei ein Bit bzw. eine Kombination von Bits in diesem Long-Wert. Eine AccessMask besteht in der Regel aus der Addition mehrerer einzelner Zugriffsrechte.
- ACE-Flags: Über die ACE-Flags wird die Vererbung der Rechte gesteuert.

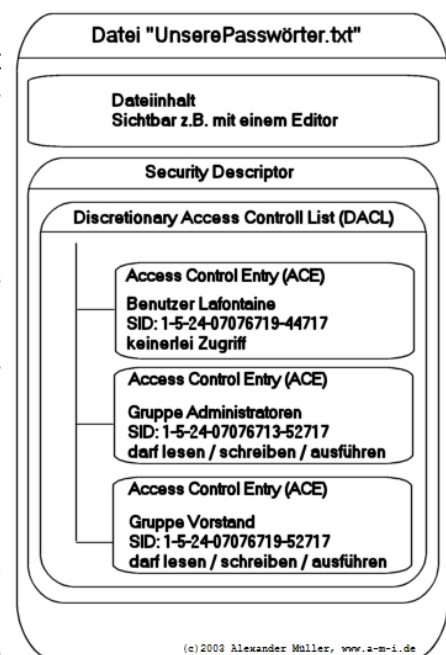


Abbildung 1: Windows ACL Struktur



Oft ist die Zugriffskontrolle mittels Listen für Firmen und deren Sicherheitsanforderungen nicht umfassend genug. So sind etwa die Anforderungen nicht genügend komfortabel umsetzbar oder verwaltbar. Die Verwaltung der Listen kann sehr aufwändig werden, weil die Zugriffsinformationen pro Ressource vorhanden sein müssen. Diese Informationen zu warten und zu pflegen kann deshalb sehr kostenintensiv sein. Bei Zugriffskontrolllisten lassen sich Abhängigkeiten verschiedener Aktionen und Berücksichtigungen des Umgebungszustands nicht in die Zugriffsberechtigung integrieren. Daher können natürliche Hierarchien und Umgebungsattribute kaum oder nur schwer in die Zugriffskontrollen miteinbezogen werden.

### 3.2 Rollenbasierte Zugriffskontrolle

Das Konzept der rollenbasierten Zugriffskontrolle (engl. Role-based Access Control, RBAC) wurde im Zusammenhang mit Multi-Benutzer- und Multi-Applikationen-Systemen in den 1970er-Jahren entwickelt [SCHFY95]. Der zentrale Punkt von RBAC ist dabei die Gruppierung von Zugriffsberechtigungen in Benutzerrollen.

Eine Benutzerrolle (oder kurz Rolle) definiert Aufgaben, Eigenschaften und vor allem Rechte eines Benutzers in einer Software bzw. in einem Betriebssystem. Benutzerrollen werden verwendet um die Einstellungen der vorgenannten Bereiche nicht für jeden Nutzer einzeln festlegen zu müssen. Statt Benutzern Rechte direkt zuzuweisen, wird eine Benutzerrolle definiert, die dann mehreren Benutzern zugeordnet werden kann. Dies erleichtert die Rechteverwaltung des Softwaresystems, da insbesondere bei Änderungen der Rechtstruktur, nur die Rechte der Benutzerrollen angepasst werden müssen.

Benutzerrollen werden aus den verschiedenen Geschäftsrollen und Arbeitsfunktionen abgeleitet. Eine Rolle ist daher ein semantisches Konstrukt, um welches die Sicherheitsrichtlinien formuliert werden. Die Benutzer erhalten die benötigten Rollen aufgrund ihrer Zuständigkeit, Qualifikation, Kompetenz und Verantwortung.

Einem Benutzer können verschiedene Rollen zugeordnet werden. Zusätzlich können Rollen hierarchisch strukturiert werden. Rollenhierarchien stellen eine einfache Möglichkeit dar, Organisationsstrukturen abzubilden und Zugriffsberechtigungen anhand natürlicher Kriterien zuzuweisen [Kuhl05].

Die Benutzerrollen vereinfachen die Verwaltung von Zugriffsberechtigungen, weil sie der Organisation und deren Funktionen besser entsprechen als Zugriffskontrolllisten. Änderungen in der Zugriffsberechtigung von Mitarbeitern (zum Beispiel Karriereaufstieg, Jobwechsel) können sehr komfortabel durch das Zuweisen oder Entziehen von Rollen realisiert werden.

Trotzdem können rein rollenbasierte Autorisierungslösungen den Anforderungen eines Unternehmens teilweise nicht genügen. Wenn die Zugriffsberechtigung eines Benutzers nicht ausschliesslich aufgrund seiner Rollenzuteilung bestimmt werden kann, deckt die rollenbasierte Autorisierung ungenügend Anwendungsfälle ab. Für Unternehmen, die nicht rollenorientiert arbeiten und sehr dynamische Strukturen haben, ist der Verwaltungsaufwand unverhältnismässig kostenintensiv. Innerhalb projektorientiert arbeitender Organisationen, deren Projekte jeweils von kurzer Dauer sind, wechseln auch oft die Rollen der Mitarbeiter. Dies führt dazu, dass diese auch immer wieder neu zugewiesen bzw. entzogen werden müssen.

Bei grossen Änderungen von Unternehmensstrukturen, beispielsweise Fusionen, kann es zudem sein, dass die Benutzerrollen und –gruppen weitläufig überarbeitet werden müssen.



### 3.3 Richtlinienbasierte Zugriffskontrolle

Alexander Keller und Heiko Ludwig beschreiben in [KeLu04], dass im Systemmanagement Richtlinien (engl. Policies) als ein zentrales Werkzeug angesehen werden, um die kontinuierlich steigende Komplexität und die Dynamik verteilter Systeme zu bewältigen. Durch die Verwendung von Richtlinien erreicht man eine Automatisierung der Managementprozesse. Ein Administrator kann im Sinne des „Management by Objectives“ Verhaltensregeln definieren, die ein Managementsystem in die Lage versetzen, anhand dieser vorgefertigten Regeln selbstständig zu entscheiden, wie es sich beim Eintreffen bestimmter Ereignisse verhält.

Durch die Verwendung von Policies, ist ein Managementsystem in der Lage, auf die unterschiedlichsten Ereignisse korrekt zu reagieren, ohne dass seine zugrunde liegende Programmlogik modifiziert werden muss [Verm00]. Morris Sloman's allgemein akzeptierbare Definition von Policies spiegelt dieses Prinzip wieder: „Policies are rules governing the choices in behavior of a system“ [Slom94]. Richtlinien sind ausserdem dauerhaft und beschreiben wiederholbare Verhaltensmuster.

Die Internet Engineering Task Force (IETF) hat für richtlinienbasierte Architekturen eine geeignete Terminologie definiert [rfc3198]. In den RFC's 2748 [rfc2748] und 2753 [rfc2753] hat sie eine einfache Architektur festgelegt.

Das von der IETF [rfc3198] [rfc2748] und ISO [iso10181] definierte abstrakte Modell für Richtliniendurchsetzung (engl. policy enforcement) kann wie im Folgenden Bild dargestellt werden:

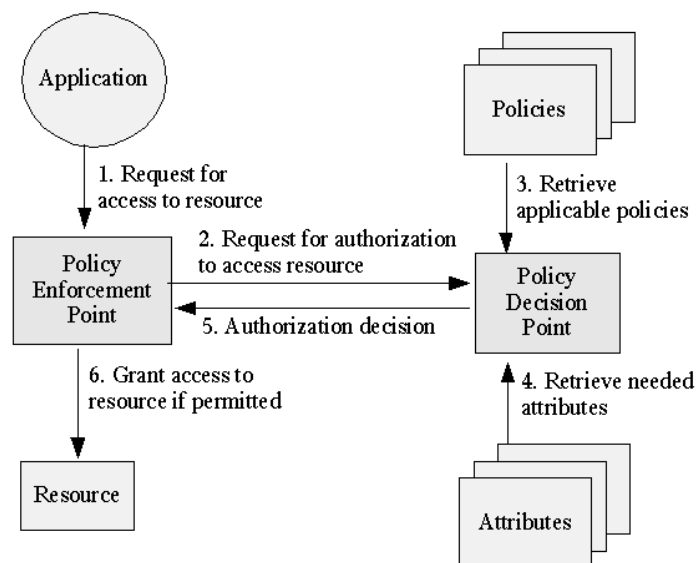


Abbildung 2: richtlinienbasierte Zugriffskontrolle

Ein Benutzer einer Applikation möchte auf geschützte Ressourcen zugreifen. Der Zugriff wird durch den „Policy Enforcement Point (PEP)“ unterbrochen. Er formuliert eine Anfrage und sendet diese an den „Policy Decision Point (PDP)“. Dieser wertet die Anfrage anhand der definierten Policies aus. Dazu benötigt er eventuell zusätzliche Informationen oder Attribute. Nach der Auswertung sendet der PDP seine getroffene Entscheidung zurück an den PEP. Der PEP lässt den Zugriff bei positiver Antwort zu, oder bricht den Zugriff ab.

Diese Architektur bringt grosse Vorteile mit sich:



- Die Autorisierung kann an einer zentralen Stelle durchgeführt werden. Ein PDP kann mehrere PEP's bedienen.
- Die Kommunikation zwischen PDP und PEP erfolgt über ein wohldefiniertes Protokoll. Dadurch lassen sich alle Policies in derselben Sprache formulieren. Nur die technische Realisierung von neuen PEP's muss entwickelt werden.
- Die Autorisierungsarchitektur kann als Service im Sinne einer Service-orientierten Architektur (SOA) implementiert werden. Dies führt zu höherer Entkopplung und Interoperabilität.

Um alle möglichen Zugriffskontroll-Szenarien abbilden zu können, sollte eine Sprache zur Beschreibung von Richtlinien folgende Anforderungen (wie Tim Moses [Mos03] und die XACML Spezifikation [XACML2.0] beschreibt) erfüllen können:

- Drei-Werte Logik
  - Wird eine Policy evaluiert, so muss das Resultat entweder ein Boolean „true“ oder „false“ zurückliefern. Kann keine Aussage getroffen werden, so muss das Resultat „indeterminate“ lauten.
- Kombinierbarkeit
  - Falls mehrere Policies für eine Anfrage zutreffen, so muss die Anfrage gegen jede zutreffende Policy evaluiert werden und deren Resultat eindeutig bestimmbar kombiniert werden können.
- Interpretierbarkeit von Instruktionen
  - Aus einer Policy-Instanz muss eine Menge von Instruktionen abgeleitet werden können, welche bestimmte Attribute in einer Anfrage lokalisieren, verstehen und evaluieren können.
- Verwendung allgemeingebäuchlicher Datentypen
  - Damit verschiedene Policy-Typen auf eine effektive Weise unterstützt werden können, muss ein allgemeingebäuchliches Datentypen-System verwendet werden.
- Erweiterbarkeit von Datentypen
  - Das Datentypen-System muss erweitert werden können, um spezifische Datentypen zu unterstützen, welche für Spezialanwendungen verwendet werden.
- Verwendung allgemeingebäuchlicher Operatoren
  - Damit verschiedene Policy-Typen auf eine effektive Weise unterstützt werden können, muss ein allgemeingebäuchliches Operatoren-System verwendet werden. Dies betrifft logische Operatoren, Vergleichsoperatoren, Zahlenoperatoren usw.
- Erweiterbarkeit von Operatoren
  - Das Operatoren-System muss erweitert werden können, um spezifische Operatoren zu unterstützen, welche für Spezialanwendungen verwendet werden.



Mit einer richtlinienbasierten Autorisierungslösung, können Unternehmen ihre Zugriffsbestimmungen beliebig detailliert und flexibel umsetzen. Verfeinerungen der Zugriffsberechtigungen können komfortabel, durch zusätzlich erstellte Richtlinien, realisiert werden. Die zentrale Verwaltbarkeit der Richtlinien führt zu komfortablem, überschaubarem Management der Zugriffskontrolle. Eine funktionierende rollenbasierte Autorisierungslösung verspricht daher Kostenreduktionen und eine bessere Anpassung an die Unternehmensanforderungen.

Die in der Literatur wohl meistbesprochenen Sprachen für die Definition von Richtlinien in der Zugriffskontrolle sind die "Enterprise Privacy Authorization Language (EPAL)" und die "eXtensible Access Control Markup Language (XACML)".

Die beiden Sprachen wurden in der Literatur bereits mehrfach miteinander verglichen. Einer der meist zitierten Vergleiche wurde von Ann Anderson verfasst [And05]. Der Vergleich zeigt, dass XACML über eine umfangreichere Funktionsvielfalt verfügt als EPAL. Zudem lässt sich EPAL aus einer Untermenge von XACML nachbilden. Deshalb verwendet auch HERAS<sup>AF</sup> XACML. Ausserdem wächst das allgemeine Interesse an XACML stark an, was auch ein Indiz für die zukünftige Verbreitung der Sprache darstellt.

### 3.3.1 eXtensible Access Control Markup Language (XACML)

XACML [XACML2.0] ist ein Standard, definiert von der Standardisierungsorganisation OASIS, für die universelle Beschreibung von Zugriffskontroll-Richtlinien unter Verwendung von XML.

XACML definiert die Syntax einer Beschreibungssprache für Policies, die Semantik für die Verarbeitung von Policies sowie für das Anfrage/Antwort-Format (die Kommunikation) zwischen PEP und PDP.

Die eXtensible Access Control Markup Language (XACML) Version 2.0 wurde im Februar 2005 vom OASIS-Konsortium ratifiziert. Mitglieder der OASIS-Gruppe wie Entrust, IBM, Hewlett-Packard, Sun Microsystems und weitere haben bei der Entwicklung der Spezifizierung von XACML mitgeholfen.

Sun Microsystems hat unter der Leitung von Seth Proctor eine quelloffene Referenzimplementation „Sun's XACML Implementation“ [SunXACML] entwickelt, und arbeitet derzeit an der Integration von XACML 2.0. HERAS<sup>AF</sup> stützt sich auf die kommende Veröffentlichung davon.

XACML unterstützt verschiedene andere OASIS Sicherheitsstandards, stellt Erweiterungspunkte zur Verfügung oder bindet andere Standards ein. Dazu definiert OASIS diverse Profile, welche die Anbindungen an die Standards beschreiben. Zu den mit XACML verwendbaren Standards gehören: Security Assertion Markup Language (SAML) [SAML2.0], WS-Security, Service Provisioning Markup Language (SPML), Digital Signature Services (DSS) und Public Key Infrastructure (PKI).

Der Datenfluss von XACML wird über verschiedene Einheiten bewerkstelligt. Abbildung 3 zeigt die wichtigsten Aktoren innerhalb des Datenflussmodells. Einige dieser Komponenten werden auch im Modell von SAML verwendet. Diese sind im Bild rot umrandet.

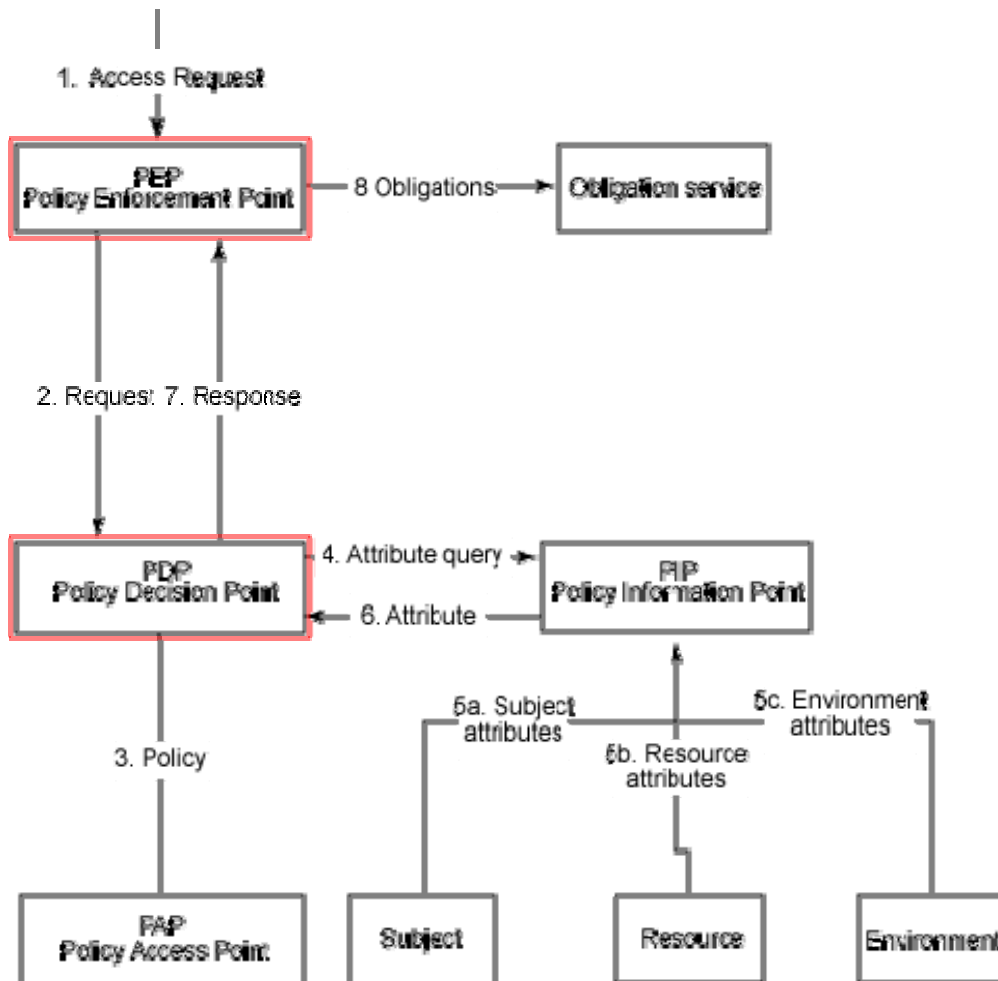


Abbildung 3: XACML Datenfluss

Will ein Benutzer auf eine gesicherte Ressource zugreifen, so wird die Zugriffsanfrage an den Policy Enforcement Point (PEP) gestellt (1).

Der PEP formuliert eine Autorisierungsanfrage (XACML) und sendet diese an den Policy Decision Point (PDP) (2).

Der PDP sucht anwendbare Richtlinien im Policy Store (3). Die gefundenen Policies werden ausgewertet und evaluiert.

Um die Werte der Attribute, die für die Evaluierung benötigt werden, zu erhalten, erfragt der PDP den Policy Information Point (PIP) (4).

Der PIP sucht die angefragten Werte heraus (5a bis 5c) und stellt die gefundenen Werte dem PDP zu Verfügung (6).

Der PDP evaluiert die Policies gegen die Autorisierungsanfrage und sendet dem PEP die resultierende Autorisierungsentscheidung (7). Die Entscheidung kann eine Zugriffserlaubnis, oder ein Untersagen des Zugriffs sein. Kann der PDP keine Entscheidung treffen, so lautet die Antwort: unbestimmbar (engl. indeterminate).

Diese Antwort des PDP's kann auch zusätzliche Verpflichtungen an den PEP beinhalten. Diese muss der PEP dann zuerst erfüllen (8) und kann dann die Anfrage fortsetzen.



In der Regel benötigt eine gesamte Policy mehrere einzelne Entscheidungen die getrennt voneinander ausgewertet und dann wieder zusammengesetzt werden. So kann z.B. der Zugriff auf personenbezogene Daten einerseits durch eine allgemeine Datenschutzrichtlinie und zusätzlich durch eine Firmenbezogene individuelle Policy eingeschränkt werden.

XACML definiert drei Top-Level Policy-Elemente: Rule, Policy und PolicySet. Das Zusammenspiel dieser Elemente wird in folgendem UML Diagramm erläutert [XACML2.0]:

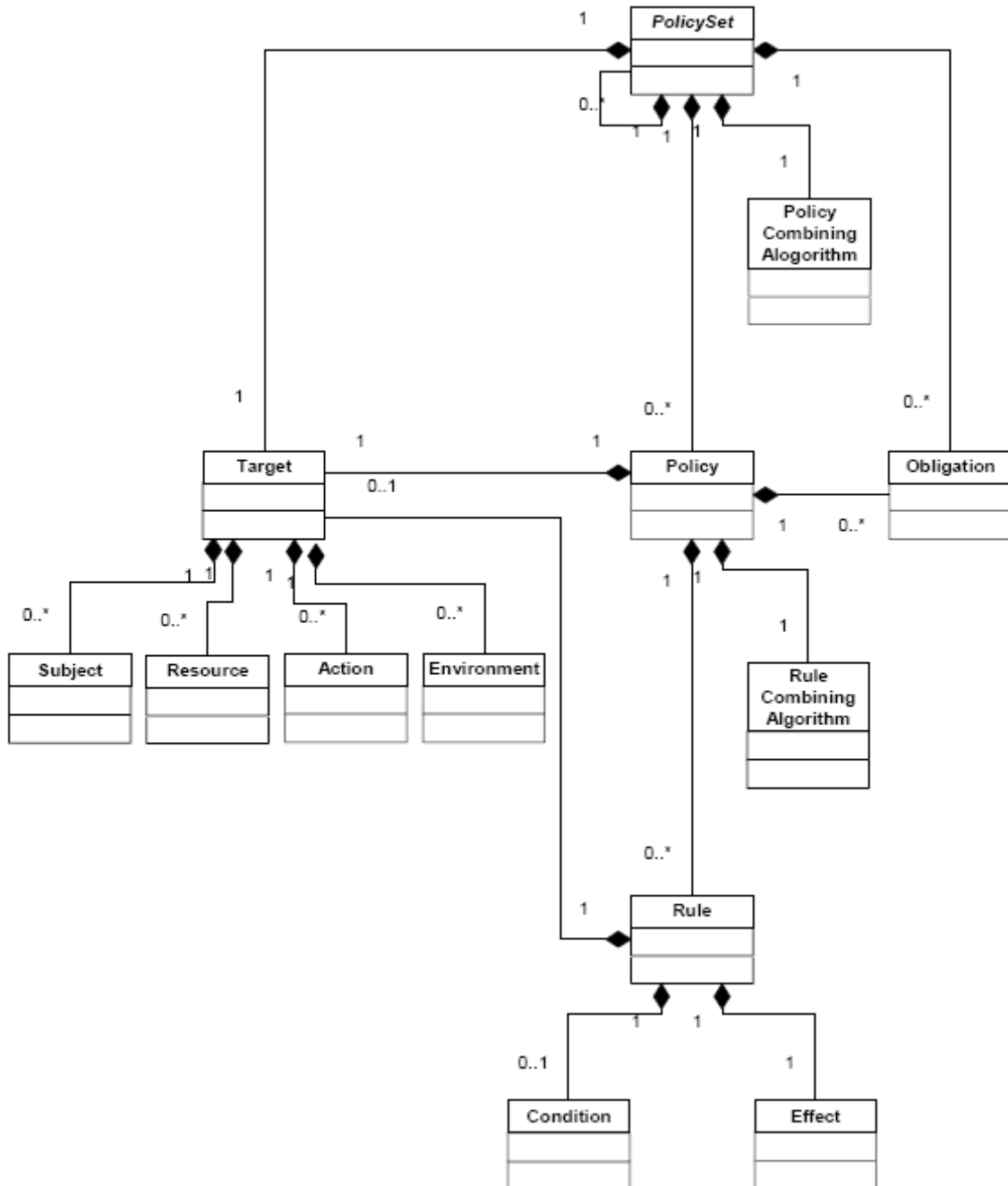


Abbildung 4: Modell XACML 2.0



Eine Policy, die für eine bestimmte Zugangsentscheidung benötigt wird, setzt sich in der Regel aus einer Vielzahl von Regeln (Set Of Rules) zusammen. Jede Policy hat einen bestimmten Algorithmus, welcher definiert, wie die einzelnen Regeln bei der Auswertung zu kombinieren und zu Bewerten sind. Es handelt sich hier um den Rule Combining Algorithm. Wenn zum Beispiel zwei Regeln für den Zugriff auf eine Ressource relevant sind, aber nur eine dieser beiden erfüllt ist, entscheidet der Algorithmus.

Mehrere Policies, die im gleichen Zusammenhang verwendet werden (z.B. bei einer Transaktion), werden in einem Policy Set zusammengefasst. Analog zur Auswertung der einzelnen Richtlinien-Regeln gibt es auch hier einen Algorithmus, welcher definiert, wie die unterschiedlichen Policies in einem Policy Set bei der Auswertung zu kombinieren sind. In diesem Zusammenhang spricht man von „Policy Combining Algorithm“

Jede Policy und jede Rule hat ein Target. Das Target wird über die vier Elemente Subject, Resource, Action und Environment klassifiziert. Diese Elemente werden für die Berechtigungsprüfung herangezogen.

Die Rules sind die Basiselemente einer Policy. Sie geben vor, wie bei der Behandlung des Target vorzugehen ist. Sie enthalten einen booleschen Ausdruck der durch den PDP ausgewertet wird.

An dieser Stelle wird nicht weiter auf das XACML-Modell eingegangen. In der Literatur finden sich viele Abhandlungen über XACML und dessen Modell:

- XACML 2.0 Core Specification  
[http://docs.oasis-open.org/xacml/2.0/access\\_control-xacml-2.0-core-spec-os.pdf](http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf)
- A Brief Introduction to XACML  
[http://www.oasis-open.org/committees/download.php/2713/Brief\\_Introduction\\_to\\_XACML.html](http://www.oasis-open.org/committees/download.php/2713/Brief_Introduction_to_XACML.html)
- XML-based Access Control Languages  
<http://seclab.dti.unimi.it/Papers/RI-3.pdf>



Eine einfache Richtlinie könnte zum Beispiel so lauten: „Jedes Subjekt mit einer Emailadresse aus der Domäne med.example.com kann jede Aktion auf jeder Ressource ausführen.“ Dieses Beispiel kann in XACML so aussehen [XACML2.0]:

```
<?xml version="1.0" encoding="UTF-8"?>
  <Policy
    xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:os"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="urn:oasis:names:tc:xacml:2.0:policy:schema:os
http://docs.oasis-open.org/xacml/access_control-xacml-2.0-policy-schema-
os.xsd"
    PolicyId="urn:oasis:names:tc:example:SimplePolicy1"
    RuleCombiningAlgId="identifier:rule-combining-algorithm:deny-overrides">
    <Description>
      Medi Corp access control policy
    </Description>
    <Target/>
    <Rule
      RuleId="urn:oasis:names:tc:xacml:2.0:example:SimpleRule1"
      Effect="Permit">
      <Description>
        Any subject with an e-mail name in the med.example.com domain
        can perform any action on any resource.
      </Description>
      <Target>
      <Subjects>
        <Subject>
          <SubjectMatch
            MatchId="urn:oasis:names:tc:xacml:1.0:function:rfc822Name-match">
            <AttributeValue
              DataType="http://www.w3.org/2001/XMLSchema#string">
              med.example.com
            </AttributeValue>
            <SubjectAttributeDesignator
              AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
              DataType="urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name"/>
            </SubjectMatch>
          </Subject>
        </Subjects>
      </Target>
    </Rule>
  </Policy>
```

**Abbildung 5: XACML Policydefinition**

Die wesentlichen Vorteile bei der Verwendung von XACML sind [SunXACML]:

- XACML ist standardisiert. Verwendet man eine standardisierte Sprache, so verwendet man etwas, das von einer grossen Gemeinschaft von Experten und Benutzern geprüft wurde. Für eine weite Verbreitung der Sprache, ist es somit nötig, dass sie ein offizieller Standard ist. Umso weiter sich eine Sprache verbreitet, desto höher wird die Interoperabilität zu anderen Applikationen, weil sie den gleichen Standard verwenden.
- XACML ist generisch. Dadurch lässt sich XACML in jeglicher Umgebung und für jegliche Ressourcen, Subjekte usw. einsetzen. Somit ist die Portabilität von Policies sichergestellt. Dies vereinfacht auch das Management von Richtlinien.
- Policies können verteilt werden und verschieden lokalisiert sein. Daher müssen grosse Policies, die aus weiteren Policies bestehen, nicht monolithisch an einem Ort gehalten werden. Somit können Policies an verschiedenen Orten, von verschiedenen Personen verwaltet werden.



- XACML bietet bereits umfangreiche Unterstützung von Datentypen und Funktionen. Zusätzlich bietet es komfortable Möglichkeiten für Erweiterungen.
- Verschiedene Standardisierungsgruppen arbeiten an Erweiterungen und Anbindungsprofilen, um XACML mit anderen Standards zu verwenden, wie zum Beispiel SAML, LDAP und Shibboleth.

Ein Nachteil bei der Verwendung einer so flexiblen Sprache ist der Mehraufwand, den man betreiben muss, um die vielen 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, so entsteht eine relativ grosse Nachricht, die Übertragen werden muss. Betrachtet man die effektiven Nutzdaten in dieser Nachricht, so sind diese sehr klein.

### 3.4 Anforderungen für Unternehmenstauglichkeit

Die allgemeine Erfahrung von Unternehmungen und Organisationen zeigt, dass IT-Sicherheitslösungen auf einer taktischen Grundlage entworfen, oder gekauft und installiert werden [SCL05b]. Üblicherweise wird eine Anforderung ausgearbeitet, eine passende Spezifikation entworfen und eine entsprechende Lösung gesucht.

In diesem Prozess gibt es nur wenige Möglichkeiten, die strategische Dimension zu betrachten. Eine mögliche Folge davon ist, dass die Organisation eine Mischung von technischen Lösungen aufbaut, welche zwar einzeln für sich und unabhängig funktionieren, jedoch ohne Garantie, dass diese kompatibel oder interoperabel sind.

Häufig gibt es auch keine Analysen der langfristigen Kosten. Besonders bei den operationalen Kosten, die einen grossen Anteil an den Gesamtkosten bilden, wären Analysen wichtig. Strategien für die Vorhersage des Geschäftsnutzens einer Sicherheitslösung sind bisher noch nicht bekannt. Daher kann auch nur sehr schwer gesagt werden, ob eine Lösung die Geschäftsziele optimal unterstützt.

Eine Annäherung, die diese bruchstückhaften Probleme vermeidet, ist die Entwicklung einer Unternehmenssicherheitsarchitektur, die sich an den Geschäftsabläufen orientiert und die eine strukturierte Beziehung zwischen den technischen und den Verfahrenslösungen beschreibt [SCL05p]. So kann der langfristige Nutzen des Geschäfts gestützt werden.

Die Entscheidungskriterien für eine Unternehmenssicherheitsarchitektur sollten von einem vollständigen Verständnis der Geschäftsanforderungen abgeleitet werden und umfassen [SCL05p]:

- Kostenreduktion (betreffend Entwicklungs-, Operations- und Administrationskosten)
- Modularität
- Skalierbarkeit (bezüglich Plattformen, Applikationen und Sicherheitslevel)
- Einfache Wiederverwendung von Komponenten
- Funktionsfähigkeit
- Nutzbarkeit
- Interoperabilität (intern sowie auch extern)
- Integration in die bestehende IT-Infrastruktur und in Legacy-Systeme
- Verwaltbarkeit



Die Integration von Sicherheit und Zugriffskontrolle kann nicht einfach anhand von Checklisten oder simplen Kontrollprozessen realisiert werden. Um IT-Sicherheit in Unternehmungen effektiv zu realisieren wird ein ganzheitlicher Ansatz benötigt, wie er zum Beispiel im System-Engineering angewendet wird. Eine Sicherheitslösung sollte daher in verschiedenen Schichten gegliedert werden, die jeweils die Anforderungen der zuständigen Geschäftsinstanzen erfüllen [SCL05p]:

- Geschäftsschicht (kontextuell)
- Architekturschicht (konzeptionell)
- Entwicklungsschicht (logisch)
- Entwicklungsschicht (physikalisch)
- Komponentenschicht
- Wartungsschicht (operationell)

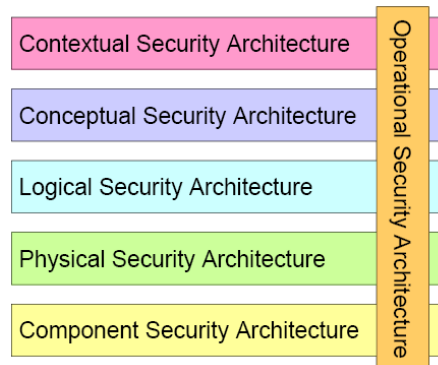


Abbildung 6: Systemschichten

Jede Schicht hat ihre eigenen Anreifer und Anforderungen. Optimalerweise erfüllt jede Schicht ihre Anforderungen und unterstützt die Erfüllung der Anforderungen der übergeordneten Schicht.

### 3.4.1 Unternehmenstaugliche Zugriffskontrolle

Bezieht man die oben beschriebenen Anforderungen und Ansätze in die Überlegungen ein, wie ein unternehmenstaugliches Zugriffskontrollsystem aussehen sollte, kristallisieren sich folgende Merkmale besonders heraus.

#### Verwaltbarkeit

- Ortunabhängig: Policies sollten zentral verwaltet werden können. Unabhängig davon, wo sich diese befinden, oder wohin sie publiziert worden sind.
- Plattformunabhängig: Die Verwaltung bzw. Definition und Verwendung von Policies soll plattformunabhängig sein. So können Systemwechsel oder –änderungen durchgeführt werden, ohne dass die Richtlinien unbrauchbar werden.
- Wiederverwendbarkeit: Eine erstellte Policy kann mehrfach publiziert und somit an anderen Orten (z.B. einer Geschäftsaussenstelle) wieder verwendet werden.
- Einheitlich: Jede gewünschte Applikation kann dieselbe Autorisierungslösung verwenden. Dies vereinfacht auch die Erfassung von ausgestellten Zugriffsberechtigungen.

#### Technische Abstraktion

- Geschäftsunterstützung: Ein Zugriffskontrollsystem muss so verwendet werden können, dass keine Unternehmensprozesse abgeändert bzw. angepasst werden müssen. Die Geschäftsprozesse stehen im Vordergrund und definieren die Arbeitsweise des Autorisierungssystems. Die Zugriffskontrolle muss so angewendet werden können, wie sie in der Geschäftslogik integriert ist und sollte die Prozesse nicht zusätzlich verkomplizieren.



- **Kompetenzbereichunterstützung:** Eine unternehmenstaugliche Anwendung unterstützt die verschiedenen technischen Fähigkeiten und Kompetenzen der Personen, welche die Zugriffskontrolle umsetzen und betreuen. Die Definition von Richtlinien kann in einer geschäftsorientierten Sprache vorgenommen werden. Technische Kompetenzen (wie z.B. den Aufbau von XACML zu kennen) dürfen Mitarbeitern nicht aufgezwungen werden, die keine Funktionen im technischen Umfeld ausüben. Zugriffsberechtigungen sollen damit in einer geschäftsspezifischen Sprache definiert werden können, wodurch Policies verständlicher werden und leichter aus der Geschäftslogik abgeleitet werden können.
- **Trennung der Geschäftsbelange** (engl. separation of business concerns): Die verschiedenen Anforderungen der einzelnen Unternehmensbereiche müssen sinnvoll unterteilt und in der Zugriffskontrolle entsprechend umgesetzt werden können. Dies beinhaltet beispielsweise, dass funktionspezifische Berichte generierbar sind, welche jeweils die gewünschten Informationen für einen Geschäftsbereich beinhalten. Die Möglichkeit Zugriffskontrollfunktionen spezifisch auf Geschäftsaufgaben zuzuschneiden, steigert die Effektivität in der Anwendung und der Betreuung einer Autorisierungslösung.
- **Anforderungsunterstützung:** Ein flexibles Zugriffskontrollsystem, sollte so verwendet werden können, dass alle Anforderungen und Anwendungsfälle eines Unternehmens erfüllbar sind. Dadurch werden ungewollte Kompromisse beim Einsatz einer Autorisierungslösung vermieden.

### **Erweiterbarkeit**

- **Unterstützung zukünftiger Infrastruktur:** Eine Autorisierungslösung darf den Fortschritt und die Neuerungen in der Infrastruktur einer Unternehmung nicht behindern. Werden neue Anwendungen eingeführt, oder bestehende Arbeitsmittel erneuert, so muss ein Zugriffskontrollsystem offen ausgelegt sein, so dass diese Änderungen in komfortabler Weise eingebunden werden können. Eine Autorisierungslösung darf den Fortschritt einer Unternehmung weder behindern noch verunmöglichen.
- **Skalierbarkeit:** Ein Zugriffskontrollsystem muss an die flexible Anzahl von Benutzern angepasst werden können. Steigt die Nutzeranzahl, so muss die Autorisierungslösung Mechanismen zur Verfügung stellen, um die Performanz und Funktionsbereitschaft sicherzustellen bzw. die Verfügbarkeit zu steigern. Auf eine unternehmenstaugliche Autorisierungslösung können übliche Verfahren angewendet werden, welche die Verfügbarkeit des Systems erhöhen. Dazu gehören beispielsweise Verfahren wie Caching, Clustering, Load Balancing und Backups.
- **Verwendung von Standards:** Um eine möglichst hohe Interoperabilität sicherzustellen, sollte eine Autorisierungslösung möglichst standard-konform ausgelegt werden. Gerade bei der Weiterentwicklung oder Erweiterung einer Autorisierungslösung, kann die Verwendung von Standards eine wichtige Grundlage darstellen, wenn neue Komponenten integriert werden sollen.
- **Flexible Spezialisierbarkeit der Anwendung:** Eine allgemein ausgelegte Autorisierungslösung kann ein breites Feld von Anwendungen unterstützen. Spezialanwendungen (wie z.B. policy-based network management) können ohne grossen Aufwand aus der allgemeinen Lösung abgeleitet werden. Im besten Fall müssen nur neue (detailliertere) Policies definiert werden.



## Flexibilität

- Zielunabhängigkeit: Eine flexible Autorisierungslösung sollte weitestgehend unabhängig sein von den zu schützenden Applikationen. Durch die Integration von Policy Enforcement Points in die Zielanwendungen, können beliebige Applikationen geschützt und auf Zugriff kontrolliert werden.
- Anpassbare Zugriffskontrolle: In einer unternehmenstauglichen Autorisierungslösung lassen sich Zugriffsberechtigungen beliebig fein auslegen und können auch kombiniert angewendet werden. So lassen sich beispielsweise grobe Zugriffsfiler mit feingliederten Berechtigungen gemeinsam anwenden. Dadurch beispielsweise können mehrschichtige Zugriffsstufen realisiert werden.

## Reduzierung der Kosten

- Verwaltungskosten: Zentrale Autorisierungslösungen verbessern die Verwaltbarkeit der Zugriffskontrolle. Die Verwaltung von Policies kann effektiver und auch anwendungsbezogener durchgeführt werden. Dies führt zu geringeren Verwaltungskosten und einer längeren Lebensdauer der Richtlinien, wodurch sich die Zugriffskontrolle besonders langfristig auf die Kosten auswirkt.
- Wartungskosten: Da die Anstellung von neuen Mitarbeitern nicht unbedingt auch Änderungen in den definierten Policies zur Folge haben, können die Kosten für Betreuung und Wartung einer Autorisierungslösung reduziert werden. Einem neuen Mitarbeiter eines Unternehmens, werden Attribute zugewiesen, die seine Funktion beschreiben. Diese Attribute sind meist schon vor der Einstellung des neuen Mitarbeiters definiert worden. Das Autorisierungssystem kennt dann bereits Policies, die gegen die neuen Zugriffsanfragen evaluiert werden können.
- Risikokosten: Durch die bessere Unterstützung bei der Definition von Zugriffsberechtigungen, entstehen auch weniger Fehler bei der Zuweisung von Rechten. Dies vermindert das Risiko von ungewollten Zugriffen und damit auch die Wahrscheinlichkeit von Schadensfällen. Eine verringerte Schadenswahrscheinlichkeit kann Kosten in der Risikoplanung vermindern und Reserven in Budgetplanung verfügbar machen.



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

HERAS<sup>AF</sup> ist ein Open Source Projekt in der Entstehungsphase. Initiiert wurde das Projekt im Januar 2006, nach ca. 8 Monaten Ideenentwicklung, Konzeption und Projektion. Die Idee zu HERAS<sup>AF</sup> ist eine Gemeinschaftsidee von Wolfgang Giersche, Yan Graf und mir.

### 4.1 Motivation

Zentrale Policyverwaltung und Zugriffssteuerung ist noch nicht weit verbreitet. Die Meisten kommerziellen und nicht gewerblichen Anwendungen kommen mit ihrem eigenen Benutzerspeicher und ihren Zugriffssteuerungsmechanismen daher. Quelloffene, standardisierte Lösungen gibt es noch keine.

Häufig führt diese Situation zu unvermeidlichen zusätzlichen Bemühungen, Kosten und Sicherheitsgefahren. Es kann sogar zu Inkonsistenzen kommen, in denen unterschiedliche Anwendungen, unterschiedliche Niveaus des Zugangs zur gleichen tatsächlichen Ressource gewähren. Dies kann auch geschehen, wenn eine schützenswerte Ressource einfach übersehen wird und somit nicht geschützt wird. Aber auch wenn Synchronisationen zwischen unterschiedlichen Anwendungen zu überflüssigen Policyinformationen führt.

Daher ist eine konsistente, zentrale Policyverwaltung und Zugriffssteuerung einer der ersten Grundsteine, wenn es darum geht Gefahr, Kosten und Bemühungen zu verringern.

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

### 4.2 Credo

Eine Autorisierungslösung sollte idealerweise...

- ...einen ganzheitlichen Ansatz des Autorisierungsprozesses verfolgen
- ...unternehmenstauglich sein
- ...Security für nahezu jegliche Applikation anbieten
- ...als Architektur-Framework funktionieren, welches sich leichtgängig in die existierende IT-Infrastruktur einfügen

HERAS<sup>AF</sup> nimmt folgende Herausforderungen an:

#### **Ganzheitlicher Ansatz**

- HERAS<sup>AF</sup> unterstützt den gesamten Autorisierungsprozess.
- Im technischen Sinne bedeutet dies, dass jegliche Zugriffe auf geschützte Ressourcen von verteilten Agenten (Policy Enforcement Points) unterbrochen und an einen Policy-Server (Policy Decision Point) umgeleitet werden, wo sie, basierend auf anwendbare Policies 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 nicht-technisches Personal im Frameworkdesign beachtet werden, speziell im Policy Administration Point.



## 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, welche HERAS<sup>AF</sup> verwendet, jederzeit garantiert. Eigene Komponenten können durch Integration der beschriebenen API realisiert werden.
- Die API von HERAS<sup>AF</sup> kann verwendet werden um unternehmensspezifische Einbindungen zu realisieren. Die spezifischen Komponenten brauchen lediglich die Erweiterungspunkte von HERAS<sup>AF</sup> zu benutzen. Beispielsweise können so Verbindungen zu ERP-Systemen gebaut 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, sofern diese verfügbar und angemessen sind. Somit 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 Policy Administration Point 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 Policies 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.



## 5 HERAS<sup>AF</sup> Logik

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

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

Abbildung 7 zeigt das Komponentenschema von HERAS<sup>AF</sup>. Die Beschreibung der einzelnen Komponenten findet sich in den folgenden Kapiteln.

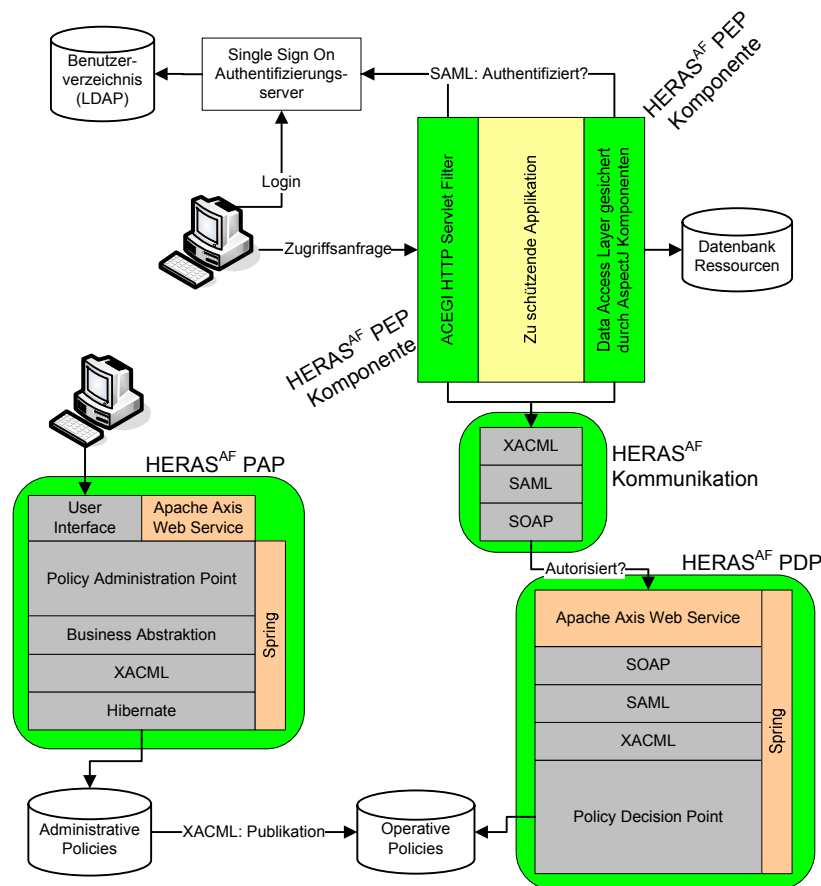


Abbildung 7: HERAS<sup>AF</sup> Komponentenschema

### 5.1 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 (engl. intercepting filters [ACM01]) oder der unterbrechenden Web-Agenten (engl. intercepting web agent [SNL05]) angewandt.



Applikationsspezifische PEP's können daher auf diverse Arten realisiert werden, weil verschiedene Unterbrechungsmöglichkeiten zur Verfügung stehen. Mehrere PEP's können auch gleichzeitig verwendet werden. Die Unterbrechung kann realisiert werden durch:

- **Filter-Technologien**

Speziell im Umfeld von Web-Technologien, wo Web-Applikationen in einem J2EE-Container ausgeführt werden, stehen diverse Filtertechnologien zur Verfügung. So können beispielsweise Servlet-Filter, Apache Module und spezialisierte Frameworks wie Acegi eingesetzt werden.

Die Zugriffe eines Benutzers werden anhand der erfragten Ressourcen gefiltert. Ein zur Anwendung kommender Filter kann weitere Logik realisieren und den Autorisierungsprozess auslösen.

Nach Erhalt der Autorisierungsentscheidung, kann der Filter seine Arbeit an der Unterbrochenen Stelle fortsetzen.

- **Aspektororientierte Programmierung**

Aspektororientierte Programmierung (engl. aspect oriented programming, AOP) ist ein Programmierparadigma, um räumlich getrennte Programmbestandteile von zentraler Stelle, mit bestimmten Eigenschaften auszustatten. Dazu werden so genannte Aspekte in eigenen Dateien definiert und zur Übersetzungszeit automatisch in den Programmcode eingefügt.

Agenten, die also AOP verwenden, können auf diese Weise nachträglich in existierende Applikationen hineinkompiliert werden.

Daher können beispielsweise zukünftige Applikationsentwickler beinahe vollständig auf die Realisierung von Autorisierungslogik verzichten, weil sich der Agent nachträglich einfügen lässt.

Auf diese Weise lässt sich eine Autorisierungslogik realisieren, welche nicht nur den Zugriff auf Objekte, sondern auch auf Methoden oder Datenfelder eines Objekts, kontrolliert. Dadurch lässt sich eine mehrfach verfeinerte Autorisierungslogik realisieren.

- **Spezifische Programmierung**

Die Aufgaben von Agenten lassen sich auch bereits in der Logik der zu schützenden Applikationen implementieren. Somit können Anwendungen auch wie bisher, bereits bei der Entwicklung, die Autorisierungsprozesse realisieren.

HERAS<sup>AF</sup>-PEP bietet die benötigte Logik und stellt Erweiterungspunkte zur Verfügung, an denen die spezifischen Implementationen ansetzen können. Zur Logik gehören vor allem die Erstellung von Autorisierungsanfragen, die Kommunikation mit dem PDP, sowie die kontextspezifische Interpretation einer Autorisierungsantwort.

## 5.2 Policy Decision Point

Der HERAS<sup>AF</sup>-PDP realisiert die komplette Logik der Zugriffsentscheidungsfindung. Dazu gehören das Auffinden von anwendbaren Policies, die Evaluierung einer Anfrage gegen die gefundenen Policies, wie auch die Kombination der erhaltenen Resultate. Für die Anreicherung von Policies mit fremdverwalteten Attributen, implementiert der PDP auch die Kommunikation mit dem PIP (Policy Information Point). Für die Auffindung von Richtlinien stellt er auch die Kommunikation mit dem Policy Store sicher.



Das Augenmerk beim PDP liegt vor allem in der Performanz. Die Abarbeitung einer Anfrage bis zum Senden der Antwort darf nicht so lange dauern, dass Benutzer einer geschützten Applikation Wartezeiten in Kauf nehmen müssen. Deshalb unterstützt oder verwendet der PDP leistungsfördernde Mechanismen wie Caching, indexierte Suche, Clustering und schnelle Vergleichsalgorithmen. Alle Komponenten, die Leistungskomponenten im Besonderen, lassen sich austauschen und somit auf Leistung und Funktionalität optimieren.

Durch die konsequente Verwendung von etablierten J2EE-Komponenten und die Verwendung von J2EE-Containern, werden keine zusätzlichen Kenntnisse von Wartungspersonal und Infrastrukturbetreuern benötigt. Clustering oder Performanzoptimierungen können auf der Ebene der Container angepasst werden.

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

### 5.3 Kommunikation PEP↔PDP

HERAS<sup>AF</sup> verwendet eine komplett standardisierte Form einer Entscheidungsanfrage über TCP/IP: Die Einbettung einer „XACML Authorization Decision Query“ in eine „SAML Assertion“ ist beschrieben in [SAMLXACML] und die Einbettung der SAML Assertion in [SAMLSOAP].

Gewählt wurden diese offenen, etablierten Standards weil sie sich in den Anwendungsbereichen der Sicherheit und Datenübermittlung offenbar bewährt haben, was man aus ihrer Popularität und Verbreitung schliessen kann.

Für die Übertragung von Autorisierungsanfragen und –antworten werden die Inhalte mehrfach gekapselt und verschachtelt. XACML-Nachrichten werden in SAML-Nachrichten verpackt, diese wiederum in SOAP-Nachrichten. Die Datenübermittlung wird über das Hypertext Transfer Protocol (http) bewerkstelligt.

Durch diese Verschachtelung wird sichergestellt, dass die Vorteile jeder Technologie genutzt werden können. Beispielsweise könnte man problemlos ein anderes Übertragungsprotokoll als http verwenden, weil SOAP die die Transportschicht weitgehend abstrahiert und sich fast beliebig austauschen lässt.

Durch die Einbindung in SAML, können Funktionen genutzt werden, die ausserhalb der Anwendung von HERAS<sup>AF</sup> liegen. Diese sind beispielsweise:

- Single Sign On
  - Ein Benutzer ist nach der Anmeldung an einer Webanwendung automatisch auch für die Benutzung von weiteren Anwendungen berechtigt.
- Verteilte Transaktionen
  - Mehrere Benutzer arbeiten gemeinsam an einer Transaktion und teilen sich die Sicherheitsinformationen
- Authentifizierungsdienste

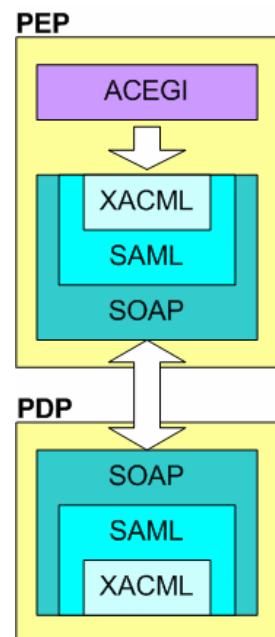


Abbildung 8:  
PEP↔PDP  
Kommunikation



## 5.4 Policy Information Point

PIP's sind verteilbare Einheiten, welche dem PDP zusätzliche Informationen zur Verfügung stellen. Beispielsweise können zusätzliche Attribute und Werte aus einer Personaldatenbank geholt werden, um die Information über ein Subjekt anzureichern.

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.

HERAS<sup>AF</sup> verzichtet auf eine eigenständige Komponente in Form eines Policy Information Point. Die Funktionalität der zusätzlichen Informationsbeschaffung wird stattdessen in den Policy Decision Point integriert. Im PDP können so genannte „AttributeFinder“ eingehängt werden, welche Legacy-Daten auffinden können. Diese stehen dann dem PDP für die Entscheidungsfindung zur Verfügung. Für die Integration von AttributeFinders bietet HERAS<sup>AF</sup> Erweiterungspunkte an, welche Schnittstellen definiert, um spezifische AttributeFinder zu realisieren.

## 5.5 Policy Store

HERAS<sup>AF</sup> ist in der Lage verschiedene Speicherorte für Richtlinien zu benutzen. Aus Performanzgründen ist es ratsam eine sehr schnelle, indexierbare Speichervariante zu verwenden. Daher empfiehlt es sich, die Policies in einem LDAP-Verzeichnis (Lightweight Directory Access Protocol), oder einer ähnlichen Applikation, zu halten.

HERAS<sup>AF</sup> beinhaltet einen Deployment-Manager, der die Logik definiert, wie Policies verteilt, publiziert und wieder gefunden werden. Der Deployment-Manager bieten Extensionspunkte an, um verschiedenste Speichervarianten von Policies zu realisieren.

## 5.6 Policy Administration Point

Der HERAS<sup>AF</sup>-PAP stellt die Funktionalität zur Verfügung, um Policies zu erstellen, zu bearbeiten und aktiv zu schalten. Zum PAP gehören:

- eine Datenbank, für die Speicherung der definierten Policies und Vorlagen
- der Deployment-Manager, um die Policies in den Policy Store zu publizieren
- die Logik und API, die von den Programmierern von Benutzeroberflächen verwendet werden kann
- eine Default-Implementierung eines GUIs, realisiert als Web-Interface mittels Java Server Faces

Die API und Logik beinhalten eine spezielle Abstraktionsebene, die es ermöglicht, nicht-technischem Personal, Policies in der Geschäftssprache zu formulieren.

Richtlinien in XACML zu beschreiben, kann für Personen, welche über wenig informatiktechnisches Wissen verfügen, zu einer grossen Herausforderung werden. Beispielsweise kennt sich nicht jeder Prozessmanager einer Unternehmung mit den technischen Begriffen (z.B. String, URI, rfc822-Name usw.) aus, die in einer XACML-Policy verwendet werden. Üblicherweise kennen Prozessmanager auch den technischen Aufbau der Applikationen und Ressourcen nicht.



Die betroffene Person könnte die Richtliniendefinition (und alles Dazugehörige) zwar erlernen, doch dies ist im besten Fall mit Kosten für die Ausbildung verbunden. In ungünstigeren Fällen können langfristig enorme Kosten entstehen, weil die Person vielleicht nicht effektiv arbeiten kann oder grobe Fehler bei der Policybeschreibung machen könnte. Um dieses Problem in der Policyverwaltung zu vermeiden, müsste ein Prozessmanager eng mit Technikern zusammenarbeiten, weil Kompetenzen aus beiden Tätigkeitsbereichen nötig sind.

In der Softwareentwicklung wird seit mehreren Jahren das Prinzip „Separation of Concerns“ verfolgt, wonach verschiedene Aufgabenbereiche einer Anwendung in eigene Teile separiert werden. Dies erhöht die Austauschbarkeit von Systemteilen und damit die Wartbarkeit einer Applikation. Letztlich bedeutet das einen verbesserten Investitionsschutz. Dieses Prinzip wurde innerhalb HERAS<sup>AF</sup> nicht nur im technischen Sinne aufgenommen, sondern zusätzlich auf Geschäftsprozesse und Unternehmensbelange ausgeweitet („Separation of Business Concerns“). Die Trennung der Kompetenzen innerhalb eines Business Prozesses führen dabei zu gesteigerter Effektivität und damit zu Kostenreduktionen.

Im Policy Administration Point ist „Separation of Business Concerns“ einer der wichtigsten Aspekte für die unternehmenstaugliche Anwendbarkeit. Dort steht die Trennung der Kompetenzen von Technikern und in den Geschäftsprozess involvierten Personen im Mittelpunkt. Realisiert wird die Trennung durch die Implementierung eines Vorlagenmechanismus, auf dem zwei verschiedenen Bedienungsansichten (UI Views) aufgebaut werden.

Eine Ansicht (Technical View) wird für die Definition von Policy-Vorlagen verwendet. Ein Techniker kann mittels dieser Ansicht Vorlagen für alle Policy-Elemente entwickeln und mit geschäftsspezifischen Metadaten anreichern. Ein Techniker kann die Vorlagen bereits mit technischen Details füllen. Des Weiteren kann er die Felder bestimmen, die ein Nicht-Techniker nachträglich ausfüllen kann. Zu allen Vorlagenfeldern kann er zusätzlich geschäftsspezifische Namen / Ausdrücke, Beschreibungen und Standardwerte erfassen.

In der zweiten Ansicht (Business View) kann beispielsweise ein Prozessmanager die Vorlagen ausfüllen und so neue Richtlinien erstellen. Dabei stehen ihm die geschäftsspezifischen Informationen (Metadaten) über die auszufüllenden Felder zur Verfügung. So können Richtlinien erstellt werden, ohne über weit reichende technische Kenntnisse verfügen zu müssen. Ein Nicht-Techniker braucht damit also nicht zu wissen, was eine URL oder Klassenmethode ist. Der PAP bietet die Möglichkeit, Policies anhand von Vorlagen und selbst wählbarer Nomenklatur zu entwickeln.



## 6 HERAS<sup>AF</sup> Technik

HERAS<sup>AF</sup> ist als Multiprojekt-Architektur realisiert, um die Abhängigkeiten innerhalb der Module und Komponenten minimal zu halten. HERAS<sup>AF</sup> ist vollständig in Java programmiert und nutzt Funktionen der Java Version 5.0.

### 6.1 Architektur

HERAS<sup>AF</sup> orientiert sich an den Anforderungen und Vorteilen eines objektorientierten Applikationsframeworks [Fayed97]. Die Architektur und das Design des Frameworks sind konsequent darauf ausgelegt, höchstmögliche Modularität, Wiederverwendbarkeit, Skalierbarkeit und Entkopplung mittels „Inversion of Control“ zur Verfügung zu stellen.

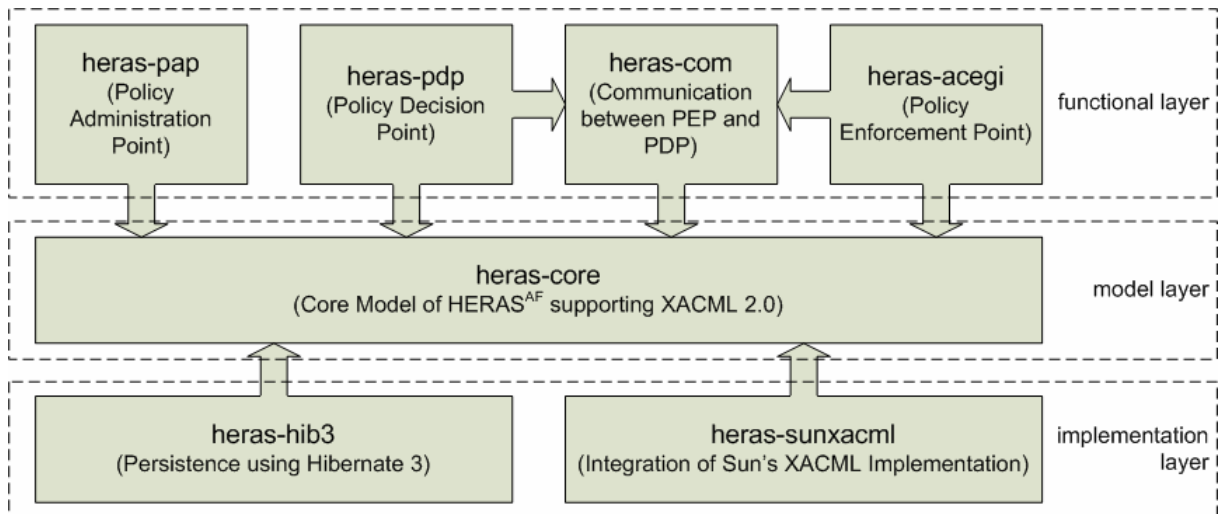
Objektorientierte Designprinzipien und Patterns werden in HERAS<sup>AF</sup> durchgehend angewendet und stellen tragende Pfeiler für der Architektur dar. So wurde beispielsweise die „Programmierung gegen Interfaces“ so ausgeprägt wie möglich umgesetzt, damit die Trennung von Schnittstellen und Implementationen sichergestellt sind.

Die Modularität und Wiederverwendbarkeit wurde durch die Aufteilung der Gesamtfunktionalität in einzeln verwendbare Module sichergestellt. Die Module und deren Design wurden anhand der Methodik der „separation of concerns“ (aufteilen der Modelle hinsichtlich der Anforderungen) vorgenommen. In HERAS<sup>AF</sup> werden Funktionen und Aufgaben die unterschiedliche Belange erfüllen, in Komponenten und Modulen separiert. Dadurch wird erreicht, dass beispielsweise die Persistenzkomponente und die XACML-Implementation voneinander völlig unabhängig sind.

Zusätzlich konnte durch Verwendung des Spring IoC-Containers bewerkstelligt werden, dass zwischen den funktionalen Modulen und den konkreten, technologieabhängigen Implementationen keine direkten Abhängigkeiten bestehen. Dies erlaubt es, beispielsweise die Persistenzkomponente heras-hib3 durch eine andere zu ersetzen, ohne dass andere Module programmtechnisch angepasst werden müssen. Einzig die Konfiguration der benutzenden Komponenten muss an die neue Implementation angepasst werden.



Abbildung 9 zeigt die Abhängigkeiten der Module von HERAS<sup>AF</sup>. Die Pfeile zeigen die Abhängigkeiten auf. Zum Beispiel: „Modul heras-hib benötigt heras-core“.



**Abbildung 9: HERAS<sup>AF</sup> Module und Abhängigkeiten**

Die Basis des Frameworks bildet der heras-core. Alle Schnittstellen für die Verwendung und Realisierung von HERAS<sup>AF</sup> sind innerhalb dieses Moduls definiert. Auf diese Art werden auch die Erweiterungspunkte für konkrete Implementationen definiert. An diesen Erweiterungspunkten werden die Persistenz- und der Adapter der XACML-Implementation mittels Spring Inversion of Control eingehängt. Die Funktionalen Module (heras-pap, heras-pdp usw.) verwenden den heras-core und bauen ihre Funktionalität darauf auf.



## 6.2 Kern (heras-core, heras-hib3, heras-sunxacml)

Der Kern von HERAS<sup>AF</sup> kann schematisch mit drei logischen Schichten dargestellt werden. Die Modellschicht stellt alle Interfaces und Erweiterungspunkte für andere Module zur Verfügung. Die Implementationsschicht realisiert die Datenstrukturen und Logik von HERAS<sup>AF</sup>. Die Klassen und Schnittstellen innerhalb dieser Schicht werden nicht von anderen Modulen verwendet. Die Module für Persistenz- und XACML-Implementierung sind in der Erweiterungsschicht des Kerns zusammengefasst. Die direkten Abhängigkeiten innerhalb des Kerns zu Bibliotheken von Drittherstellern werden nicht als logische Schicht angesehen.

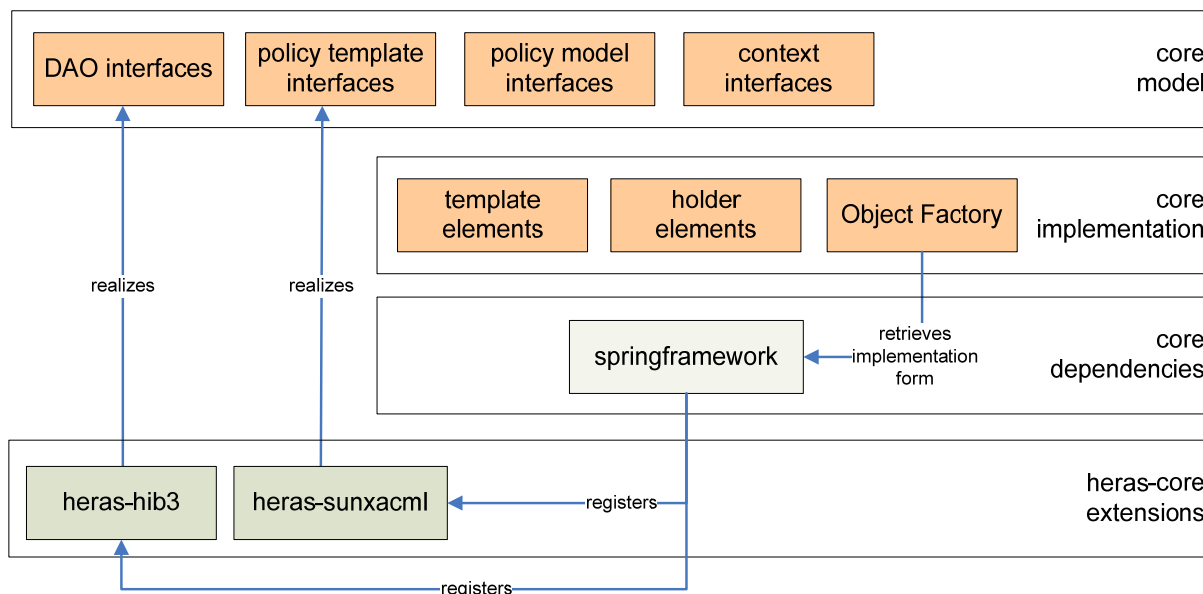


Abbildung 10: Schema des Kerns

Das Modul heras-hib3 realisiert die Schnittstellen für die Persistenzimplementierung und nutzt die persistenzspezifischen Erweiterungspunkte des Kerns. heras-core verwendet in seiner Logik ausschliesslich die abstrakten Schnittstellen. Deshalb ist der Kern von dieser Implementierung unabhängig. Die Persistenz kann daher ohne weiteres ausgetauscht werden.

Genauso wird mit der XACML-Implementierung verfahren mit dem Unterschied, dass heras-sunxacml andere Schnittstellen realisiert bzw. Erweiterungspunkte verwendet.

Die einzige direkte Abhängigkeit von heras-core zu externen Bibliotheken, ist diejenige zum springframework. Der Spring IoC-Container hängt die externen Module im Kern ein. Um den Datenzugriff auf gespeicherte Objekte sicherzustellen, wird die Persistenzimplementierung eingehängt. Um die korrekte Erstellung von neuen Objekten zu gewährleisten, wird die XACML-Implementierung eingehängt. heras-core kann somit neue Objekte herstellen ohne deren konkrete Implementation zu kennen.

Die Funktionalen Module (wie heras-pap, heras-pdp, usw.) sind somit auch nicht auf eine direkte Verwendung der Implementierungsmodule angewiesen. Alle Datenzugriffsfunktionen können via heras-core verwendet werden. Jegliche innerhalb HERAS<sup>AF</sup> verwendete Objekte (gespeichert oder ungespeichert), können somit immer durch den heras-core verwaltet werden, egal welche Implementierungen verwendet werden.

Die Module heras-core, heras-hib3 und heras-sunxacml werden in drei einzelnen Bibliotheken (JAR-Dateien) verteilt.



### 6.3 Policy Enforcement Point (heras-acegi, heras-com)

Im Rahmen des Prototyp von HERAS<sup>AF</sup> wurde eine J2EE-PEP-Komponente (heras-acegi) entwickelt. Diese wurde mittels Acegi Security realisiert. Acegi Security beinhaltet unter anderem einen „Inteceptor Concept“, welches in J2EE-Containern eingesetzt werden kann. Das Modul heras-acegi kann daher in beliebige J2EE-Applikationen als ServletFilter integriert werden.

Der Policy Enforcement Point war nicht teil dieser Diplomarbeit. Detaillierte Informationen und weitere Auskünfte erteilt das HERAS<sup>AF</sup>-Projektteam.

### 6.4 Policy Decision Point (heras-pdp, heras-com)

Geplant ist der PDP als eigenständige J2EE-Anwendung, die auf einem oder mehreren Applikationsservern läuft. Das Design sieht eine einzeln verteilbare PDP-Komponente vor, welcher verschiedene Apache Axis Webservices zur Verfügung stellt. Der Kunde dieser PDP-Webservices ist ein Policy Administration Point. Dieser verwendet den Service um Policies zu publizieren und zu verwalten. Die Policy Enforcement Points kommunizieren ebenfalls via Webservice mit dem PDP. Dabei werden Sicherheitsanfragen bzw. -antworten ausgetauscht.

Im Rahmen dieses Prototyps von HERAS<sup>AF</sup> wird der Policy Decision Point nicht als eigenständig verteilbare Einheit realisiert, weil die Realisierung einer nicht-proprietären Kommunikation zwischen PAP und PDP den Rahmen dieser Prototypentwicklung gesprengt hätte. Das Modul heras-pdp wird daher zum Endzeitpunkt dieser Arbeit noch nicht realisiert worden sein.

Daher ist in diesem Prototyp der PDP im Policy Administration Point integriert. Dadurch kann die Kommunikation intern gelöst werden, weil der PAP und der PDP in derselben virtuellen Maschine laufen. (Siehe Abschnitt 6.7 für die technische Beschreibung des PDPs).

### 6.5 Policy Information Point

Ein Policy Information Point ist in HERAS<sup>AF</sup> nicht vorgesehen. Die zusätzliche Funktionalität der Auffindung und Berücksichtigung von extern verwalteten Attributen und Informationen wird innerhalb des Policy Enforcement Points gewährleistet. Im PEP können diverse Attributfinder (engl. attribute finder) eingehängt werden. Diese können zusätzliche Werte auffinden, auswerten und damit die Zugriffsanfrage anreichern. Damit entfällt die Notwendigkeit eines separat verteilbaren PIPs.

### 6.6 Policy Store

Der Policy Store von HERAS<sup>AF</sup> wurde im Rahmen des Prototyps nicht integriert. Aus Performanzgründen des Policy Decision Point müssen Policies sehr schnell im Store gefunden werden können. Daher ist es unumgänglich, dass Policies sinnvoll indiziert und einfach aufgefunden werden können. Da ein Policy Store auch mehrere PDPs bedienen kann, sind auch Caching, Load Balancing und Clustering sehr wichtige Anforderungen.



Eine erste vorgesehene Implementierung eines Policy Stores von HERAS<sup>AF</sup>, ist die Verwendung eines LDAP-Directories. Policies können in einem LDAP-Baum nach verschiedenen Kriterien indexiert werden. Ausserdem stellt LDAP bereits umfangreiche Funktionen für Caching, Load Balancing und Clustering zur Verfügung. Diese lassen sich LDAP-seitig verwenden, was dazu führt, dass nur die Publikation der Policies entwickelt werden muss. Die Herausforderung bei der Realisierung des Publikationsmechanismus liegt darin, die Policies aufgrund ihrer Zusammensetzung im LDAP-Baum zu indexieren. Es bietet sich an, Policies mehrfach zu indexieren, anhand passender Targets einer Zugriffsanfrage. Die Targets einer Policy, oder deren Regel können beispielsweise in Bezug auf folgende Target-Elemente passen:

- Subjekt(e) passend anhand...
  - ...Subjekt-ID (Benutzername, Rolle, Gruppe usw.)
  - ...Subjekt-Beziehungen (Abteilung, Mitarbeiterhierarchien, im Auftrag von, Auftraggeber usw.)
  - ...sonstige Subjekt-Attribute
- Ressource(n) passen anhand...
  - ...Resource-ID (Ressourcenbezeichner)
  - ...Ressource-Struktur (Teildokument einer Akte)
  - ...Ressourcen-Ort (Server, Verzeichnis, Unterverzeichnis usw.)

Daran zeigt sich, dass die die Definition der LDAP-Struktur und das Design des Publikationsmechanismus' einen erheblichen Zeitaufwand in Anspruch nimmt. Während der Entwicklung des HERAS<sup>AF</sup>-Prototyps stand diese Zeit nicht zu Verfügung. Deshalb wird in einer ersten Realisierung der Policy Store im Hauptspeicher des Policy Decision Points gehalten. Durch die schnellen Zugriffsmöglichkeiten im RAM kann in einer ersten Phase auf eine komplexe Indexierung der Policies verzichtet werden.



## 6.7 Policy Administration Point (heras-pap)

Der PAP von HERAS<sup>AF</sup> wurde einem J2EE-Projekt realisiert. In der ersten Version des Prototyps integriert das PAP-Modul (heras-pap) die Komponenten des PDP und PAP. Das Modul wird als eigenständige J2EE-Anwendung (WAR-File) verteilt.

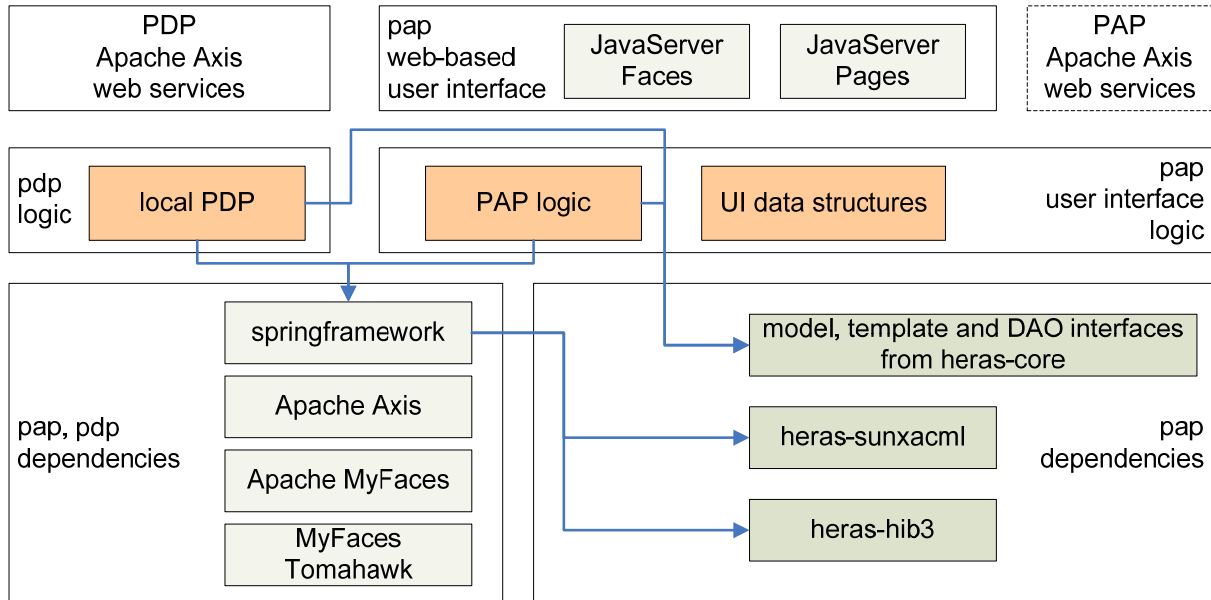


Abbildung 11: Schema der Policy Administration Points

Für die Bedienung des PAPs wurde eine Webbenutzerschnittstelle mittels JavaServer Faces und JavaServer Pages realisiert. Für die Zukunft ist auch Apache Axis Webservices vorgesehen, welche die Bedienung des PAPs auch für andere Benutzeroberflächen ermöglichen. Beispielsweise könnte eine Eclipse RCP Anwendung entwickelt werden, welche die Webservices ansteuert.

Der Policy Decision Point bietet Webservices (realisiert mit Apache Axis) für die Zugriffsanfrage. Intern verwendet der HERAS<sup>AF</sup>-PDP die PDP-Implementierung von Sun's XACML Implementation [SunXACML]. Publierte Policies werden dort im RAM des PDPs gehalten und nicht in einem externen Policy Store gespeichert.

Beide Komponenten (PDP, PAP) verwenden das Modul heras-core direkt. Der PAP verwendet das Modul heras-sunxacml indirekt (IoC).

## 6.8 Verwendete Frameworks und Technologien

Im Prototyp von HERAS<sup>AF</sup> wurden ausschliesslich quelloffene Frameworks, Technologien und Bibliotheken verwendet. Zudem wurden für die Entwicklung nur Werkzeuge eingesetzt, welche OpenSource zur Verfügung stehen.

Die Komponenten und Werkzeuge wurden anhand folgender Kriterien ausgewählt:

- Vollständige Kriterienerfüllung der Open-Source-Definition [OpenSource]
- Kontinuierliche Weiterentwicklung
- Möglichst grosse Entwickleranzahl
- Aktive Anwendergemeinschaft



- Langer Weiterbestand und gute Zukunftsaussicht
- Schnelle Hilfe bei Problemen verfügbar
- Multi-Plattform-Lauffähigkeit (Microsoft Windows, Linux, Unix usw.)

### 6.8.1 Entwicklungstools

HERAS<sup>AF</sup> wird vollumfänglich mit der Entwicklungsumgebung Eclipse 3.1 (<http://www.eclipse.org>) entwickelt. Eclipse ist eine Open-Source Entwicklungsumgebung (nicht nur) für Java, deren Entwicklung ursprünglich von IBM gesponsert wurde. Sie ist modular, elegant, mächtig, erweiterungsfähig und in kurzer Zeit sehr populär geworden.

Zusätzlich wurde Maven 1.0.2 (<http://maven.apache.org>) für das Build- und Deploy-Management eingesetzt. Maven ist eine Weiterentwicklung des Apache-Ant-Projekts, welches auch weiterhin den Kern von Maven ausmacht. Speziell vereinfacht Maven das Handling von Abhängigkeiten, von benötigten Artefakten und von Multiprojekten. Das Build-Management-Tool basiert auf einer Plugin-Architektur, die es ermöglicht, Plugins für verschiedenste Anwendungen (compile, test, build, deploy, checkstyle, pmd, scp-transfer) auf das Projekt anzuwenden, ohne diese explizit installieren zu müssen.

Um HERAS<sup>AF</sup> zu testen, wurde TestNG 4.7 (<http://testng.org>) verwendet. TestNG ist ein Framework zum Testen von Software. Es hat grosse Ähnlichkeit mit JUnit, bietet aber einen grösseren Funktionsumfang. TestNG ist so entworfen, dass es all Kategorien von Tests unterstützt (unit, functional, end-to-end, integration). Diese wurden auch in HERAS<sup>AF</sup> getestet.

### 6.8.2 Sun's XACML Implementation

Sun's XACML Implementation (<http://sunxacml.sourceforge.net>) ist eine quelloffene Implementierung des OASIS XACML Standards und in Java<sup>TM</sup> programmiert.

Das Projekt wurde in den Labors von Sun Microsystems entwickelt und ist Teil eines weitergehenden Projektes zu Thema Internet Autorisierung in der Internet Security Research Group.

Das Projekt unterstützt alle pflichtigen und einige optionale Funktionen des XACML Standards. Insbesondere das Parsen von Richtlinien und Anfrage- bzw. Antwort-Dokumenten wird unterstützt. Auch die Logik bezüglich Anwendbarkeit einer Richtlinie und Evaluierung von Anfragen gegen Richtlinien stellt die Implementierung zur Verfügung. Hinzu kommen alle Standard-Attribut-Typen, Funktionen und Kombinalgorithmen, welche realisiert sind.

HERAS<sup>AF</sup> verwendet Sun's XACML Implementation in der Version 2.0. Zum Zeitpunkt der Arbeit war diese Version noch im Entwicklungsstadium und noch nicht offiziell veröffentlicht. Trotzdem wurde der aktuelle, unveröffentlichte Code verwendet, weil nur dieser die Spezifikation von XACML 2.0 unterstützt.

### 6.8.3 OpenSAML

OpenSAML (<http://www.opensaml.org>) ist eine Zusammenstellung von OpenSource-Bibliotheken geschrieben in Java und C++. Diese können benutzt werden, um SAML-Nachrichten zu erstellen, zu übertragen oder zu parsen.



OpenSAML kann die individuellen Felder, welche zu einer SAML-Nachricht gehören speichern, die korrekte XML-Repräsentation erstellen und umgekehrt. OpenSAML unterstützt die Einbettung von SAML in SOAP für den Austausch von Anfragen und Antworten. Ausserdem stellen die Bibliotheken zusätzliche Hilfen, für die Unterstützung des SAML browser/POST Profils für einmalige Web-Anmeldung (web single sign on), zur Verfügung.

OpenSAML wurde von Mitgliedern der Internet2-Gruppe (<http://www.internet2.edu>), als Teil des Shibboleth Projekts (<http://shibboleth.internet2.edu>) entwickelt.

HERAS<sup>AF</sup> verwendet OpenSAML in der Version 2.0. Zum Zeitpunkt der Arbeit war diese Version noch im Entwicklungsstadium und noch nicht offiziell veröffentlicht. Trotzdem wurde der aktuelle, unveröffentlichte Code verwendet, weil nur dieser die Spezifikation des SAML-XACML Profile 2.0 unterstützt.

### 6.8.4 Spring Framework

Spring [Spring] ist ein OpenSource Applikationsframework mit der Intention J2EE Entwicklung zu vereinfachen und gute Programmierpraktiken zu fördern [John05]. Es besteht aus einem IoC-Container, einem Framework für die Konfiguration von Komponenten und einer Menge von modularen Diensten für Transaktionen, Persistenz und Web-Schnittstellen.

Das Spring Framework stellt einen konsistenten Mechanismus zur Verfügung, um Geschäftsobjekte in einer J2EE-Umgebung zu verwalten, ohne dabei den in Applikationen häufig vorkommenden ad-hoc Mix aus Singletons und Factories aufkommen zu lassen [Seuf05]. Der zentrale Bestandteil der Spring Architektur, ist dessen leichtgewichtiger Inversion-of-Control-Container, welcher auf einer JavaBeans Infrastruktur basiert. Das Konzept des IoC-Containers fördert gute OO-Programmierpraktiken, wie z.B. das Programmieren gegen Interfaces, wodurch ein höherer Grad an Wiederverwendbarkeit der Geschäftsobjekte erzielt wird.

HERAS<sup>AF</sup> verwendet Spring in der Version 1.2.7, welches zum Startzeitpunkt der Diplomarbeit das aktuellste Release war.

### 6.8.5 Hibernate 3 (ORM)

Hibernate (<http://www.hibernate.org>) ist ein leistungsfähiger, objektrelationaler Mapper (ORM), mit dem sich Objekte in relationale Datenbankmanagement-Systeme (RDBMS) speichern und wieder laden lassen.

Durch Hibernate lassen sich Klassen nach dem objekt-orientierten Ansatz persistieren. Dazu gehören Assoziationen, Vererbung, Polymorphie, Kompositionen, und Kollektionen. Zusätzlich bietet Hibernate die Möglichkeit, Abfragebefehle in einer SQL ähnlichen Abfragesprache (HQL), nativem SQL oder in objekt-orientierten Kriterien zu formulieren.

Hibernate wird unter der LGPL OpenSource Lizenz publiziert. Die Software ist somit vollständig quelloffen und darf auch in kommerziellen Projekten eingesetzt werden.

Hibernate ist ein professionelles Open Source Projekt und eine mittlerweile feste Komponente des JBoss Enterprise Middleware Systems (JEMS).

HERAS<sup>AF</sup> verwendet Hibernate in der Version 3.1.3, welches zum Startzeitpunkt der Diplomarbeit das aktuellste Release war.



### 6.8.6 Acegi Security

Acegi Security (<http://www.acegisecurity.org>) ist ein quelloffenes Projekt, welches umfangreiche Authentisierungs- und Autorisierungsdienste für Unternehmensanwendungen zur Verfügung stellt, welche auf dem Spring Framework basieren. Acegi kann mit einer Vielzahl von Authentisierungsanbietern verwendet werden und kann Webanfragen und Methodenaufrufe autorisieren.

Acegi liefert einen integrierten Sicherheitsansatz über die verschiedenen Ziele und bietet auch Zugangskontrolllisten (ACL) an, um einzelne Objekte zu sichern.

Auf dem Implementierungsniveau wird Acegi durch Spring's Inversion of Control gehandhabt. Die Sicherheit wird durch Abfangen der Anfragen durch Servletfilter und Java AOP Frameworks gewährleistet.

HERAS<sup>AF</sup> verwendet Acegi Security in der Version 0.9, welches zum Startzeitpunkt der Diplomarbeit das offizielle Release war.

### 6.8.7 Tomcat 5.5

Tomcat 5.5 (<http://tomcat.apache.org>) ist ein leichtgewichtiger Java Servlet Container. Er ist die Referenz Implementation der Java Servlet und JavaServer Pages. Tomcat 5.5 wurde von der Jakarta Projektgruppe entwickelt. Diese wird von der Apache Software Foundation unterstützt.

HERAS<sup>AF</sup> verwendet Apache Tomcat in der Version 5.5.

### 6.8.8 Apache Axis (Apache eXtensible Interaction System)

Axis (<http://ws.apache.org/axis>) ist eine SOAP Engine – ein Framework für die Konstruktion von SOAP Prozessoren, Clients, Servern, Gateways usw. AXIS ermöglicht es, sowohl die Server- als auch die Client-Seite eines Web Services sehr einfach zu implementieren.

Innerhalb HERAS<sup>AF</sup> wird Apache Axis im Policy Decision Point, sowie bei der Kommunikation zwischen PDP und Policy Enforcement Point eingesetzt.

HERAS<sup>AF</sup> verwendet Apache Axis in der Version 1.2.1.



## 7 HERAS<sup>AF</sup> Implementierung

Die Framework-Architektur von HERAS<sup>AF</sup> stellt hohe Anforderungen an das Design und die Implementierung der Komponenten und Module. Stabilität, Flexibilität und Anwendbarkeit sind wichtige Ausprägungen für ein gutes Framework. Das Design von HERAS<sup>AF</sup> orientiert sich an diesen Eigenschaften. Um die Designqualität zu messen, wurde während der Entwicklung des Prototyps der Ansatz von Robert Martin „OO Design Quality Metrics“ [Martin94] angewendet. Für das Frameworkdesign wurden daher, neben den allgemein bekannten Softwaredesign-Anforderungen, folgende Ziele definiert:

- Minimale Abhängigkeiten zwischen Paketen
- Programmierung gegen konkrete Klassen ist nur innerhalb von Strukturen erlaubt, die sowieso voneinander abhängig sein müssen (z.B. paketintern). Ausserhalb wird ausschliesslich gegen Interfaces programmiert.
- Erzeugen von Instanzen mit dem Schlüsselwort „new“ ist nur paketintern erlaubt. Ausserhalb wird der Spring IoC-Container für die Erzeugung verwendet.
- Zyklische Abhängigkeiten zwischen Paketen sind nicht erlaubt.
- Ein Paket darf nur einen Nutzungs- oder Erweiterungspunkt darstellen. Einzige Ausnahme ist das Paket, welches das Struktur-Modell repräsentiert. Dieses Paket darf von allen anderen Modulen verwendet werden.
- Klassen werden im POJO-Stil programmiert

Die oben beschriebenen Regeln bilden die Grundlage für die Weiterentwicklung und Wartung von HERAS<sup>AF</sup>. Durch die gezielte Anwendung des Abhängigkeitseinfügings-Musters (engl. Dependency Injection Patterns [Fowler04]), welches eine Anwendung des Kontrollumkehrungsprinzips (engl. Inversion of Control Principle) ist, konnten die Regeln sehr komfortabel eingehalten werden. Daraus resultieren folgende positive Eigenschaften:

- Änderungen in den Implementierungen der Schnittstellen wirken sich nur auf einzelne, klar verfolgbare Programmstrukturen aus.
- Das Modell (org.smurve.heras.model in heras-core) von HERAS<sup>AF</sup> gibt die Struktur und die Funktionen vor, welche für die Konformität zu XACML 2.0 und die Geschäftsabstraktion nötig sind. Die konkrete XACML-Implementierung kann jederzeit ausgetauscht bzw. integriert werden, ohne dass Änderungen an anderen Modulen vorgenommen werden müssen.
- Das Persistenzpaket (org.smurve.heras.persistence in heras-core) gibt die Schnittstelle der Persistenz vor, welches von einer beliebigen Persistenzlösung implementiert werden kann. Somit kann auch das Persistenzmodul, ohne Änderungen an anderen Modulen, beliebig ausgetauscht werden.
- Persistenz-Modul und XACML-Implementierung benötigen keinerlei Kenntnis voneinander. Anpassungen und Änderungen wirken sich also nur innerhalb der Module aus.
- Bereits installierte Lösungen von HERAS<sup>AF</sup> können gewartet und verbessert werden, indem weiterentwickelte Module nachträglich ausgeliefert werden. Dies bedeutet, dass die bestehende Lösung nicht programmatisch angepasst werden muss. Die verwendeten Module können ausgetauscht und in der Lösung eingehängt werden.



Diese Diplomarbeit hatte die Aufgabe den Kern von HERAS<sup>AF</sup> zu realisieren. Zu diesem Kern gehören die Aufgaben des HERAS<sup>AF</sup>-Modells, dessen Implementierung und Persistenz. Für diese drei Hauptaufgaben wurde je ein eigenständiges Javaprojekt (Modul) definiert.

Für die erste XACML-Implementierung von HERAS<sup>AF</sup> wurde, aus bereits erwähnten Gründen, die „Sun’s XACML Implementation 2.0“ [SunXACML] gewählt. Die erste Persistenz wurde mittels „Hibernate 3.0“ realisiert. Dementsprechend wurden die Module auch benannt.

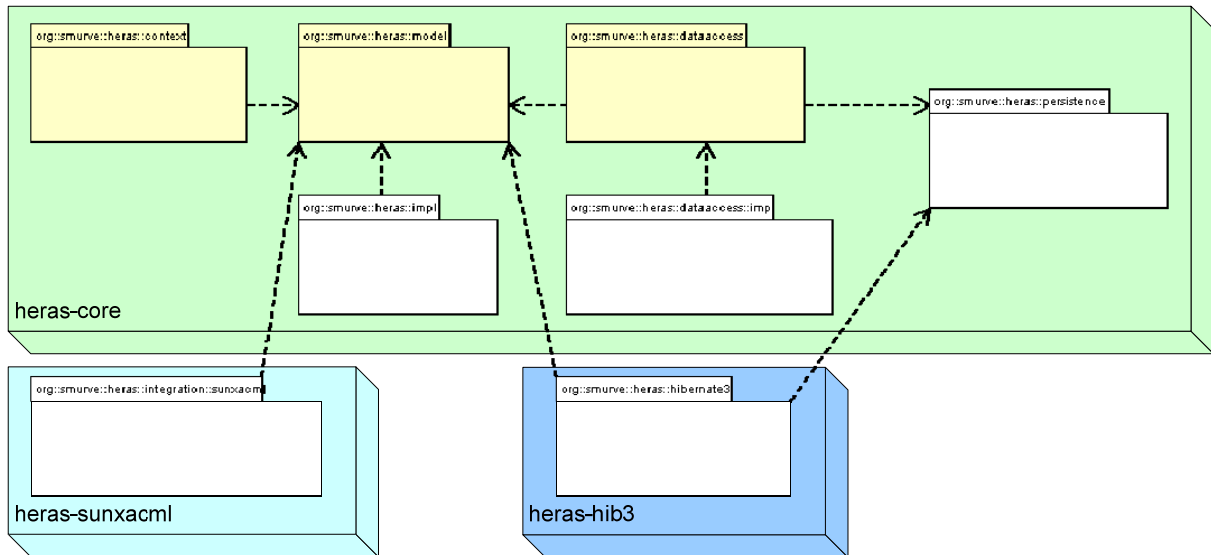


Abbildung 12: HERAS<sup>AF</sup> Kernmodule

- **heras-core**

Dieses Modul beinhaltet das Strukturmodell, die Kernfunktionalität des Modells, die HERAS<sup>AF</sup>-Logik und dessen Implementierung. Es ist das eigentliche Zentrum der Architektur und wird von allen anderen Modulen verwendet. heras-core hat keinerlei Abhängigkeiten zu anderen Modulen.

- **heras-sunxacml**

Dieses Modul adaptiert die „Sun’s XACML Implementation 2.0“ [SunXACML]. Aus heras-core wird nur das Paket org.smurve.heras.model benötigt. Es gibt die Strukturen und deren Schnittstellen vor, welche an passende Strukturen der XACML-Implementierung adaptiert werden müssen.

- **heras-hib3**

Dieses Modul hat zwei Abhängigkeiten zu heras-core. Das Paket org.smurve.heras.persistence gibt die Schnittstellen der Persistenzsteuerungsobjekte (engl. Data Access Objects) vor. Diese werden in heras-hib3 implementiert und führen Speicheroperationen der Modellobjekte aus. Deshalb besteht auch eine Abhängigkeit zum Paket org.smurve.heras.model.



## 7.1 heras-core

Das Modul wurde bewusst so schlank wie möglich gehalten, damit möglichst viel Funktionalität ausgegliedert werden kann. Dies bringt den Vorteil, dass viel Logik von bestehenden Modulen und Bibliotheken verwendet werden kann. So ist es nicht nötig, die Logik einer XACML-Implementation erneut zu entwickeln (wie beispielsweise Evaluierungsfunktionen von Policies). Diese Anwendungen wurden bereits von anderen Entwicklern gebaut und stehen teilweise quelloffen zur Verfügung.

heras-core abstrahiert die existierenden Lösungen und erweitert diese Abstraktion um zusätzliche Funktionen.

- Zu der Abstraktion gehört ein eigenes Strukturmodell, welches aus Interfaces aufgebaut ist und der XACML-Spezifikation 2.0 entspricht.
- Das abstrakte Modell wird innerhalb heras-core implementiert, was die funktionale Trennung der Persistenz von der XACML-Implementation ermöglicht. Damit können die Modellobjekte mit den von HERAS<sup>AF</sup> angereicherten Daten gespeichert werden. Die effektive XACML-Implementation bleibt von der heras-core internen Datenstruktur unabhängig und kann ihre eigene Repräsentation verwenden.
- Persistenzoperationen werden abstrahiert und um zusätzliche Logik ergänzt. So können Module, die heras-core verwenden, nicht nur Objekte speichern und laden, sondern auch erzeugen. Dies zentralisiert den Umgang mit Modellobjekten in einem einzigen Paket.
- Persistenzimplementierungen nutzen die Abstraktion ohne Zusatzfunktionen, um lediglich die Speicheroperationen zu unterstützen. Dies vereinfacht die Integration eines Persistenzmoduls.
- Das Modell von HERAS<sup>AF</sup> wurde in Bezug auf Richtlinienstrukturen und Zugriffsnachrichtenaustausch getrennt. Je ein Paket steht für die Verwaltung von Policies und die Verwaltung von Zugriffsnachrichten zur Verfügung.

### 7.1.1 Package-Design

In heras-core sind vier hauptsächlich abstrakte Pakete enthalten. Diese stellen auch die Erweiterungspunkte zu anderen Modulen zur Verfügung. Zwei konkrete Pakete realisieren die Logik von HERAS<sup>AF</sup>.

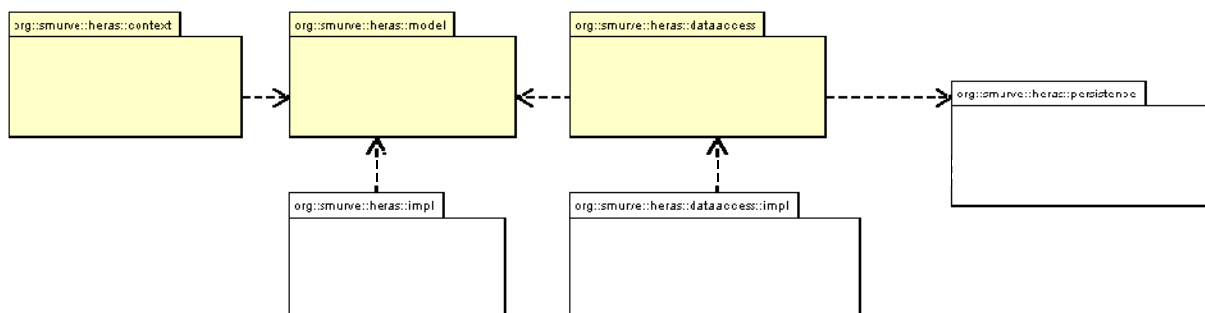


Abbildung 13: heras-core Paketstruktur

**org.smurve.heras.model** (kurz: o.s.h.model)

In diesem Paket befinden sich die Interfaces der XACML-Struktur (z.B. Policy, Rule, Target, usw.) und der Elemente, die für den Vorlagenmechanismus benötigt werden (PolicyTemplate, RuleTemplate usw.). Das Paket ist sehr abstrakt gehalten und stellt das „main interface“ zu HERAS<sup>AF</sup> dar.

Ein Policy Decision Point arbeitet ausschliesslich mit den Interfaces für die XACML-Strukturen. Ein Policy Administration Point verwendet die Template-Interfaces, um die Vorlagenfunktionen zu nutzen.

Eine detailliertere Beschreibung findet sich in den Abschnitten: 7.1.2 und 7.1.3 .

**org.smurve.heras.context** (kurz: o.s.h.context)

In diesem Paket finden sich die Interfaces für den Anfrage/Antwort-Context von XACML (z.B. RequestCtx, ResponseCtx, PDP). Das Paket ist sehr abstrakt gehalten.

Das Paket wird von Policy Decision Point und Policy Enforcement Points verwendet, um Zugriffsnachrichten auszutauschen. Zudem beschreiben die Schnittstellen das Verhalten eines PDP bezüglich Evaluierungsfunktionen.

**org.smurve.heras.persistence** (kurz: o.s.h.persistence)

Dieses Paket enthält Interfaces, welche die technischen Aspekte von Data Access Objects beschreiben (z.B. *PolicyDaoTechnical*). Es handelt sich hierbei um die Festspeicheroperationen der verschiedenen Strukturelemente aus *org.smurve.heras.model*.

Das Paket dient als Erweiterungspunkt an dem Persistenzmodule ansetzen können. *heras-hib3* implementiert die Interfaces, die sich in diesem Paket befinden. Funktionale Module (*heras-pdp*, *heras-pap* usw.) benutzen dieses Paket nicht.

Eine detailliertere Beschreibung findet sich im Abschnitt: 7.1.4.

**org.smurve.heras.dataaccess** (kurz: o.s.h.dataaccess)

Die Interfaces in diesem Paket (z.B. *PolicyDao*) stellen den Datenzugriff für die funktionalen Module zur Verfügung. Zusätzlich zu den Operationen, welche durch die technischen Data Access Object Interfaces beschrieben werden, bieten diese Interfaces noch Methoden an, mit denen neue Objekte erstellt werden können.

Die funktionalen Module dürfen aufgrund der abstrahierten Modellstruktur keine neuen Objekte direkt (mit dem Schlüsselwort „*new*“) erzeugen. Dies kommt daher, dass die funktionalen Module die effektiven Implementationen hinter den Interfaces nicht kennen können und sollen.

Damit keine zusätzliche abstrahierte API für die funktionalen Module realisiert werden muss, wird die Templategik innerhalb der Daos gekapselt. Diese bieten beispielsweise die Methode *newInstanceFromTemplate(HerasTemplate template)* an. Weil die Data Access Interfaces die Struktur von HERAS<sup>AF</sup> kennen, können sie auch die nötigen Abläufe für die Erstellung eines Policy-Objekts anhand eines Templates implementieren.

Eine detailliertere Beschreibung findet sich in den Abschnitten: 7.1.4 und 7.3.

**org.smurve.heras.impl** (kurz: o.s.h.impl)

Dieses Paket beinhaltet die Implementierung des Modells von HERAS<sup>AF</sup>. Darin enthalten sind die konkreten Klassen (z.B. *PolicyHolder*), welche die Trennung zwischen dem Persistenzmodul und dem Adaptermodul zur XACML-Implementation bewerkstelligen.



Diese Klassen implementieren die Interfaces aus `o.s.h.model` und delegieren diese Funktionalität die Klassen des XACML-Adaptermoduls. Dabei kennen diese Klassen nicht die effektiven Implementierungen, an die sie delegieren, sondern nur deren implementierte Interfaces aus `o.s.h.model`. Die effektiven Implementierungen werden durch den Spring IoC-Container eingehängt.

Die Klassen in `o.s.h.impl` kapseln ausserdem die persistenten Daten und werden demzufolge durch das Persistenzmodul gespeichert und geladen.

Eine detailliertere Beschreibung findet sich in den Abschnitten: 7.1.3, 7.2 und 7.3.

#### **org.smurve.heras.dataaccess.impl**

Die Klassen in diesem Paket implementieren die Interfaces aus `o.s.h.dataaccess` und sind nur innerhalb `heras-core` bekannt. Sie realisieren die Templatelogik bezüglich der Erstellung von Objekten (z.B. `newInstanceFromTemplate`). Die Persistenzaspekte des Datenzugriffs werden an die Klassen des Persistenzmoduls (`heras-hib3`) delegiert. Dabei kennen diese Klassen nicht die effektiven Implementierungen, an die sie delegieren, sondern nur deren implementierte Interfaces aus `o.s.h.model`. Die effektiven Implementierungen werden durch den Spring IoC-Container eingehängt.

Eine detailliertere Beschreibung findet sich im Abschnitt: 7.1.4.



### 7.1.2 Modell (Objektstruktur für XACML 2.0)

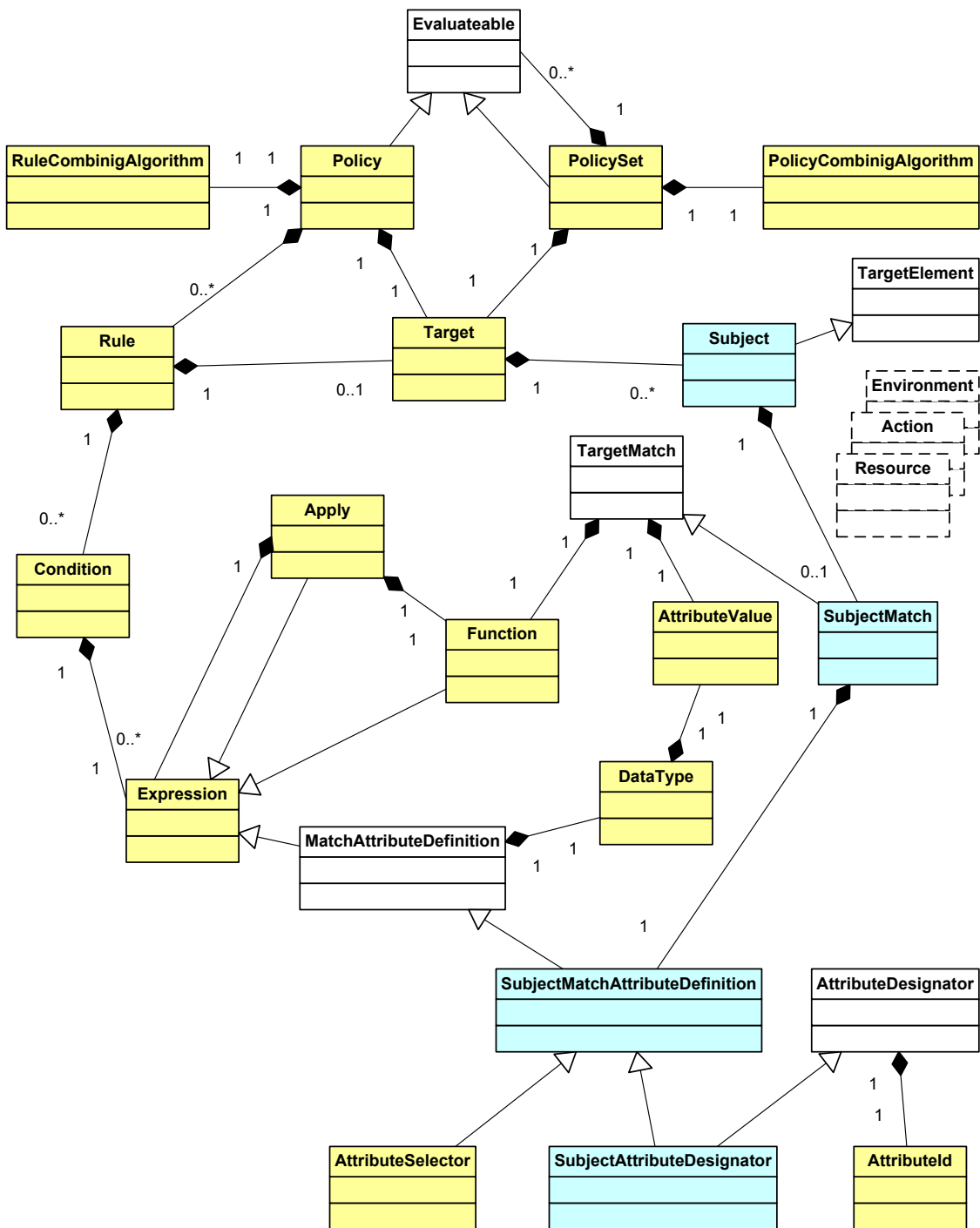


Abbildung 14: org.smurve.heras.model Interfacestruktur

Das Modell von HERAS<sup>AF</sup> entspricht weitestgehend dem XML-Schema von XACML 2.0. Die in Abbildung 14 ungefärbten Interfaces sind Hilfskonstrukte, die benötigt werden um die Elementbeziehungen, wie sie von XACML 2.0 vorgegeben werden, zu realisieren. Das Interface *Evaluateable* wird für den Policy Decision Point benötigt, damit nicht unterschieden werden muss, um welches evaluierbares Konstrukt (PolicySet, Policy) es sich handelt.



Um die Abbildung übersichtlich zu halten, wurde darauf verzichtet, die Elemente Resource, Action und Environment darzustellen. Die Beziehungen dieser Elemente sind analog wie die des Subject-Elements aufgebaut, inklusive dessen Subelementen.

Im Rahmen des Prototyps wurden aus Zeitgründen folgende Elemente nicht in das Modell integriert:

- PolicySetIdReference
- PolicyIdReference
- VariableReference
- Obligations und Obligation
- AttributeAssignment

Die Funktionalität der Referenzen und Zusicherungen werden in einer späteren Version von HERAS<sup>AF</sup> integriert.

Einige XACML-Elemente konnten oder können als Datenfelder integriert werden und mussten daher nicht als eigenständige Interfaces integriert werden. Zu diesen gehören:

- Description
- Version (wird in nächster Version von HERAS<sup>AF</sup> realisiert)



### 7.1.3 Vorlagenmechanismus (Templating)

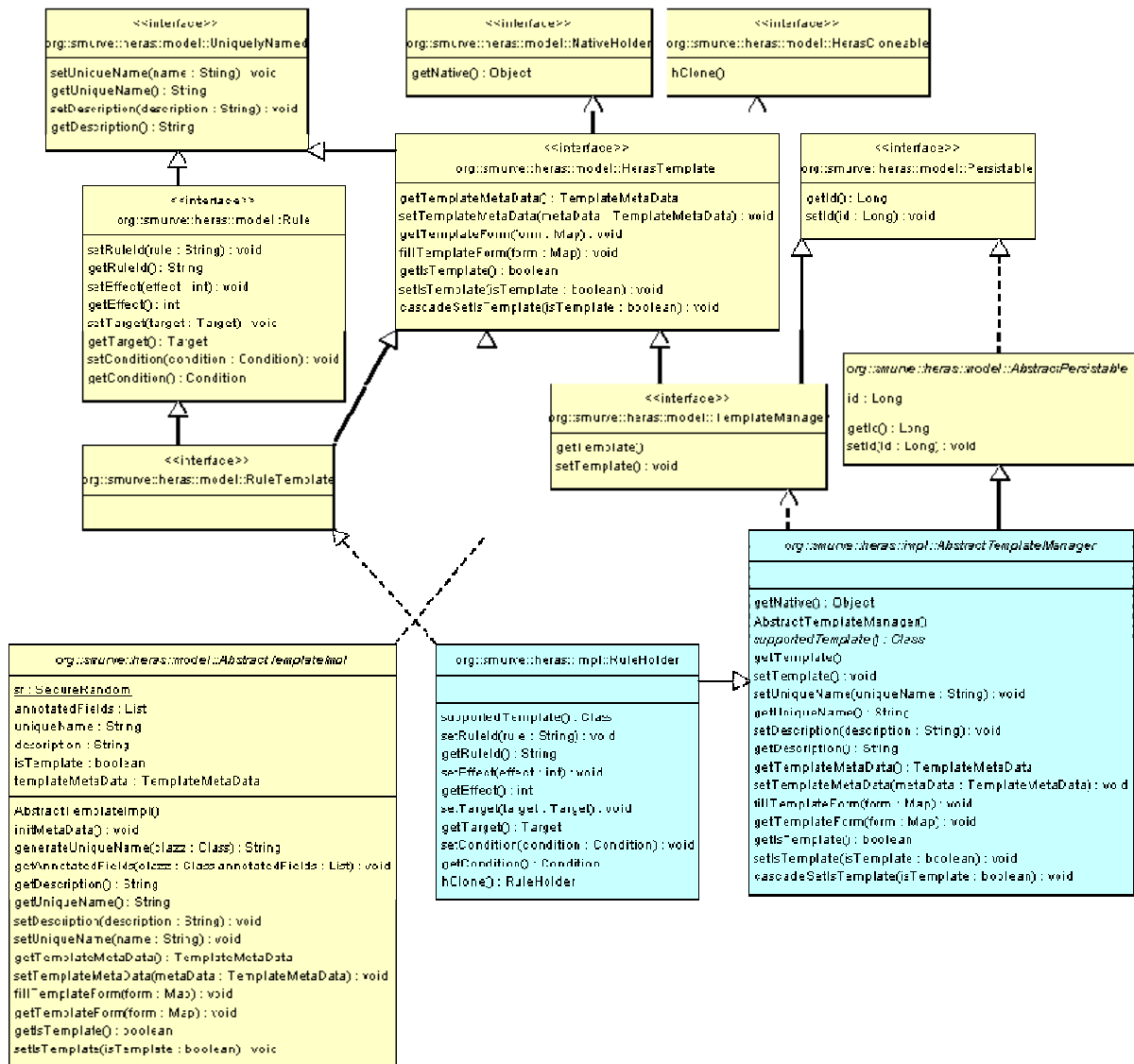


Abbildung 15: Klassen- und Interfacestruktur für Templating

Abbildung 15 zeigt die Interface und Klassenstruktur anhand einer Rule auf. Die anderen Elemente aus dem Kern von HERAS<sup>AF</sup> sind analog aufgebaut. Die Interfacestruktur ist durch viele Vererbungen realisiert. Dies kommt daher, dass alle Aspekte in eigenständigen Interfaces beschrieben werden. Komplexere Schnittstellen können diese Aspekte integrieren, indem sie von den anderen Interfaces erben.

**org.smurve.heras.model.UniquelyNamed** beschreibt den Aspekt der eindeutigen Namensgebung und der freien Beschreibung eines Elements.

**org.smurve.heras.model.NativeHolder** wird für die Trennung zu heras-sunxacml oder einem anderen XACML-Implementationsadapter verwendet. Das Interface schreibt vor, dass ein Element aus heras-core, welches ein Element aus heras-sunxacml hält, dieses auch in nativer Form zurückgeben können muss. Beispielsweise muss eine HERAS<sup>AF</sup>-Rule eine com.sun.xacml.Rule zurückgeben können.



**org.smurve.heras.model.Persistable** beschreibt den Aspekt der eindeutigen Identifikationsnummer. HERAS<sup>AF</sup> verwendet diesen Aspekt nicht, er ist für Persistenzfunktionen vorgesehen. Beispielsweise stellen ORM-Frameworks anhand der *Id* fest, ob das Objekt bereits in der Datenbank gespeichert wurde.

**org.smurve.heras.model.HerasCloneable** definiert den Aspekt des Klonens im Sinne von HERAS<sup>AF</sup>. Die Java `clone()`-Methode wurde nicht verwendet, da sie üblicherweise nicht so angewendet wird, wie es in HERAS<sup>AF</sup> benötigt wird. Die Klonmethode `hclone()` erstellt einen Klon des Objekts auf dem es aufgerufen wird und kaskadiert den Methodenaufruf an alle Subobjekte. Dabei werden auch Listen, Maps und Sets inklusive deren Inhalt geklont. Auch dabei wird nicht `java.lang.Object clone()` aufgerufen, sondern die Kopie explizit erstellt. Die Aspekte von *UniquelyNamed* und *Persistable* werden beim Klonen nicht mitkopiert. Die *Id* wird zurückgesetzt und für *uniqueName* wird mit *SecureRandom* ein Name generiert.

**org.smurve.heras.model.Rule** stellt das XACML-Element dar und bietet alle Funktionen, um seine Elementbeziehungen herstellen zu können. Das Interface definiert keine Weiterführende Methoden, welche zusätzliche Funktionalitäten anbieten.

**org.smurve.heras.model.TemplateMetaData** (in der Abbildung nicht gezeigt) ist eine Datenstruktur und speichert die Informationen, welche Datenfelder eines *HerasTemplates* für Vorlagen verwendet werden können. Die Datenstruktur verfügt über vier Felder (*fieldName*, *identifier*, *defaultValue*, *isTemplateField*), welche mehrere Key-Value-Beziehungen realisiert. Auf alle Felder kann mittels *identifier* als Schlüsselangabe zugegriffen werden. Zusätzlich besteht zwischen *fieldName* und *identifier* eine Key-Key-Beziehung. Der *identifier* entspricht dem geschäftsspezifischen Bezeichner eines Feldes und realisiert somit die „Business Abstraktion“.

**org.smurve.heras.model.HerasTemplate** beschreibt die Methoden, welche für den Vorlagenmechanismus benötigt werden. Die Information, ob ein Template auch als solches verwendet wird, wird mit einem booleschen Flag (*isTemplate*) gesetzt. Templates beinhalten alle eine Datenstruktur (*TemplateMetaData*). Beim Aufruf von `getTemplateForm` wird eine Map zurückgegeben, welche alle Templatefelder und deren Defaultwerte enthält. Der Aufruf wirkt kaskadierend, so dass auch alle Subelemente eines Templates ihre Vorlagenfelder in der Map eintragen. Mit `fillTemplateForm` werden Vorlagen ausgefüllt. Dabei holt sich jedes Template innerhalb der Kaskade die ausgefüllten Werte wieder aus der Map und ruft via Java Reflection den zugehörigen Setter auf. Auf diese Art und Weise kann beispielsweise mit zwei Methodenaufrufen eine neue Policy erstellt werden.

**org.smurve.heras.model.RuleTemplate** erweitert das allgemeine *HerasTemplate*. Falls zu einem späteren Zeitpunkt elementspezifische Templatefunktionen eingebunden werden müssen, so wird dies in diesen Templateinterfaces gemacht.

**org.smurve.heras.model.AbstractTemplateImpl** ist eine abstrakte Klasse. Diese stellt den Erweiterungspunkt für heras-sunxacml dar. Die Adapterklassen des Moduls für eine XACML-Implementation müssen von dieser Klasse ableiten. Siehe dazu Abschnitt 7.1.5. Zudem wird in dieser Klasse die Funktionalität bezüglich der *TemplateMetaDaten* integriert (`getTemplateMetaData`, `fillTemplateForm` usw.).

**org.smurve.heras.model.TemplateManager** definiert die Methoden, welche für die „Dependency Injection“ benötigt werden. Das sind Getters und Setters für den Spring IoC-Container.

**org.smurve.heras.impl.AbstractTemplateManager** fasst die aus der Vererbungsstruktur vorgegebenen Methoden zusammen und implementiert diejenigen, welche für alle Elemente unspezifisch realisiert werden können. Der *AbstractTemplateManager* hat ein *HerasTemplate* als Member und delegiert die Funktionsausführung an dieses.



**org.smurve.heras.impl.RuleHolder** ist eine konkrete Klasse. Diese realisiert die Funktionalität des *Rule*-Interfaces und leitet von *AbstractTemplateManager* ab. Der RuleHolder kann somit persistiert werden und die Funktionalität seines Delegates nutzen (Bridge Pattern).

### 7.1.4 Erweiterungspunkt: Persistenz und Datenzugriff

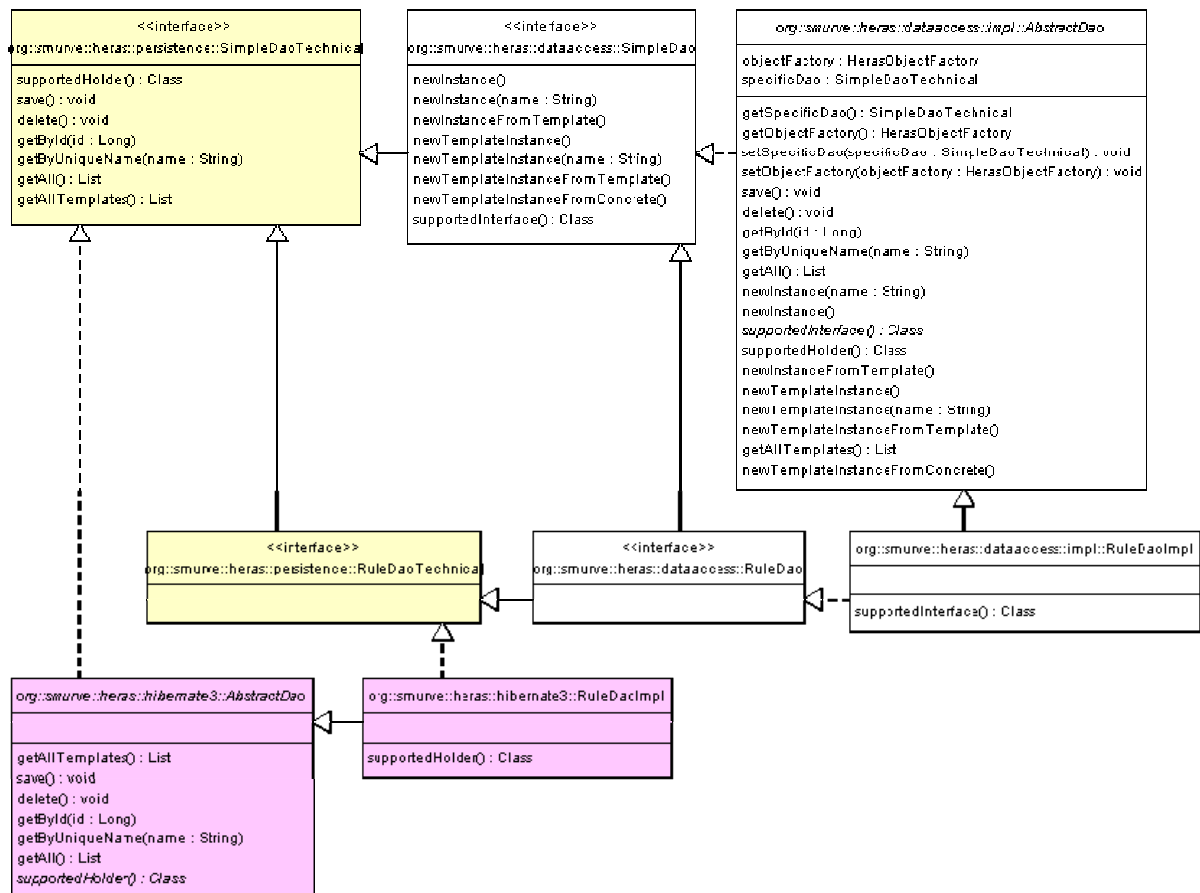


Abbildung 16: Klassen- und Interfacestruktur für Datenzugriff

Abbildung 16 zeigt die Interface und Klassenstruktur anhand einer Rule auf. Die anderen Elemente aus dem Kern von HERAS<sup>AF</sup> sind analog aufgebaut.

Der Datenzugriff innerhalb HERAS<sup>AF</sup> kann in drei Schichten (mit drei verschiedenen Aspekten) aufgeteilt werden. Die technische Schicht definiert die Persistenzaspekte des Speicherns und Ladens von Objekten. Die Zugriffsschicht erweitert die Persistenzschicht um Aspekte der Objekterzeugung. Die dritte Schicht realisiert die Implementation der Erzeugungsaspekte sowie die Delegationsaspekte an die technischen Dao-Implementationen.

**org.smurve.heras.persistence.SimpleDaoTechnical** beschreibt die Methoden, welche für alle technischen Datenzugriffsobjekte gelten.

**org.smurve.heras.persistence.RuleDaoTechnical** könnte zusätzliche elementspezifische Speicheroperationen definieren, sobald diese nötig werden. Zum Zeitpunkt dieser Diplomarbeit war dies noch nicht nötig. Der RuleDaoTechnical stellt den Erweiterungspunkt für heras-hib3 und andere Persistenzmodule dar. Die Klassen dieser Module müssen alle technischen ElementDaos realisieren.



**org.smurve.heras.dataaccess.SimpleDao** beschreibt die Methoden, welche für alle erweiterten Datenzugriffsobjekte gelten. Das Interface erweitert *SimpleDaoTechnical* um die Methoden, welche für die Objekterzeugung benötigt werden. Ausserdem kapselt es die Logik für die vorlagenbasierte Objekterzeugung.

**org.smurve.heras.dataaccess.RuleDao** könnte zusätzliche elementspezifische Datenzugriffsoperationen definieren, sobald diese nötig werden. Zum Zeitpunkt dieser Diplomarbeit war dies noch nicht nötig.

**org.smurve.heras.dataaccess.impl.AbstractDao** trennt die technischen Aspekte von den Objekterzeugungsaspekten (Bridge Pattern). Mittels Spring IoC-Container wird dem *AbstractDao* ein technisches Dao injiziert. An dieses delegiert er die Persistenzoperationen. Die Erzeugungsoperationen werden in der abstrakten Klasse implementiert.

**org.smurve.heras.hibernate3.AbstractDao** realisiert das *SimpleDaoTechnical*-Interface und implementiert die Persistenzoperationen, welche für alle technischen Dao's implementiert werden können.

**org.smurve.heras.persistence.RuleDaoImpl** ist die Realisierung des konkreten technischen Datenzugriffsobjektes.

### 7.1.5 Erweiterungspunkt: XACML-Implementation

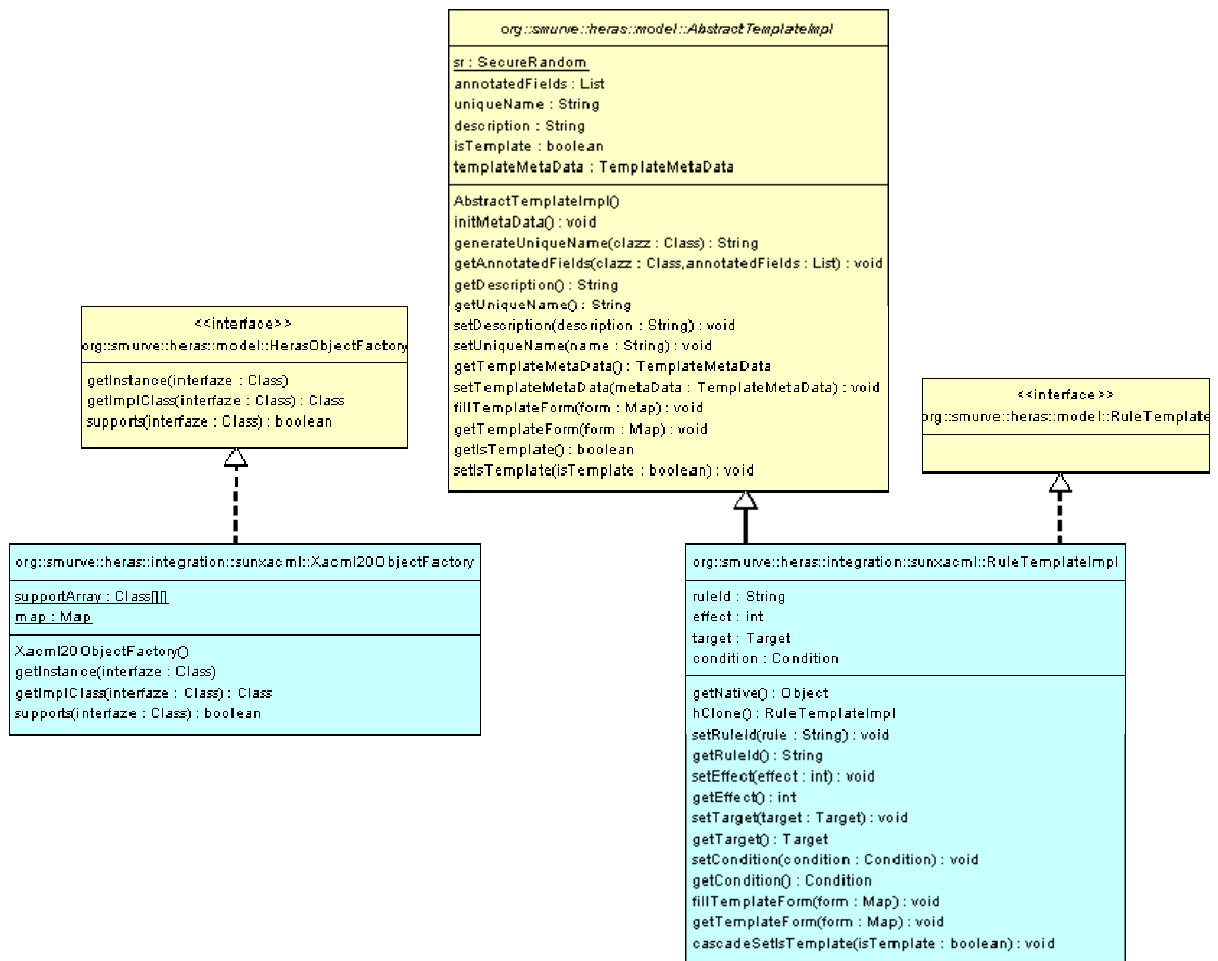


Abbildung 17: Implementationsstruktur für heras-sunxacml



Sun's XACML Integration wird in heras-sunxacml adaptiert. HERAS<sup>AF</sup> benötigt dazu Adapterklassen für die Elemente und die ObjectFactory.

**org.smurve.heras.model.HerasObjectFactory** wird verwendet, um die unabhängige Objekterzeugung der Modellelemente von HERAS<sup>AF</sup> zu realisieren. Weil heras-core vollkommen unabhängig ist, kann dem Modul nicht bekannt sein, welche Implementationen der Modellelemente im XACML-Adaptionsmodul vorliegen. Deshalb können innerhalb heras-core keine Objekte mit vollständiger Struktur erstellt werden. *org.smurve.heras.impl.ObjectFactoryImpl* baut daher die heras-core Objektstrukturen zusammen und benutzt die spezifische (IoC-injizierte) Factory, um die konkreten Objektstrukturen zu vervollständigen.

**org.smurve.heras.integration.sunxacml.Xacml20ObjectFactory** realisiert die spezifische Factory. Diese Factory implementiert das *HerasObjectFactory*-Interface und erzeugt die implementationsabhängigen Elemente.

**org.smurve.heras.model.AbstractTemplateImpl** ist eine abstrakte Klasse. Diese stellt den Erweiterungspunkt für heras-sunxacml dar. Die Adapterklassen des Moduls für eine XACML-Implementation müssen von dieser Klasse ableiten.

**org.smurve.heras.integration.sunxacml.RuleTemplateImpl** ist der konkrete Adapter an das entsprechende Element aus Sun's XACML Implementation.

## 7.2 heras-sunxacml

heras-sunxacml adaptiert die Sun's XACML Implementation. Die Adapterklassen werden mittels Spring IoC-Container in den heras-core injiziert. Ein Adapter eines Elements (z.B. Rule) muss die HERAS<sup>AF</sup>-Elementstruktur an die zu integrierende Struktur binden und in der Lage sein, implementationsabhängige Objekte zurückzuliefern. Die Verwendung von Factories ist in diesem Falle sehr empfehlenswert.

## 7.3 heras-hib3

Für die Realisierung der Persistenz mit Spring und Hibernate3 sind keine umfangreichen Implementationen erforderlich (siehe Abbildung 16). Die Implementierung der technischen Dao-Interfaces reichen aus, um die Persistenz in HERAS<sup>AF</sup> einzubinden.

Die Dao-Implementierungen in heras-hib3 verwenden den *HibernateDaoSupport* aus *org.springframework.orm.hibernate3.support*. Diese Klasse wird beerbt und stellt damit die load-, save- und query-Methoden zur Verfügung.

Die Abbildung von Java-Objekten auf die Datenbank wird mittels Hibernate Mapping Files realisiert. Abbildung 18 zeigt den Aufbau einer solchen XML-Datei am Beispiel der Rule.



```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
  "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
  "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd" >

<hibernate-mapping>
  <class name="org.smurve.heras.impl.RuleHolder" table="RULE"
    lazy="false">

    <id name="id" column="id" type="java.lang.Long">
      <generator class="native" />
    </id>

    <property name="uniqueName" column="UNIQUE_NAME"
      type="java.lang.String" not-null="true" />

    <property name="description" column="DESCRIPTION"
      type="java.lang.String" not-null="false" />

    <property name="isTemplate" column="IS_TEMPLATE"
      not-null="true" />

    <many-to-one name="templateMetaData" column="METADATA" unique="true"
      not-null="true" cascade="all-delete-orphan"
      class="org.smurve.heras.model.TemplateMetaDataImpl" />

    <property name="ruleId" column="RULE_ID" type="java.lang.String"
      not-null="true" />

    <property name="effect" column="EFFECT" not-null="true" />

    <many-to-one name="target" column="TARGET" not-null="false"
      insert="true" update="true"
      class="org.smurve.heras.impl.TargetHolder" lazy="false"
      cascade="all-delete-orphan" />

    <many-to-one name="condition" column="CONDITION"
      not-null="false" insert="true" update="true"
      class="org.smurve.heras.impl.ConditionHolder" lazy="false"
      cascade="all-delete-orphan" />
  </class>
</hibernate-mapping>
```

Abbildung 18: Hibernate Mapping File

Wie in der Abbildung zu sehen ist, verwendet heras-hib3 die Element-Holder für die Speicherung. Es wäre auch denkbar gewesen, diese nicht zu verwenden und stattdessen eine weitere Abstraktionsschicht explizit für die Persistenz zu realisieren. Da Hibernate aber keine direkten Abhängigkeiten aufgrund von Inhalten der Mapping-Files erzeugt, konnte diese Schicht weggelassen werden.



### 7.3.1 Datenbank

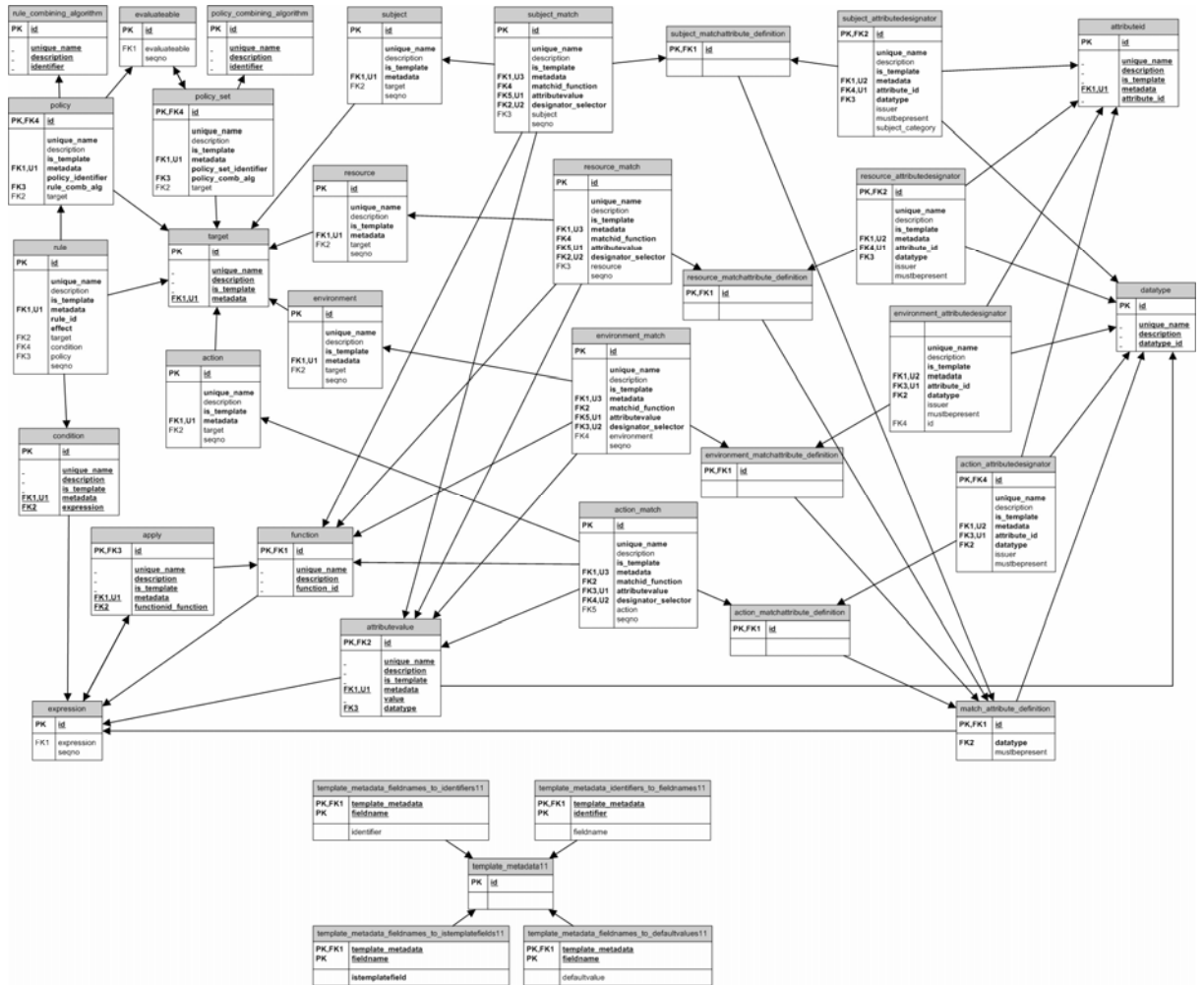


Abbildung 19: Datenbank ER-Diagramm

Durch den Einsatz eines ORM-Frameworks wird die Datenbankstruktur etwas unübersichtlich. Abbildung 19 zeigt die Tabellenabhängigkeiten. Diese vor allem wegen der Anwendung des Composite Patterns in den Klassenstrukturen so aufwändig.

Die Beziehungen zu den TemplateMetaDaten (unten im Bild) werden aus Übersichtsgründen nicht dargestellt, weil zu jedem Template auch die Metadaten gespeichert werden.

Für den Prototyp von HERAS<sup>AF</sup> wurden die Integritätsbedingungen auf Datenbankebene vernachlässigt. Für eine produktive Datenbank müssten diese noch aufgearbeitet werden.



## 7.4 Design Qualitätsanalyse

Die Designanalyse von HERAS<sup>AF</sup> zeigt (beurteilt anhand Robert Martins „OO Design Quality Metrics“) die typischen Merkmale eines Frameworks.

Die hoch interoperablen Pakete, welche von abhängigen Modulen benutzt werden, sind hochgradig abstrakt. Die implementationsbedingten Pakete sind hingegen sehr konkret.

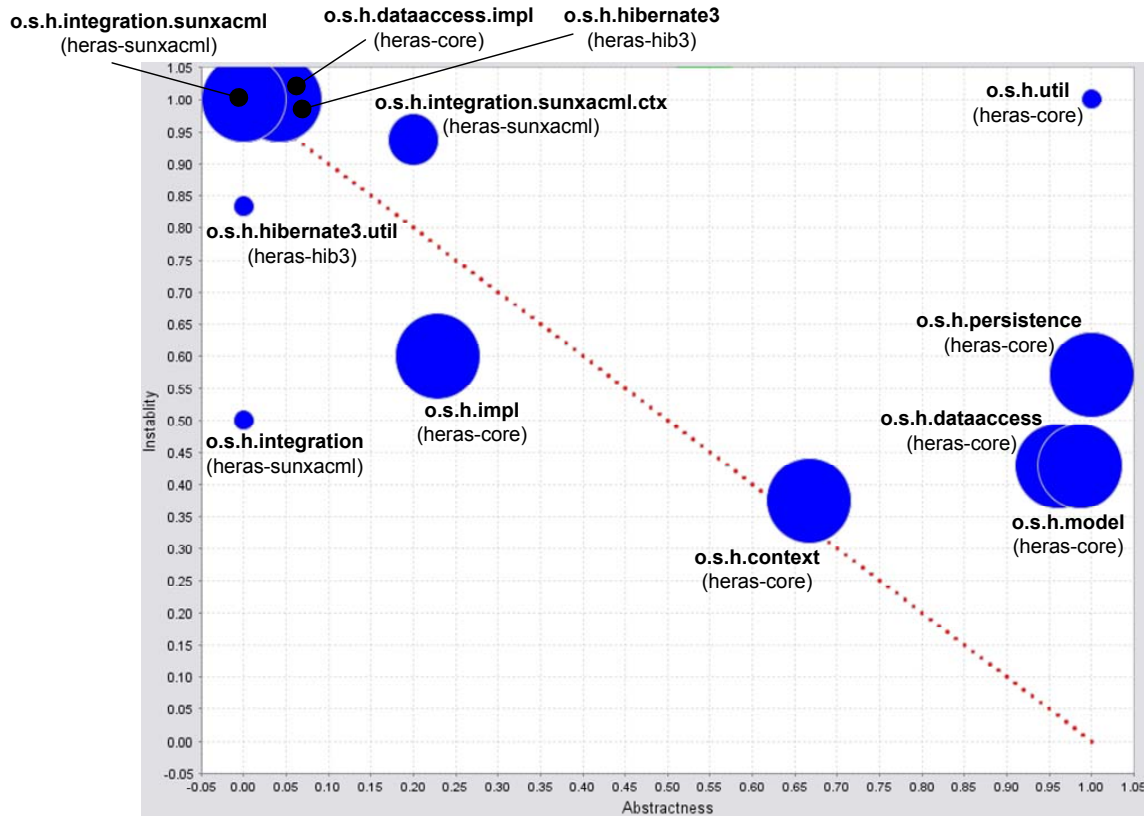


Abbildung 20: Analyse des Kerns von HERAS<sup>AF</sup> nach „OO Design Quality Metrics“

In Abbildung 20 lässt sich erkennen, dass für das Design von HERAS<sup>AF</sup> ein Paketstruktur entworfen wurde, welche keine unnötigen Abhängigkeiten enthält. Somit ist die komfortable Wartung und Weiterentwicklung von HERAS<sup>AF</sup>, trotz des grossen Umfangs, gewährt.



## 8 Resultate

### 8.1 Diplomarbeit

Die Ziele der Diplomarbeit konnten vollumfänglich umgesetzt werden:

- Das logische Struktur-Modell von HERAS<sup>AF</sup> realisiert ca. 95 Prozent der Funktionalität wie sie von XACML 2.0 vorgegeben wird.
- Sun's XACML Implementation konnte im Modul heras-sunxacml für alle implementierten Elemente des Modells von HERAS<sup>AF</sup> adaptiert werden.
- Das Persistenzmodul speichert alle Elemente von HERAS<sup>AF</sup> mittels Hibernate.
- Die API für die Benutzung des Modells und der Logik des Policy Administration Points konnte in die Data Access Objects verpackt werden. Damit konnte gezeigt werden, dass der Vorlagenmechanismus und die Business-Abstraktion mit dem Konzept von HERAS<sup>AF</sup> sinnvoll realisiert werden können.

Im Rahmen dieser Diplomarbeit wurde der Kern von HERAS<sup>AF</sup> so ausgearbeitet, dass dieser mit wenigen Optimierungen oder Änderungen, produktiv eingesetzt werden kann. Für den produktiven Einsatz müssen wenige Aspekte ausgebaut werden: Exception Handling, Database Constraints, JavaDoc der API.

### 8.2 HERAS<sup>AF</sup> Prototype und Anschauungsstudie

Der Prototyp, welcher rund um diese Diplomarbeit entwickelt wurde besteht aus:

- heras-acegi, welches einen funktionstüchtigen Policy Enforcement Point mittels Acegi Security realisiert. Dieser PEP kann mit wenig Aufwand in Webapplikationen integriert werden.
- heras-com, das die Kommunikationskapselung XACML, SAML, SOAP implementiert. Somit wurde eine offene, auf Standards aufbauende Kommunikation zwischen PEP und PDP realisiert werden.
- heras-pap, das den Policy Administration Point und den Policy Decision Point beinhaltet. Mit der JSF-Benutzeroberfläche konnte gezeigt werden, dass die Business-Abstraktion von HERAS<sup>AF</sup> in geeigneter Weise verwendet werden kann, um auf einfache Art und Weise Policies zu verwalten. Der PDP kann via Webservices bedient werden und führt die Evaluierungsaufgaben in korrekter Form aus.

Der Prototyp zeigt somit, dass eine unternehmenstaugliche, quelloffene Autorisierungslösung umsetzbar ist. Ausserdem konnten die Konzepte und Strategien von HERAS<sup>AF</sup> verifiziert werden.

### 8.3 Fazit

Es wurde gezeigt, dass es berechtigte Gründe dafür gibt, aus HERAS<sup>AF</sup> ein professionelles Open Source Projekt zu machen. Der Start des Projekts ist mit dem Prototyp erfolgreich gelungen. Das Projektteam wird daher die Ziele von HERAS<sup>AF</sup> weiter verfolgen.



## 9 Appendix

### 9.1 Definitionen

### 9.2 Abkürzungen

ACL	Access Control List
API	Application Programming Interface
EPAL	Enterprise Privacy Authorization Language
Hib3	Hibernate 3
HSR	Hochschule für Technik Rapperswil
IoC	Inversion of Control
o.s.h	org.smurve.heras
PAP	Policy Administration Point
PBAC	Policy-based Access Control
PDP	Policy Decision Point
PEP	Policy Enforcement Point
PIP	Policy Information Point
RBAC	Role-based Access Control
XACML	eXtensible Access Control Language
SAML	Security Assertion Markup Language
SOAP	Simple Object Access Protocol

### 9.3 Literaturverzeichnis

#### 9.3.1 Spezifikationen, Standards und Definitionen

- [XACML2.0] OASIS  
eXtensible Access Control Markup Language TC v2.0 (XACML)  
<http://docs.oasis-open.org/xacml/2.0/XACML-2.0-OS-NORMATIVE.zip>
- [SAML2.0] OASIS  
Security Assertion Markup Language (SAML) v2.0  
<http://docs.oasis-open.org/security/saml/v2.0/saml-2.0-os.zip>
- [SAMLXACML] SAML 2.0 profile of XACML v2.0



[http://docs.oasis-open.org/xacml/2.0/access\\_control-xacml-2.0-saml-profile-spec-os.pdf](http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-saml-profile-spec-os.pdf)

- [SAMLSOAP] Bindings for the OASIS Security Assertion Markup Language (SAML) V2.0  
<http://docs.oasis-open.org/security/saml/v2.0/saml-bindings-2.0-os.pdf>
- [rfc3198] The Internet Society  
Terminology for Policy-Based Management  
<http://www.ietf.org/rfc/rfc3198.txt>
- [rfc2748] The Internet Society  
The COPS (Common Open Policy Service) Protocol  
<http://www.rfc-archive.org/getrfc.php?rfc=2748>
- [rfc2753] The Internet Society  
A Framework for Policy-based Admission Control  
<http://rfc.sunsite.dk/rfc/rfc2753.html>
- [iso10181] JTC 1; ISO Standards  
Information technology -- Open Systems Interconnection -- Security frameworks for open systems: Access control framework  
ISO/IEC 10181-1:1996
- [OpenSource] The Open Source Definition  
<http://www.opensource.org/docs/definition.php>

### 9.3.2 Bücher

- [SCL05b] John Sherwood, Andrew Clark, David Lynas.  
Enterprise Security Architecture.  
CMPBooks, ISBN: 1-57820-318-X
- [Mart03] Robert C. Martin. Agile Software Development, Principles, Patterns, and Practices.  
Prentice Hall, ISBN: 0-13-597444-5
- [FKC03] David F. Ferraiolo, D. Richard Kuhn, Ramaswamy Chandramouli.  
Role-Based Access Control  
Artech House Publishers, ISBN: 1580533701
- [GHHJPTW02] Ben Galbraith, Whitney Hankison, Andre Hiotis, Murali Janakiraman, D. V. Prasad, Ravi Trivedi, David Whitney  
Professional Web Services Security  
Wrox Press, ISBN: 1861007655
- [Verm00] Dinesh Verma  
Policy-Based Networking: Architecture and Algorithms



Sams, ISBN: 1578702267

- [ACM01] Deepak Alur, John Crupi, Dan Malks  
Core J2EE Patterns: Best Practices and Design Strategies  
ISBN: 0130648841

- [SNL05] Christopher Steel, Ramesh Nagappan, Ray Lai  
Core Security Patterns – Best Practices and Strategies for J2EE, Web Services, and Identity Management  
ISBN: 0-13-146307-1

### 9.3.3 Artikel, Arbeiten

- [Dem05] Yuri Demchenko.  
Using SAML and XACML for Authorisation assertions and messaging: SAML and XACML standards overview and usage examples.  
<http://www.uazone.org/demch/analytic/draft-authz-xacml-saml-02.pdf>
- [Dem06] Yuri Demchenko.  
Policy-based Access Control to Data Services in Service-oriented Architecture and Grid  
<http://www.touchbriefings.co.uk/download.cfm?fileID=6143>
- [DeFe05] Nelly Delessy and Eduardo B. Fernandez  
Patterns for the eXtensible Access Control Markup Language  
[http://hillside.net/plop/2005/proceedings/PLoP2005\\_ndelessyandebfernandez0\\_1.pdf](http://hillside.net/plop/2005/proceedings/PLoP2005_ndelessyandebfernandez0_1.pdf)
- [KeLu04] Alexander Keller, Heiko Ludwig  
Policy-basiertes Management: State-of-the-Art und zukünftige Fragestellungen  
<http://www.research.ibm.com/people/a/akeller/Data/pik2004.pdf>
- [And05] Anne Anderson.  
A Comparison of Two Privacy Policy Languages: EPAL and XACML.  
<http://research.sun.com/techrep/2005/smlitr-2005-147/TRCompareEPALandXACML.html>
- [LPLKS03] Markus Lorch, Seth Proctor, Rebekah Lepro, Moffett Field, Dennis Kafur, Sumit Shah  
First experiences using XACML for access control in distributed systems.  
[http://portal.acm.org/ft\\_gateway.cfm?id=968563&type=pdf&coll=ACM&dl=ACM&CFID=74569526&CFTOKEN=46608553](http://portal.acm.org/ft_gateway.cfm?id=968563&type=pdf&coll=ACM&dl=ACM&CFID=74569526&CFTOKEN=46608553)
- [Rih04] Jan Rihak  
Access Control Markup Languages for XML Documents  
<http://www.infsec.ethz.ch/education/projects/archive/AC4XMLReport.pdf>
- [Verm04] Manish Verma



XML Security: Control information access with XACML. The objectives, architecture, and basic concepts of eXtensible Access Control Markup Language

<http://www-128.ibm.com/developerworks/xml/library/x-xacml/>

- [FeKu95] David Ferraiolo, Rick Kuhn  
AN INTRODUCTION TO ROLE-BASED ACCESS CONTROL  
<http://csrc.nist.gov/rbac/NIST-ITL-RBAC-bulletin.html>
- [FeKu92] David Ferraiolo, Rick Kuhn  
Role-Based Access Control  
<http://csrc.nist.gov/rbac/ferraiolo-kuhn-92.pdf>
- [SCHFY95] Ravi S. Sandhu, Edward J. Coynek, Hal L. Feinsteink, Charles E. Youmank  
Role-Based Access Control Models  
<http://csrc.nist.gov/rbac/sandhu96.pdf>
- [LABW92] Butler Lampson, Martín Abadi, Michael Burrows, Edward Wobber  
Authentication in distributed systems: theory and practice  
<http://doi.acm.org/10.1145/138873.138874>
- [Slom94] Morris Sloman  
Policy Driven Management For Distributed Systems (1994)  
<http://citeseer.ist.psu.edu/sloman94policy.html>
- [Mos03] Tim Moses  
Web-services policy language use3 cases and requirements  
<http://www.oasis-open.org/committees/download.php/1608/wd-xacml-wspl-use-cases-04.pdf>
- [John05] Rod Johnson  
Introduction to the Spring Framework  
<http://www.theserverside.com/articles/article.tss?l=SpringFramework>
- [Seuf05] Christian Seufert  
Das Spring Framework  
<http://www.st.informatik.tu-darmstadt.de/database/seminars/data/Spring.pdf?id=87>
- [SCL05p] John Sherwood, Andrew Clark, David Lynas  
Enterprise Security Architecture – White Paper  
<http://www.idrisk.com/uploads/EnterpriseSecurity.pdf>
- [Fayed97] Mohammed Fayed, Douglas C. Schmidt  
Object-Oriented Application Frameworks  
<http://www.cs.wustl.edu/~schmidt/CACM-frameworks.html>



- [Martin94] Robert Martin  
OO Design Quality Metrics  
<http://www.objectmentor.com/resources/articles/oodmetrc.pdf>
- [Fowler04] Martin Fowler  
Inversion of Control Containers and the Dependency Injection pattern  
<http://www.martinfowler.com/articles/injection.html>
- [Kuhl05] Dr. Martin Kuhlmann  
Geschäftsorientiertes Provisioning mit Regeln und Rollen  
[http://www.uspmarcom.de/itverlag/idm05/documents/BetaSystemsProvisioningmitRegelnundRollen\\_MartinKuhlmann.pdf](http://www.uspmarcom.de/itverlag/idm05/documents/BetaSystemsProvisioningmitRegelnundRollen_MartinKuhlmann.pdf)

### 9.3.4 Sonstige

- [SunXACML] Sun's XACML Implementation Guide. Programmer's Guide for Version 1.2  
<http://sunxacml.sourceforge.net/guide.html>
- [OpenSAML] OpenSAML - an Open Source Security Assertion Markup Language implementation  
<http://www.opensaml.org/>
- [Spring] Spring Framework  
<http://www.springframework.org/>
- [Hib3] Hibernate - Relational Persistence for Java and .NET  
<http://www.hibernate.org/>
- [WikiAC06] Wikipedia.org  
Access Control  
[http://en.wikipedia.org/wiki/Access\\_control](http://en.wikipedia.org/wiki/Access_control)