

Diplomarbeit

Distributed Access Control Policies -  
Enterprise Ready

Yan Graf

Abteilung Informatik  
Hochschule für Technik Rapperswil

Dezember 2006

**Betreuer:**  
Wolfgang Giersche  
**Experte:**  
Tomasz Bieruta

## Diplomarbeit

**Titel:** Distributed Access Control Policies - Enterprise Ready

**Autor:** Yan Graf, ygraf@hsr.ch

**Datum:** Dezember 2006

Abteilung Informatik

Hochschule für Technik Rapperswil

# Abstract

In der heutigen Zeit entstehen vermehrt sogenannte *Virtuelle Unternehmen*, welche mittels zwischenbetrieblicher Kooperation ihre Dienstleistungen anbieten. Zusätzlich nehmen immer mehr Benutzer mit unterschiedlichen Rechten diese Dienstleistungen in Anspruch. Dies führt zu gegensätzlichen Herausforderungen. Unternehmen müssen sich öffnen, zugleich aber die Sicherheit erhöhen.

Diese Diplomarbeit realisiert einen unternehmenstauglichen *Policy Decision Point* für verteilte Autorisierungsprozesse. Die zentrale Aufgabe besteht im Verteilen und Auffinden von Sicherheitsrichtlinien. Die gesamte Funktionalität wird über *Webservices* zur Verfügung gestellt und kann so in einer *service-orientierten Architektur (SOA)* eingesetzt werden. Definierte Schnittstellen ermöglichen das Bereitstellen von unternehmensabhängigen Informationen (zum Beispiel zusätzliche Benutzerdaten) für den Autorisierungsprozess. Entsprechende Realisierungen dieser Schnittstellen können mittels *Dependency Injection* dem *Policy Decision Point* zur Verfügung gestellt werden.



# Erklärung

Ich versichere hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die im Literaturverzeichnis angegebenen Quellen benutzt habe.

Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder noch nicht veröffentlichten Quellen entnommen sind, sind als solche kenntlich gemacht.

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

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

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

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



# Inhaltsverzeichnis

<b>Abstract</b>	<b>iii</b>
<b>Erklärung</b>	<b>v</b>
<b>1 Einführung</b>	<b>1</b>
1.1 HERAS <sup>AF</sup> . . . . .	1
1.2 Zielsetzung . . . . .	1
<b>2 Zugriffskontrolle</b>	<b>3</b>
2.1 Strategien . . . . .	3
2.1.1 Listenbasierte Zugriffskontrolle . . . . .	4
2.1.2 Rollenbasierte Zugriffskontrolle . . . . .	4
2.1.3 Richtlinienbasierte Zugriffskontrolle . . . . .	5
2.2 Anforderungen an PBAC . . . . .	8
2.3 XACML . . . . .	9
2.3.1 Modell . . . . .	11
2.4 HERAS <sup>AF</sup> . . . . .	15
2.4.1 Ausgangslage und Motivation . . . . .	16
2.4.2 Ziele . . . . .	16
<b>3 Policy Decision Point</b>	<b>19</b>
3.1 Anforderungen . . . . .	19
3.1.1 Funktionstüchtigkeit . . . . .	19
3.1.2 Performanz . . . . .	20
3.1.3 Erreichbarkeit . . . . .	20
3.1.4 Sicherheitsaspekte . . . . .	21
3.1.5 Kommunikation <i>PDP</i> zu <i>PDP</i> . . . . .	21
3.1.6 Mehr-Werte Logik . . . . .	21
3.1.7 Zusicherungen . . . . .	22
3.2 HERAS <sup>AF</sup> <i>PDP</i> . . . . .	22
3.2.1 Performanz . . . . .	23
3.2.2 Erreichbarkeit . . . . .	25
3.2.3 Sicherheitsaspekte . . . . .	25
3.2.4 Kommunikation <i>PDP</i> zu <i>PDP</i> . . . . .	27
<b>4 Architektur</b>	<b>29</b>
4.1 Module . . . . .	29
4.1.1 <i>heras-com</i> . . . . .	29

---

4.1.2	<i>heras-int</i>	30
4.1.3	<i>heras-pdp</i>	30
4.2	<i>ContextHandler</i>	31
4.3	<i>Requestors</i>	32
4.4	Kommunikation <i>PDP</i> zu <i>PDP</i>	33
4.4.1	Verteilte Evaluierung	35
4.5	Administration	36
4.6	Persistenz	36
4.6.1	Persistenzstrategien	37
4.6.2	Derby	38
<b>5</b>	<b>Design und Realisierung</b>	<b>39</b>
5.1	<i>Spring IoC Container</i>	39
5.2	Packages	41
5.3	Package Design	41
5.4	Klassen	43
5.5	Änderungen von HERAS <sup>AF</sup>	44
<b>6</b>	<b>Zusammenfassung und Ausblick</b>	<b>47</b>
6.1	Zusammenfassung	47
6.2	Ausblick	48
	<b>Literaturverzeichnis</b>	<b>49</b>
	<b>Appendix</b>	<b>50</b>
<b>A</b>	<b>Internetadressen</b>	<b>51</b>
<b>B</b>	<b>Abkürzungen</b>	<b>53</b>

# Abbildungsverzeichnis

2.1	Richtlinienbasierte Zugriffskontrolle . . . . .	6
2.2	XACML Datenfluss [OASIS05a] . . . . .	12
2.3	XACML 2.0 Modell [OASIS05a] . . . . .	14
3.1	Obligations . . . . .	23
3.2	<i>Point-to-Point</i> Architektur . . . . .	27
3.3	<i>Hub-and-Spoke</i> Architektur . . . . .	28
4.1	<i>PDP</i> zu <i>PDP</i> Kommunikation . . . . .	30
4.2	<i>PolicyRequestor</i> . . . . .	33
4.3	<i>Policy</i> Deployment . . . . .	37
5.1	Package Design . . . . .	42



# Einführung

Diese Arbeit entstand im Rahmen einer Diplomarbeit an der Hochschule für Technik Rapperswil und wurde zur Erlangung des akademischen Grades eines Diplomingenieurs für Informatik vorgelegt.

## 1.1 HERAS<sup>AF</sup>

Im Sommer 2006 entstand der HERAS<sup>AF</sup> Prototyp, ein ganzheitliches und unternehmenstaugliches Zugriffskontrollsystem basierend auf *XACML* (*eXtensible Access Control Markup Language*) Richtlinien. Der Kern des Prototypen realisiert ca. 95% der *XACML 2.0* Funktionalität. Zusätzlich beinhaltet er eine Benutzerschnittstelle zur Erstellung und Verwaltung von Richtlinien, einen *Policy Enforcement Point* mittels *Acegi Security* zum Schützen von Webapplikationen sowie die Integration der *Sun's XACML* Implementation.

Die *eXtensible Access Control Markup Language* beschreibt den Aufbau von Sicherheitsrichtlinien sowie die Kommunikation zwischen einzelnen Komponenten für richtlinienbasierte Autorisierungssysteme. Ein besonderes Merkmal von *XACML* ist die Unterstützung von verteilten Autorisierungslösungen.

Diese Arbeit legt ein besonderes Augenmerk auf einen verteilten Autorisierungsprozess innerhalb des HERAS<sup>AF</sup> Prototypen.

## 1.2 Zielsetzung

Die zentrale Komponente in einer verteilten Autorisierungslösung ist der *Policy Decision Point* (*PDP*). Er ist dafür verantwortlich, anhand von Sicherheitsrichtlinien Autorisierungsentscheidungen zu treffen. Diese Diplomarbeit soll Anforderungen an einem dezentralen und unternehmenstauglichen *PDP* beschreiben und entsprechende Lösungen anbieten. Die wesentlichen Aufgaben sind:

- das Verteilen von Richtlinien
- das Auffinden und die Evaluierung von verteilten Richtlinien

- das Finden von verteilten Attributen

Ein weiteres Ziel ist die Realisierung eines *PDPs*, der im Rahmen des HERAS<sup>AF</sup> Prototypen einen verteilten Autorisierungsprozess umsetzt.

# Zugriffskontrolle

Die zentrale Frage im Zusammenhang mit Zugriffskontrolle ist:

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

Die Zugriffskontrolle steuert also den Zugriff auf Ressourcen (passive Entitäten, Daten, Dienste) durch Subjekte (aktive Entitäten, Benutzer, Systeme).

Das *Wer* bezeichnet das Subjekt, welches auf Ressourcen zugreift (*Was*), und welche Aktion (*Wie*) ausgeführt werden soll. Die *Umstände* bezeichnen äussere Einflüsse, welche Rahmenbedingungen für den Zugriff beschreiben. So können Zugriffe zum Beispiel nur zu bestimmten Zeiten erlaubt werden.

Zur Zugriffskontrolle gehören folgende Aufgaben:

**Authentisierung:** Das Nachweisen der eigenen Identität.

**Authentifizierung:** Die Überprüfungen der Identität eines Gegenübers, zum Beispiel einer Person oder eines Computerprogramms.

**Autorisierung:** Die Überprüfung der Zugriffsrechte von Subjekten auf Daten und Dienste.

**Auditing:** Das detaillierte Protokollieren von Zugriffen.

Diese Arbeit befasst sich im wesentlichen mit der verteilten Autorisierung.

## 2.1 Strategien

Die bekanntesten und am meisten verwendeten Zugriffskontrollstrategien sind:

- listenbasierte Zugriffskontrolle
- rollenbasierte Zugriffskontrolle
- richtlinienbasierte Zugriffskontrolle

### 2.1.1 Listenbasierte Zugriffskontrolle

Die listenbasierte Zugriffskontrolle verwendet sogenannte Zugriffskontrolllisten (engl. *Access Control List, ACL*). Aktuelle Betriebssysteme (Windows, Unix, Linux, Mac OS X, ...) unterstützen Zugriffskontrolllisten, um den Zugriff auf Dateien oder Dienste zu erlauben oder verbieten. Die Vorteile von Zugriffskontrolllisten werden am folgenden Beispiel illustriert.

Die klassische Zugriffskontrolle unter Linux verwendet für Dateien und Dienste die drei Klassen *Benutzer* (engl. *user*), *Gruppe* (engl. *group*) und *Anderere* (engl. *other*). Jeder dieser Klassen können Lese- (*read*), Schreib- (*write*) und Ausführungsrechte (*execute*) gesetzt werden. In vielen Fällen reicht diese Rechteverwaltung aus. Es können aber Situationen auftreten, welche eine detailliertere Beschreibung der Zugriffsberechtigung erfordern. Dies zeigt folgendes Beispiel:

- Mitarbeiter<sup>1</sup> in einem Krankenhaus führen eine Liste mit Patienten (*Patienten.xml*) in ihrer Station.
- Die Mitarbeiter dieser Station dürfen die Liste lesen.
- Die Mitarbeiter sind in der Gruppe *Mitarbeiter* zusammengefasst.
- Nur Ärzte dürfen in die Liste schreiben.
- Ärzte sind zusätzlich in der Gruppe *Ärzte* zusammengefasst.
- Ansonsten darf die Liste von niemandem gelesen werden.

Da eine Datei nur *einem* Benutzer und *einer* Gruppe zugewiesen ist, kann das genannte Beispiel mit der klassischen Zugriffskontrolle nicht abgebildet werden.

Durch den Einsatz von Zugriffskontrolllisten kann der Datei *Patienten.xml* für unterschiedliche Gruppen verschiedene Rechte zugewiesen werden. So können der Gruppe *Ärzte* Lese- und Schreibrechte, der Gruppe *Mitarbeiter* nur Leserechte vergeben werden.

Zugriffskontrolllisten werden meist für jede Ressource einzeln geführt. Die Verwaltung ist daher sehr aufwändig und teuer. Darüber hinaus ist die Berücksichtigung von äusseren Zuständen und Abhängigkeiten verschiedener Aktionen kaum oder nur schwer realisierbar.

### 2.1.2 Rollenbasierte Zugriffskontrolle

Rollenbasierte Zugriffskontrolle (engl. *Role Based Access Control, RBAC*) wurde zu einem wichtigen Modell für die Zugriffskontrolle in grossen und vernetzten Applikationen, da es die Komplexität und Kosten für die Administration reduziert [NIST06].

In einem rollenbasierten Kontrollsystem werden Aufgaben, Eigenschaften und vor allem Rechte mittels Rollen (engl. *Roles*) definiert. Die Zugriffsrechte auf Ressourcen werden durch das Zuweisen von Rollen an Benutzer vergeben.

<sup>1</sup>Um den Lesefluss zu erleichtern, wird in der vorliegenden Arbeit jeweils nur die männliche Form der Substantive, stellvertretend für beide Geschlechter, verwendet.

Zugriffe können so durch Anpassung der Rollen für mehrere Subjekte erlaubt oder verweigert werden. Dadurch wird der Verwaltungsaufwand von Zugriffsrechten vereinfacht, besonders wenn sich die Rechtestruktur ändert, da nur die Rechte der *Rollen* angepasst werden müssen.

*Rollen* können zudem hierarchisch gegliedert werden. Dies erleichtert das Abbilden von Organisationsstrukturen. Entsprechende *Rollen* können von Geschäftsfunktionen abgeleitet und den entsprechenden Subjekten zugeteilt werden.

Dennoch bietet die rollenbasierte Zugriffskontrolle nicht genügend Flexibilität, um die Verwaltung der *Rollen* in Unternehmen mit dynamischen Strukturen kostengünstig zu realisieren. Insbesondere in projektorientierten Organisationen, deren Projekte von kurzer Dauer sind, ändern sich Benutzerrollen häufig. Des Weiteren müssen bei grossen Änderungen in der Unternehmensstruktur die *Rollen* weitläufig überarbeitet werden [EGG06].

### 2.1.3 Richtlinienbasierte Zugriffskontrolle

Die richtlinienbasierte Zugriffskontrolle bietet Lösungen für die oben genannten Nachteile von listen- und rollenbasierten Kontrollsystemen.

Das Definieren von Sicherheitsrichtlinien ist für jedes Unternehmen von zentraler Bedeutung. Meist werden diese Grundsätze in *Security Policies* verfasst. Das Ziel besteht darin, ein einheitliches Sicherheitsniveau in einer komplexen Organisation zu gewährleisten, sowie verbindliche und überprüfbare Vorgaben für die mindestens erforderlichen Sicherheitsmassnahmen zu machen [Secunet].

Die richtlinienbasierte Zugriffskontrolle (engl. *Policy Based Access Control, PBAC*) verfolgt den Ansatz, diese *Security Policies* durch Zugriffskontrollrichtlinien zu beschreiben. Dies ermöglicht, die semantische Lücke zwischen der natürlichen Sprache und der technischen Beschreibung klein zu halten.

Aus unternehmerischer Sicht können zwei Hauptaufgaben an die richtlinienbasierte Zugriffskontrolle gestellt werden: Einerseits sollen verschiedene Applikationen geschützt werden, andererseits muss die Verwaltung der Richtlinien (engl. *Policies*) überschaubar sein. Daraus lässt sich eine logische Trennung zwischen dem Entscheiden (*Decision*) und dem Durchsetzen (*Enforcement*) von Richtlinien ableiten. Die *Internet Engineering Task Force (IETF)* beschreibt in RFC 2753 [IETF00] und RFC 3198 [IETF01] ein abstraktes Modell für das Durchsetzen von Richtlinien. Dieses Modell ist in der Abbildung 2.1 schematisch dargestellt.

Der Ablauf geschieht folgendermassen:

1. Ein Subjekt möchte auf eine geschützte Ressource zugreifen.
2. Der *Policy Enforcement Point (PEP)*<sup>2</sup> unterbricht den Zugriff, formuliert eine Autorisierungsanfrage und schickt diese an einen *Policy Decision Point (PDP)*<sup>3</sup>.

---

<sup>2</sup>Logische Komponente, welche den Zugriff auf geschützte Ressourcen abfängt und die Entscheidungen vom PDP durchführt.

<sup>3</sup>Logische Komponente, welche mit Hilfe von Richtlinien Entscheidungen trifft.

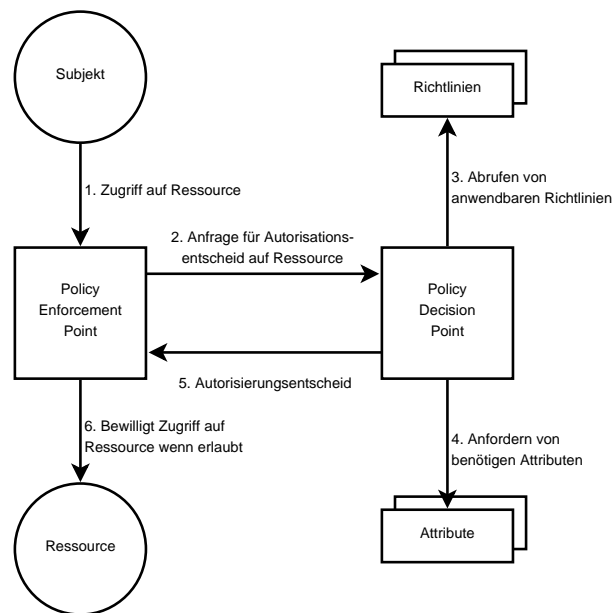


Abbildung 2.1: Richtlinienbasierte Zugriffskontrolle

3. Der *PDP* sucht auf die Anfrage anwendbare Richtlinien.
4. Sofern nötig werden zusätzliche Informationen oder Attribute angefordert.
5. Der *PDP* wertet die anwendbaren Richtlinien auf die gestellte Anfrage aus und erstellt einen Autorisierungsentscheid, welcher an den *PEP* zurück geschickt wird.
6. Ist der Entscheid positiv, so lässt der *PEP* den Zugriff auf die Ressource zu, ansonsten wird der Zugriff verweigert.

Die logische Unterteilung in Entscheidungs- und Durchführungskomponenten bringt folgende Vorteile:

**Administration von Richtlinien:** Richtlinien werden zentral gehalten und administriert. Dies erleichtert deren Verwaltung.

**Zentrale Autorisierung:** Die Autorisierung wird an einer zentralen Stelle durchgeführt. Dadurch kann ein *PDP* verschiedene *PEPs* bedienen. Zusätzliche Ressourcen können so durch Hinzufügen eines neuen *PEPs* geschützt werden.

**Transparente Integration in bestehende Systeme:** Der Zugriff auf geschützte Ressourcen wird durch einen vorangestellten *PEP* unterbrochen und abhängig von der Entscheidung des *PDPs* erlaubt oder verweigert. Aus diesem Grund sind Anpassungen an bestehenden Systemen für die Autorisierung nicht nötig.

**Plattform- und Herstellerunabhängigkeit:** Durch die Verwendung einer einheitlichen und offenen Sprache für die Kommunikation zwischen *PEP*

und *PDP* sowie für die Formulierung von Richtlinien kann die Autorisierung Plattform- und Herstellerunabhängig erfolgen.

Richtlinien ermöglichen eine detaillierte Beschreibung von Zugriffsrechten und dementsprechend einen flexiblen Zugriffsschutz auf geschützte Ressourcen oder Applikationen. Aufgrund der überschaubaren Administration können Kosten gespart werden. Zusätzlich lässt sich eine rollenbasierte Zugriffskontrolle durch Richtlinien abbilden [OASIS05a].

Damit der Einsatz von richtlinienbasierten Zugriffsstrategien realisierbar ist, muss eine einheitliche Sprache definiert werden, sowohl für die Beschreibung von Richtlinien als auch für die Kommunikation zwischen den Komponenten *PDP*, *PEP* und *PAP*<sup>4</sup>.

Die vermutlich bekanntesten Sprachen, so genannte *Privacy Policy Languages*, sind die *Enterprise Privacy Authorization Language (EPAL)* von IBM und die *eXtensible Access Control Markup Language (XACML)* von OASIS. Oft wird im Zusammenhang mit *Privacy Policy Language* auch die *Platform for Privacy Preferences (P3P)* genannt, ein Standard vom *World Wide Web Consortium (W3C)*.

*P3P* und *XACML* dienen ergänzenden Zielen. *P3P* beschreibt Richtlinien in einer Form, die für den Menschen lesbar ist. *XACML* jedoch definiert dieselben Richtlinien in einer Form, die für Computer verständlich und umsetzbar ist. *P3P* ist dementsprechend benutzerorientiert und ist für die unternehmenstaugliche Autorisierung nicht entscheidend.

Ein detaillierter Vergleich zwischen *EPAL* und *XACML* wurde von Anne Anderson [AAND05] verfasst. Aus diesem Vergleich geht hervor, dass *EPAL* aus einer Untermenge von *XACML* realisiert werden kann. Zusätzlich hat *XACML* folgende Vorteile gegenüber *EPAL*:

- Richtlinien können geschachtelt werden
- Unterstützung von Kombinalgorithmen
- Das Verweigern oder Erlauben von Zugriffen ist gleichbedeutend (in *EPAL* überschreibt das Erlauben das Verweigern) und wird mittels Kombinalgorithmen gesteuert
- Mehrere anwendbare Richtlinien können kombiniert werden
- Unterstützung von verteilten Richtlinien
- Webservices Profil (Draft)

Aufgrund dieser Vorteile und der umfangreicheren Funktionsvielfalt verwendet HERAS<sup>AF</sup> *XACML* für die Beschreibung von Richtlinien und die Kommunikation zwischen den einzelnen Komponenten. Zudem steigt das allgemeine Interesse an *XACML* stetig an, was als Zeichen für die zukünftige Verbreitung der Sprache zu sehen ist.

Zusätzliche detaillierte Informationen über *XACML* werden in Kapitel 2.3 beschrieben.

---

<sup>4</sup>*Policy Administration Point*, logische Komponente, welche das Erstellen und Administrieren von Richtlinien ermöglicht.

## 2.2 Anforderungen an PBAC

*XACML* stellt folgende elementare Anforderungen an eine richtlinienbasierte Autorisierungssprache [OASIS05a]:

**Verteilte Richtlinien:** Die Sprache muss das Verteilen von Richtlinien unterstützen und eine abstrakte Vorgehensweise für das Lokalisieren und Auswerten von verteilten Richtlinien definieren. Dadurch können Personen aus unterschiedlichen Fachbereichen Richtlinien verfassen und verwalten. Zusätzlich ermöglicht dies die Realisierung von Autorisierungsprozessen über Unternehmensgrenzen hinaus. Das Verteilen von Richtlinien ist ein zentrales Thema dieser Arbeit.

**Vereinigung von Richtlinien und Regeln:** Anwendbare Richtlinien und Regeln müssen zu einer Gruppe von Richtlinien kombiniert werden können, um einen Entscheid auf eine Autorisierungsanfrage zu treffen.

**Kombinierbarkeit von Effekten:** Der Effekt<sup>5</sup> von verschiedenen Richtlinien und Regeln muss durch eine flexible Definition kombinierbar sein. Enthält eine Richtlinie mehrere Regeln, welche zu unterschiedlichen Effekten evaluieren, so müssen diese durch Kombinieralgorithmen zu einem einzigen Effekt abgeleitet werden können.

**Mehr-Werte Logik:** Da bei der Evaluation einer Richtlinie Fehler auftreten können, muss zusätzlich zu den Entscheidungen *Erlauben* und *Verbieten* ein entsprechender Status *Unbestimmbar* möglich sein. Werden keine anwendbaren Richtlinien gefunden, so wird ein Status *Nicht Anwendbar* definiert.

**Schnelle Überprüfung der Anwendbarkeit:** Das Überprüfen auf die Anwendbarkeit einer Richtlinie muss performant sein.

**Abstrakte Schicht:** Eine abstrakte Schicht soll das Definieren von Richtlinien, abgeschirmt von den Details der Applikationsumgebung, ermöglichen.

**Obligationen:** Zusätzliche Verpflichtungen und Zusicherungen beim Durchsetzen von Autorisierungsentscheiden müssen beschrieben werden können.

**Unterstützung für mehrere Subjekte:** Mehrere Subjekte mit verschiedenen Ermächtigungen müssen unterstützt werden.

**Unterstützung von Attributen:** Autorisierungsentscheide müssen aufgrund von Attributen vom Subjekt, der Ressource oder von Umgebungsinformationen getroffen werden können. Diese Attribute sollen sowohl statische als auch dynamische Werte (abhängig zum Beispiel vom Subjekt) enthalten können.

**Zusammengesetzte Attribute:** Attribute müssen aus verschiedenen Informationen zusammengesetzt werden können.

**Unterstützung von Standarddatentypen:** Für eine effektive Evaluation von Richtlinien müssen Standarddatentypen unterstützt werden und von

---

<sup>5</sup>Die vorgesehene Konsequenz einer anwendbaren Regel, entweder Erlauben oder Verweigern.

Vorteil bereits in einer Implementation von *XACML* enthalten sein.

**Erweiterung von Datentypen:** Die Definition und Verwendung von eigenen Datentypen muss möglich sein.

**Logische und Mathematische Operationen:** Werte vom Subjekt, von der Ressource oder von der Umgebung müssen durch geeignete logische oder mathematische Operationen kombiniert oder zu neuen Werten abgeleitet werden können.

**Unterstützung von Standardoperationen:** Für die effektive Evaluation von Richtlinien müssen Standardoperationen unterstützt werden und von Vorteil bereits in einer Implementation von *XACML* enthalten sein.

**Erweiterung von Operationen:** Zusätzliche Operationen müssen definiert werden können. Dies ist unter anderem für die Verarbeitung eigener Datentypen erforderlich.

Zusätzlich zu den technischen und formalen Voraussetzungen gelten für HERAS<sup>AF</sup> weitere Anforderungen. Diese wurden unter dem Gesichtspunkt einer unternehmenstauglichen Autorisierungslösung definiert.

**Verwaltung von Richtlinien:** Richtlinien können geschachtelt und verteilt werden. Dies führt zu beliebig komplexen Strukturen. Eine entsprechende Administrationssoftware soll das Verwalten dieser Strukturen ohne technische Kenntnisse ermöglichen.

**Offene Sprache:** Die Autorisierungssprache darf nicht herstellerspezifisch sein. Dadurch lässt sich die Autorisierung über Unternehmensgrenzen hinaus realisieren, unabhängig von der Implementation des Autorisierungssystems.

**Integration in bestehende Systeme:** Die Integration von einem richtlinienbasierten Zugriffssystem soll ohne grosse Änderungen an bestehenden Systemen möglich sein.

**Technische Abstraktion:** Richtlinien sollen ohne technisches Wissen der verwendeten Autorisierungssprache erstellt und verwaltet werden können. Als Folge davon soll es in jedem Bereich eines Unternehmens möglich sein, entscheidende Sicherheitsrichtlinien in Geschäftssprache zu formulieren.

## 2.3 XACML

*XACML* ist ein Standard für eine allgemein gültige Beschreibung von Zugriffskontrollrichtlinien in *XML*. Die Version 2.0 wurde im Februar 2005 von der Standardisierungsorganisation *Organisation for the Advancement of Structured Information Standards (OASIS)* in Zusammenarbeit mit BEA Systems, Computer Associates, Entrust, IBM, Sun Microsystems und weiteren Organisationen definiert.

*XACML* beschreibt den Syntax von Richtlinien sowie das Format der Autorisierungsanfrage und -antwort. Die Sprache bietet Erweiterungspunkte, unter

anderem zur Definition von eigenen Attributen, Funktionen oder Kombinalgorithmen. Ausserdem können weitere Standards eingebunden werden. Dazu gehören:

- *Security Assertion Markup Language (SAML)*, eine XML-basierte Auszeichnungssprache für den sicheren Austausch von Authentifizierungs- und Autorisierungsinformationen zwischen Sicherheitssystemen.
- *Webservice Security (WSS)*, ein Standard für das Sicherstellen der Integrität und Vertraulichkeit von Nachrichten im Zusammenhang mit Webservices.
- *Service Provisioning Markup Language (SPML)*, ein XML-basiertes Framework für den Austausch von Benutzerzugriffsrechten inklusive zugewiesener Ressourcen zwischen verschiedenen Systemen.
- *Digital Signature Standard (DSS)*, ein Standard zur Erzeugung von digitalen Signaturen.
- *Public Key Infrastructure (PKI)*, eine Infrastruktur zur Verwaltung und Distribution von kryptographischen Schlüsseln.

Die *eXtensible Access Control Markup Language* ist kein komplettes Autorisierungssystem, vielmehr beschreibt sie eine Basis, auf welcher ganzheitliche Lösungen aufbauen können. *XACML* definiert weit entwickelte Eigenschaften, welche sich für das Verbinden von grossen Autorisierungslösungen eignen.

*Sun Microsystems* beschreibt die wichtigsten Vorteile von *XACML* folgendermassen [Sun05]:

**XACML ist ein Standard:** Zumal *XACML* eine ratifizierte Sprache ist, können Entwickler aus den Erfahrungen einer grossen Gemeinschaft profitieren. Daher müssen sie nicht immer eigene Systeme entwickeln und sich Gedanken über die verwendete Autorisierungssprache machen. Darüber hinaus, je weiter sich *XACML* verbreitet, umso einfacher lässt sich eine Integration in verschiedene Systeme realisieren, welche dieselbe standardisierte Sprache verwenden.

**XACML ist generisch:** In Folge dessen kann *XACML* für jedes Umfeld und jede Ressource verwendet werden. Richtlinien können wieder verwendet werden. Dies erleichtert zusätzlich die Administration von Richtlinien.

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

**XACML ist mächtig:** Es gibt viele Möglichkeiten, die Sprache zu erweitern, auch wenn das für die meisten Fälle nicht nötig ist. Bereits die Standardversion von *XACML* unterstützt viele Datentypen, Funktionen und Regeln für das Kombinieren von verschiedenen Resultaten von Richtlinien. Zusätzlich arbeiten Standardisierungsgruppen an Anbindungsprofilen und Erweiterungen, damit *XACML* mit anderen Standards (zum Beispiel *SAML*, *XML Digital Signature*) verwendet werden kann.

Ein Nachteil einer so flexiblen und mächtigen Sprache wie *XACML* ist die Menge von zusätzliche Metadaten. Werden *XACML* Daten mittels *SAML*<sup>6</sup> und *SOAP*<sup>7</sup> verschickt, so entstehen bereits für wenig effektive Nutzdaten verhältnismässig grosse Nachrichten.

### 2.3.1 Modell

Abbildung 2.2 zeigt den Datenfluss, wie er von der *XACML* Spezifikation definiert wird [OASIS05a]. Folgende logischen Elemente sind zu unterscheiden, wobei diese nicht als physikalisch eigenständige Komponenten implementiert werden müssen:

**Policy Administration Point (PAP):** Der *PAP* ist die Administrationskomponente. Mit Hilfe des *PAPs* können Richtlinien erstellt, verwaltet und dem *PDP* zur Verfügung gestellt werden.

**Policy Enforcement Point (PEP):** Der *PEP* ist der *Durchführer*. Er unterbricht Zugriffe auf geschützte Ressourcen und leitet eine Autorisierungsanfrage an den *Context Handler* weiter. Abhängig von den Autorisierungsentscheidungen des *PDPs* muss der *PEP* Zugriffe auf Ressourcen erlauben oder verweigern. Sind zusätzliche Obligationen in den Entscheidungen enthalten, so muss er diese erfüllen.

**Policy Decision Point (PDP):** Der *PDP* ist der *Entscheider*. Er erhält eine Entscheidungsanfrage und generiert abhängig von den anwendbaren Richtlinien eine Entscheidungsantwort, welche den Zugriff auf die geschützte Ressource erlaubt oder verweigert. Beinhalten die anwendbaren Richtlinien Obligationen, so muss der *PDP* diese mit der Entscheidungsantwort mitschicken.

**Policy Information Point (PIP):** Der *PIP* ist die Informationskomponente. Er stellt zusätzliche Informationen über Subjekte, Ressourcen, Umfeld oder Aktionen zur Verfügung, welche von einer Richtlinie referenziert werden können.

**Context Handler (CH):** Der *Context Handler* hat zwei Aufgaben. Er ist verantwortlich dafür, dass Entscheidungsanfragen von *PEPs*, welche nicht in *XACML* formuliert sind, in eine *XACML* Entscheidungsanfrage umgeformt wird. Zusätzlich ist er für das Bereitstellen von zusätzlichen Informationen für den *PDP* zuständig.

Das Modell agiert folgendermassen:

1. Der *PAP* definiert Richtlinien und stellt diese dem *PDP* zur Verfügung.
2. Der *access requester* (Benutzer oder System) schickt eine Zugriffsanfrage auf eine durch einen *PEP* geschützte Ressource.
3. Der *PEP* schickt eine Entscheidungsanfrage in *XACML* oder nativen Form an den *Context Handler*. Optional kann die Anfrage zusätzliche Attribu-

---

<sup>6</sup>Security Assertion Markup Language

<sup>7</sup>Simple Object Access Protocol

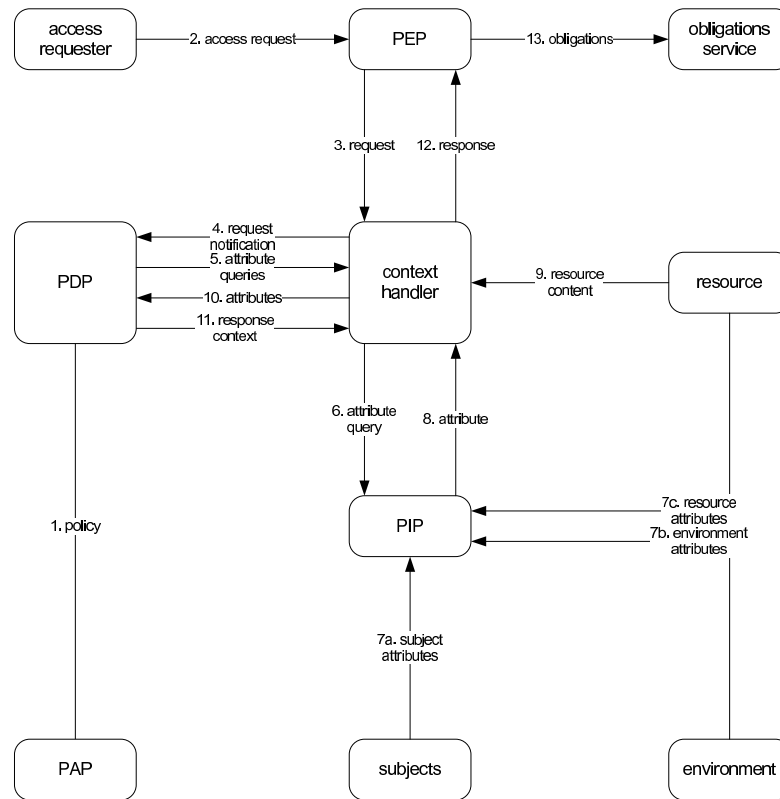


Abbildung 2.2: XACML Datenfluss [OASIS05a]

tinformationen über die Subjekte, die Ressource, das Umfeld oder die Aktion beinhalten.

4. Ist die Anfrage vom *PEP* nicht in *XACML* formuliert, so erstellt der *Context Handler* eine *XACML* Anfrage und schickt diese an den *PDP*. Ansonsten leitet er die Anfrage unverändert weiter.
5. Der *PDP* fordert, wenn nötig, zusätzliche Subjekt-, Ressource-, Umfeld- und Aktionsinformationen vom *Context Handler* an.
6. Der *Context Handler* fordert die entsprechenden Attributinformationen von einem *PIP* an.
7. Der *PIP* holt die angeforderten Informationen ein.
8. Der *PIP* sendet die angeforderten Informationen an den *Context Handler* zurück.
9. Optional kann der *Context Handler* die Ressource in die Anfrage einfügen.
10. Der *Context Handler* schickt die angeforderten Attribute und optional die Ressource an den *PDP* zurück. Der *PDP* evaluiert die anwendbaren Richtlinien auf die Anfrage.
11. Der *PDP* schickt eine Antwort an den *Context Handler* zurück, welche

den Autorisierungsentscheid beinhaltet. Sind in den evaluierten Richtlinien Obligationen enthalten, so müssen diese zusätzlich in der Antwort enthalten sein.

12. Der *Context Handler* übersetzt die Antwort in die native Form, die für den *PEP* verständlich ist. Versteht der *PEP* *XACML*, so wird die Antwort unverändert weitergeleitet.
13. Sind in der Antwort Obligationen<sup>8</sup> enthalten, so muss der *PEP* diese ausführen. Sind die Obligationen dem *PEP* unbekannt, so darf er den Zugriff nicht erlauben.  
Ist der Autorisierungsentscheid vom *PDP* positiv, der Zugriff also erlaubt, so gestattet der *PEP* den Zugriff auf die geschützte Ressource. Andernfalls verbietet er den Zugriff.

Für das Schützen von Ressourcen werden oft mehrere Richtlinien definiert. So kann der Zugriff auf Daten zum Beispiel durch zwei Richtlinien geschützt werden. Eine Richtlinie beschreibt allgemeine Datenschutzbestimmungen, eine weitere realisiert firmenbezogene Sicherheitsrichtlinien. Dieses Szenario kann mit *XACML* abgebildet werden, indem *Policies* in sogenannte *PolicySets* gruppiert werden. Weitere Komponenten aus dem *XACML*-Modell und deren Zusammenspiel zeigt das UML-Diagramm in der Abbildung 2.3.

Die wichtigsten Komponenten vom Modell sind:

- *Rule*
- *Policy*
- *PolicySet*

Diese werden nachfolgend kurz beschrieben.

### ***Rule***

Eine *Rule* ist die grundlegendste Einheit einer *Policy* und setzt sich aus den folgenden Elementen zusammen:

- einem *Target* (optional)
- einer *Condition* (optional)
- einem *Effect*

Das *Target*-Element definiert, ob eine *Rule* für eine Autorisierungsanfrage anwendbar ist. Fehlt das *Target*-Element, so gelten die *Target*-Elemente der umschließenden *Policy*.

Das *Condition*-Element kann die Anwendbarkeit auf das *Target*-Element verfeinern. Auf diese Weise kann eine *Rule* zum Beispiel abhängig von der Tageszeit anwendbar sein.

---

<sup>8</sup>Obligationen werden im Kapitel 3.1.7 erleutert.

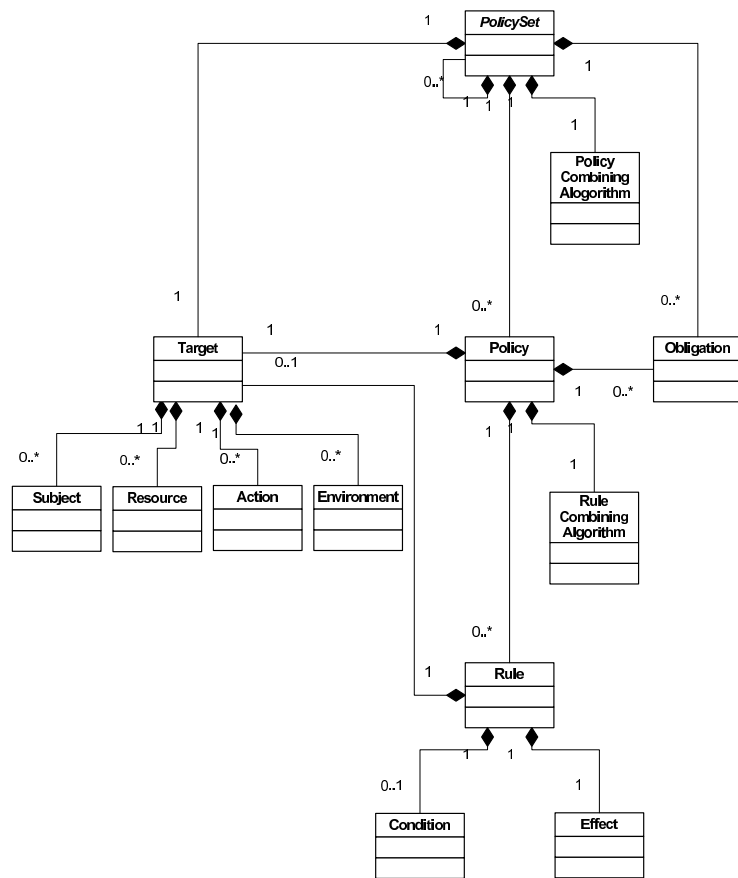


Abbildung 2.3: XACML 2.0 Modell [OASIS05a]

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

### *Policy*

Eine *Rule* kann nicht selbstständig existieren, sie muss einer *Policy* hinzugefügt werden. Eine *Policy* setzt sich aus den folgenden vier Elementen zusammen:

- einem *Target*
- einer Menge von *Rules*
- einem *rule-combining* Algorithmus
- einer Menge von *Obligations* (optional)

Das *Target*-Element definiert die Anwendbarkeit der *Policy* auf eine Autorisierungsanfrage. Damit die Effekte der enthaltenen *Rules* zu einem einzelnen

Entscheid kombiniert werden können, wird ein entsprechender *rule-combining* Algorithmus verwendet.

*XACML* definiert bereits einige Algorithmen, zum Beispiel *deny-overrides* oder *first-applicable*. Es können eigene Kombinalgorithmen geschrieben und verwendet werden.

Eine *Policy* kann *Obligations* einschliessen. *Obligations* beinhalten Aktionen, welche vom *PEP* vor der Umsetzung des Autorisierungsentscheides des *PDPs* erfüllt werden müssen.

### ***PolicySet***

Ein *PolicySet* ist eine Vereinigung einer Menge von *Policies*. Ähnlich wie eine *Policy* besitzt ein *PolicySet*

- ein *Target*
- eine Menge von *Policies* oder *PolicySets*
- ein *policy-combining* Algorithmus
- eine Menge von *Obligations*

Das *Target*-Element sowie die *Obligations* werden in der gleichen Weise verwendet, wie bei einer *Policy*. Ein *PolicySet* kann eine Menge von *Policies* oder erneut *PolicySets* enthalten. Anstelle von einem *rule-combining* Algorithmus der *Policy* verwendet das *PolicySet* einen *policy-combining* Algorithmus zum Kombinieren der Effekte von *Policies*. *XACML* bietet zum Kombinieren von *Policies* dieselben Algorithmen an, wie zum Vereinen von *Rules*, wie z. B. *deny-overrides* oder *first-applicable*.

### **Target**

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

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

Das *Target*-Element ist das wichtigste Element, um die Anwendbarkeit zu bestimmen.

## **2.4 HERAS<sup>AF</sup>**

HERAS<sup>AF</sup> ist eine Open Source Autorisierungslösung in der Entstehungsphase. Nach zirka acht Monaten Ideenentwicklung, Konzeption und Projektion wurde

das Projekt im Januar 2006 von Wolfgang Giersche, René Eggenschwiler und mir gestartet.

### 2.4.1 Ausgangslage und Motivation

In der heutigen Zeit entstehen immer mehr sogenannte virtuelle Unternehmen, welche durch Vernetzungen mit anderen Firmen ihre Dienstleistungen anbieten. Dies zwingt Firmen, den Zugriff auf kritische Ressourcen, Informationen und Applikationen zu öffnen. Produkte und Dienstleistungen können so effizienter geliefert werden. Dies führt zu tieferen Kosten und höherer Produktivität.

Oft sind Benutzerinformationen über verschiedene Systeme in einem Unternehmen verteilt. Jede Applikation hat seine eigene Benutzerverwaltung und Zugriffskontrolle. Dadurch wird das Durchsetzen von allgemeinen und firmenbezogenen Sicherheitsrichtlinien erschwert und eine Autorisierung über Unternehmensgrenzen ist kaum möglich.

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

HERAS<sup>AF</sup> nimmt diese Herausforderung an und setzt sich zum Ziel, verteilte Autorisierung unternehmenstauglich zu machen. Der Einsatz von einer offenen und technologieunabhängigen Sprache wie *XACML* macht dies möglich. Dabei setzt HERAS<sup>AF</sup> ausschliesslich auf frei verfügbare und quelloffene Komponenten.

### 2.4.2 Ziele

Eine vollwertige Autorisierungslösung sollte den oben genannten Anforderungen genügen. HERAS<sup>AF</sup> verfolgt folgende Prioritäten [EGG06]:

**Ganzheitlicher Ansatz:** HERAS<sup>AF</sup> ist eine ganzheitliche Autorisierungslösung und unterstützt den gesamten Prozess. Dies bedeutet, dass alle Applikationen und Ressourcen geschützt werden können. Durch den Einsatz von verteilten Agenten (*PEP*) werden Zugriffe auf geschützte Ressourcen unterbrochen und für den Autorisierungsentscheid an eine zentrale Einheit (*PDP*) geschickt. Ist der Entscheid positiv, so kann der Zugriff erlaubt werden. Zusätzlich unterstützt HERAS<sup>AF</sup> das Definieren und Verwalten von Richtlinien, ohne dass dafür technische Kenntnisse nötig sind.

**Unternehmenstauglich:** Für den Einsatz von HERAS<sup>AF</sup> in einem Unternehmen sollen höchstens kleine Änderungen an der bestehenden Infrastruktur und den Systemen nötig sein. Vorhandene Autorisierungslösungen sollen weiterhin bestehen oder integriert werden können.

Anpassungen und Erweiterungen von HERAS<sup>AF</sup> sollen jederzeit möglich sein. Dadurch können zum Beispiel bestehende Benutzerinformationen zur

Verfügung gestellt werden.

Aufgrund der Verwendung von offenen und etablierten Standards ist die Interoperabilität mit bestehenden und zukünftigen Systemen gewährleistet.

HERAS<sup>AF</sup> trennt das Erstellen von Richtlinien in eine technische und geschäftliche Schicht. Ein Experte erstellt Vorlagen für Richtlinien (technisch), welche von einem Berechtigungsverantwortlichen in natürlicher Sprache (geschäftlich) vervollständigt werden können.

**Application Security:** Durch das Vorschalten oder Einbetten von Agenten (PEP) kann der Schutz von Applikationen und Ressourcen an HERAS<sup>AF</sup> delegiert werden.

**Architecture Framework:** HERAS<sup>AF</sup> ist ein Architecture Framework und verbindet quelloffene Standards zu einer ganzheitlichen Autorisierungslösung.



# *Policy Decision Point*

Ein *Policy Decision Point*, kurz *PDP*, ist die logische Entscheidungseinheit in einer richtlinienbasierten Autorisierungslösung. An zentraler Stelle eingesetzt, muss der *PDP* Autorisierungsanfragen (engl. *decision requests*) von mehreren *Policy Enforcement Points*, kurz *PEP*, entgegennehmen und jeweils eine Autorisierungsentscheid (engl. *authorization decision*) zurückschicken. Die *authorization decision* wird abhängig vom *decision request* und der darauf anwendbaren *Policies* evaluiert und beinhaltet im einfachsten Fall eine Zugriffserlaubnis oder -verweigerung.

Zu den Kernaufgaben eines *PDPs* gehören:

- das Auffinden von *Policies*
- das Überprüfen der Anwendbarkeit von *Policies* bezüglich eines *decision requests*
- die Evaluation der anwendbaren *Policies* auf den *decision request*
- das Zurückschicken einer *decision response* an den *PEP*

## **3.1 Anforderungen**

Der *PDP* ist das Kernstück in einem richtlinienbasierten Autorisierungssystem. Daher muss er verschiedenen Anforderungen genügen, damit er für die Autorisierung verwendbar ist. Diese Anforderungen werden nachfolgend beschrieben.

### **3.1.1 Funktionstüchtigkeit**

Der *PDP* trifft die Autorisierungsentscheide, welche bestimmen, ob auf eine geschützte Ressource zugegriffen werden darf oder nicht. Treten bei der Evaluation Fehler auf, so kann das schwerwiegende Folgen für ein Unternehmen haben. Dabei sind folgende Szenarien denkbar:

**Zugriffe auf geschützte Ressource werden fälschlicherweise erlaubt:**

- Sensible Daten könnten in die falschen Hände geraten
- Nicht entsprechend geschulte Benutzer könnten geschäftskritische Applikationen falsch bedienen

**Zugriffe auf geschützte Ressourcen werden fälschlicherweise verboten:**

- Mitarbeiter könnten ihre Aufgaben nicht erfüllen
- Verteilte Systeme würden nicht funktionieren

In jedem Fall hat ein nicht funktionierendes Autorisierungssystem für ein Unternehmen finanzielle Folgen, im schlimmsten Fall kommen sogar Imageschaden oder rechtliche Konsequenzen hinzu.

Damit ein *PDP* die richtigen Entscheidungen treffen kann, darf er nur anwendbare Richtlinien zur Evaluation beziehen. Wird eine *authorization decision* mit nicht anwendbaren *Policies* getroffen, so ist der Entscheid unbrauchbar. Dies gilt auch, wenn anwendbare Richtlinien nicht zur Evaluation berücksichtigt werden.

Auch der *PEP* muss auf eine *authorization decision* richtig reagieren. Doch ist die Funktionstüchtigkeit des *PDPs* wichtiger, da der *PEP* eine falsche Entscheidung nicht korrigieren kann. Zudem ist die Logik für den *PEP* einfacher zu implementieren, muss er lediglich aufgrund des Autorisierungsentscheides einen Zugriff erlauben oder verweigern.

### 3.1.2 Performanz

Da ein *PDP* mehrere *PEPs* bedient, erhält er viele *authorization requests*, welche evaluiert werden müssen. Damit für einen Benutzer keine langen Wartezeiten entstehen, muss der *PDP* die Autorisierungsentscheidungen schnell treffen können. Zwei Faktoren beeinflussen die Geschwindigkeit:

- die Zeit für das Auffinden von anwendbaren Richtlinien
- die Zeit für das Auflösen von zusätzlichen Attributen

Strategien dazu werden im Kapitel 3.2.1 beschrieben.

### 3.1.3 Erreichbarkeit

Ein *PEP* ist zuständig für das Durchsetzen der *decision requests* vom *PDP* und soll keine Entscheidungen selbständig treffen. Fällt ein *PDP* aus, so sind davon meist mehrere *PEPs* betroffen. Die Folge davon ist, dass dabei gleich mehrere Anwendungen und Ressourcen nicht mehr erreichbar sind. Im Kapitel 3.2.2 werden mögliche Lösungen bezüglich Erreichbarkeit von *PDPs* behandelt.

### 3.1.4 Sicherheitsaspekte

Die Aufteilung des Autorisierungsprozesses in den *PDP* und den *PEP* führt zu zusätzliche Sicherheitsanforderungen. Werden die beiden Komponenten nicht auf dem gleichen Rechner ausgeführt, so müssen die *decision requests* und *authorization decisions* über das Intranet oder über das Internet verschickt werden. Dies bietet einige Angriffspunkte wie zum Beispiel:

- *decision requests* können verfälscht werden
- *authorization decisions* können manipuliert werden
- *decision requests* können abgefangen und nicht autoritative *authorization decisions* generiert und zurückgeschickt werden

Es existieren verschiedene Möglichkeiten, diese Probleme zu lösen. Diese haben aber meist Performanzeinbussen zur Folge. Welche Möglichkeiten wann Sinn machen, wird im Kapitel 3.2.3 diskutiert.

### 3.1.5 Kommunikation *PDP* zu *PDP*

Für eine verteilte Autorisierungslösung ist es erforderlich, dass *Policies* verteilt gespeichert werden können. Benötigt ein *PDP* Richtlinien zur Evaluation einer Autorisierungsanfrage, welche ihm nicht direkt zur Verfügung stehen, so muss er von einem anderen *PDP* die *Policies* anfordern.

Mehrere Möglichkeiten sind dabei denkbar, welche in Kapitel 3.2.4 erläutert werden.

Aufgrund der Erwartungen von *XACML* an eine richtlinienbasierte Autorisierungssprache resultieren weitere Anforderungen an einen *Policy Decision Point*:

### 3.1.6 Mehr-Werte Logik

*XACML* sieht für einen Autorisierungsentscheid zusätzlich zu *permit* und *deny* zwei weitere Zustände vor, welche entsprechend vom *PDP* unterstützt werden müssen [OASIS05a]:

**Zugriff erlauben (engl. *permit*):** Die Evaluation der anwendbaren *Policies* auf den *decision request* war positiv, der *PEP* erlaubt den Zugriff auf die geschützte Ressource.

**Zugriff verweigern (engl. *deny*):** Die Evaluation der anwendbaren *Policies* auf den *decision request* war negativ, der *PEP* muss den Zugriff auf die geschützte Ressource verweigern.

**Nicht ermittelbar (engl. *indeterminate*):** Während der Evaluation der anwendbaren *Policies* auf den *decision request* ist ein Fehler aufgetreten (zum Beispiel Attribute konnten nicht aufgelöst werden). Der *PDP* selbst kann keine Entscheidung treffen. Der Zugriff auf die geschützte Ressource darf in diesem Fall nicht erlaubt werden.

**Nicht anwendbar (engl. *not applicable*):** Der *PDP* hat keine anwendbaren *Policies* auf den *decision request* gefunden und kann daher auch keine Entscheidung treffen. Der *PEP* darf den Zugriff auf die geschützte Resource nicht erlauben.

### 3.1.7 Zusicherungen

*XACML* bietet die Möglichkeit, zusätzliche Auflagen (engl. *Obligations*) in *Policies* oder *PolicySets* einzufügen, welche von der *authorization decision* abhängig sind. Diese *Obligations* werden zusammen mit dem Entscheid (*permit* oder *deny*) an den *PEP* zurück geschickt. Der *PEP* muss diese *Obligations* verstehen und ausführen können, ansonsten darf er den Zugriff nicht erlauben. *Obligations* können Anweisungen enthalten, wie z. B. der Zugriff muss registriert oder eine Benachrichtigung muss per E-Mail verschickt werden.

Da die *Obligations* abhängig von der *authorization decision* sind, ist es die Aufgabe des *PDPs*, die entsprechend richtigen *Obligations* an den *PEP* zu schicken. Nach der Spezifikation von *XACML* geschieht das folgendermassen:

- Ist der Entscheid *indeterminate* oder *not applicable*, dann werden keine *Obligations* an den *PEP* geschickt.
- Wird die Evaluation des *PDPs* als Baum von *Policies* und *PolicySets* betrachtet, welche einzeln ein *permit* oder *deny* zurückliefern, so werden nur die *Obligations* an den *PEP* geschickt, welche an einem Pfad mit dem gleichem Gesamtentscheid des *PDPs* angegliedert sind und für den entsprechenden Entscheid bestimmt sind.

Zur Veranschaulichung sei der *Policy*-Baum in Abbildung 3.1 gegeben.

Der Gesamtentscheid der Evaluation ist *permit* (*PolicySet A1*). Der grüne Pfad entspricht den enthaltenen *Policies* und *PolicySets*, welche auch zu *permit* evaluieren. Daraus folgt, dass nur die *Obligation C2* an den *PEP* geschickt wird. Die *Obligation C4*, welche wohl für *permit* Entscheide bestimmt ist und mit der *Policy C4* verbunden ist, die auch zu *permit* evaluiert, wird nicht mitgeschickt, da das *PolicySet B2* zu *deny* evaluiert (roter Pfad). Die *Obligation B1* ist nur für *deny* Entscheide bestimmt und wird deshalb nicht an den *PEP* geschickt.

## 3.2 HERAS<sup>AF</sup> PDP

Der HERAS<sup>AF</sup> *Policy Decision Point* erfüllt alle Anforderungen eines *XACML PDPs*. Zusätzlich bietet er weitere Funktionalitäten an, welche für eine unternehmenstaugliche Autorisierungslösung entscheidend sind. In den folgenden Abschnitten werden diese Funktionalitäten beschrieben. Des Weiteren werden Lösungen zu den erwähnten Anforderungen aufgezeigt.

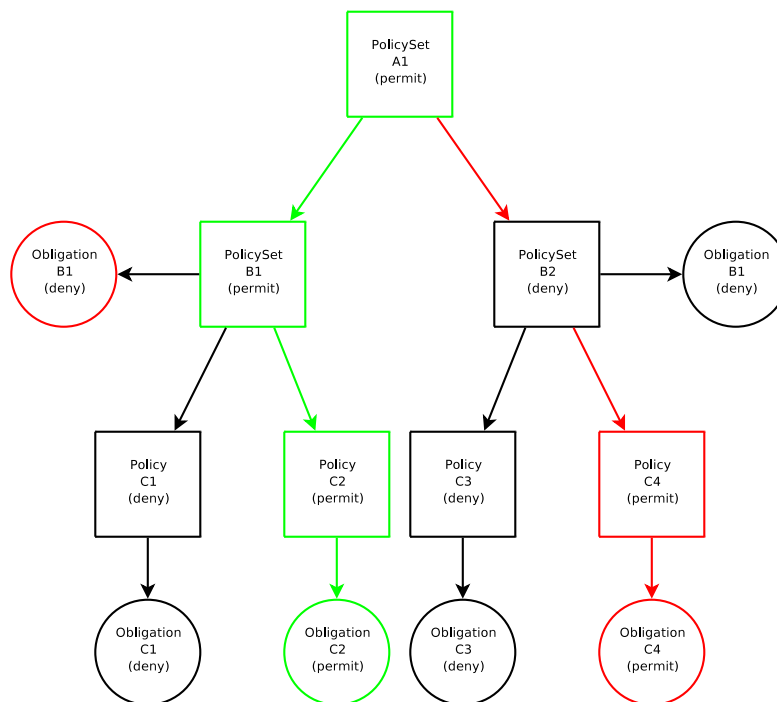


Abbildung 3.1: Obligations

### 3.2.1 Performanz

Wie bereits erwähnt ist die Performanz für die Autorisierung sehr wichtig. Da der *PDP* viele Anfragen erhält und evaluieren muss, sollte er auf einem leistungsfähigen Rechner installiert werden. Damit aber nicht unnötig Rechenleistung verbraucht wird, sind entsprechende Strategien nötig.

#### Indexierung

Ein entscheidender Faktor für die Performanz ist die Überprüfung der Anwendbarkeit von *Policies* und *PolicySets*. Muss der *PDP* für jeden *authorization request* alle ihm bekannten Richtlinien auf deren Anwendbarkeit überprüfen, so ist das sehr unperformant.

Da in einer *XACML Policy* oder in einem *PolicySet* das *Target*-Element über die Anwendbarkeit entscheidet, kann durch Indexierung der *Target*-Elemente und mittels geeigneten Suchabfragen eine kleinere Menge von Richtlinien für einen bestimmten *authorization request* dem *PDP* verfügbar gemacht werden.

Ein Ziel dieser Diplomarbeit war, eine sinnvolle Indexierung für Richtlinien zu finden. Nach längerer Betrachtung zeigte sich aber die Vielschichtigkeit dieser Aufgabe. Denn die *Target*-Elemente von Richtlinien enthalten oft keine

bestimmten Werte, sondern *Selectors* oder *Designators*<sup>1</sup>, welche zum Beispiel auf einen Wert in der angeforderten Ressource verweisen. Die Werte, welche sich aus den *Selectors* und *Designators* ergeben, sind also abhängig vom *authorization request* und sind daher beim Speichern einer Richtlinie nicht bekannt. Zudem lassen sich einige Funktionen von *XACML* nur sehr schwer in einer *SQL* Query ausdrücken.

### Serialisierung

Die Beschreibung von Richtlinien in Form von XML ist für die Portierbarkeit und Interoperabilität notwendig. Während des Autorisierungsprozesses geht dadurch aber wertvolle Zeit für das Parsen, Überprüfen und Validieren von *Policies* verloren. Anschliessend muss die *Policy* in ein Format übersetzt werden, welches der *Policy Decision Point* verarbeiten kann.

Wenn eine Richtlinie erfolgreich in ein entsprechendes *Policy*-Objekt übersetzt wurde, so kann davon ausgegangen werden, dass sie gültig ist. Wird das *Policy*-Objekt serialisiert gespeichert, so entfällt das Parsen, Überprüfen und Validieren und die Richtlinie kann dem *PDP* schneller verfügbar gemacht werden. Voraussetzung dafür ist, dass sich das *Policy*-Objekt serialisieren lässt.

Dies führt zu einem Nachteil: Serialisierte Richtlinien lassen sich nur mit der selben *PDP* Implementation verwenden. Wird die Implementierung geändert, so werden die serialisierten Richtlinien unbrauchbar.

Da ein verwendeter *PDP* nur selten ausgetauscht wird, ist das Serialisieren unter dem Gesichtspunkt der Performanz vertretbar. Zudem sind die *Policies* nicht verloren, sie können aus dem HERAS<sup>AF</sup> *PAP* neu "deployed" oder vom proprietären *Policy*-Objekt in XML Form ausgegeben werden.

### Caching

Durch das Zwischenspeichern von *Policies*, *PolicySets* sowie referenzierte Attribute kann die Performanz des *PDPs* gesteigert werden. Dies ist primär bei *Policy*- und *PolicySet*-Referenzen vernünftig, da diese nicht für jeden *authorization request* über das Netz übertragen werden müssen.

Die Dauer des Caching darf aber nicht zu lange sein. Sonst besteht die Gefahr, dass ein *PDP* Entscheidungen trifft, welche nicht autoritativ sind, da sich die ursprüngliche Richtlinie geändert hat. Aus diesem Grund lässt sich die Dauer im HERAS<sup>AF</sup> *PDP* konfigurieren.

### Clustering

Der HERAS<sup>AF</sup> *Policy Decision Point* ist als J2EE Applikation realisiert und wird in einem J2EE Container ausgeführt. Performanzoptimierungen lassen sich

---

<sup>1</sup>*Selectors* und *Designators* referenzieren *decision request* abhängige Werte in einer *Policy*.

auf Ebene des Containers, z. B. durch *SLB Clustering*<sup>2</sup> realisieren. Hierfür sind für den HERAS<sup>AF</sup> PDP keine weiteren Anpassungen notwendig.

### 3.2.2 Erreichbarkeit

Fällt ein einzelner PDP aus, so sind meist mehrere Applikationen und Ressourcen davon betroffen. Aus diesem Grund ist die Erreichbarkeit von PDPs ein wichtiger Aspekt für die Unternehmenstauglichkeit einer Autorisierungslösung.

Der HERAS<sup>AF</sup> PDP wird in einem J2EE Container ausgeführt. Die Container unterstützen meist ein *HA Clustering*<sup>3</sup>, welches eine hohe Verfügbarkeit des *Policy Decision Points* garantiert.

Eine zusätzliche Möglichkeit, einen Ausfall eines PDPs zu überbrücken, ist eine entsprechende Konfiguration der HERAS<sup>AF</sup> PEPs. Dazu können einem PEP mehrere PDPs angegeben werden, welche redundante Funktionalität anbieten. Erhält ein PEP vom ersten PDP keine Antwort, so schickt er die Autorisierungsanfrage an den nächsten. Voraussetzung dafür ist, dass die verschiedenen PDPs auf den selben *Policy-Store* zugreifen, respektive die selben *Policies* zur Verfügung haben.

### 3.2.3 Sicherheitsaspekte

HERAS<sup>AF</sup> bietet Technologien an, um den Austausch von Nachrichten zwischen den Komponenten vor möglichen Attacken zu schützen.

So kann die Kommunikation über eine geschützte *SSL*<sup>4</sup> Verbindung erfolgen. Dadurch sind die übertragenen Daten vor Manipulationen und Einsicht Dritter geschützt. Ein Nachteil dieser Lösung ist, dass der Verbindungsaufbau relativ lange dauert (der Datenaustausch über eine bestehende Verbindung ist anschliessend verhältnismässig schnell). Zudem kann nur eine Punkt-zu-Punkt Verbindung verschlüsselt werden. Führt die Kommunikation über mehrere Rechner, so wird der Einsatz von SSL umständlicher.

Aus diesem Grund stellt HERAS<sup>AF</sup> eine weitere Technologie zur Verfügung, welche unabhängig vom Transportprotokoll ist. Durch den Einsatz von *Webservice Security (WSS)* können die Nachrichten zwischen den Komponenten über mehrere Rechner signiert und verschlüsselt verschickt werden. Das *Simple Object Access Protocol (SOAP)* bietet die Grundlage für *WSS*.

Die Kommunikation zwischen den Komponenten PEP, PDP und PAP über das lokale Netz oder das Internet bietet folgende Angriffspunkte:

**Manipulation des Autorisierungsentscheides:** Kann ein Benutzer die Autorisierungsentscheide eines PDPs an den PEP abfangen und manipulieren, so kann er sich jeden Zugriff erlauben.

Der *authorization decision* muss zwingend *signiert* werden.

---

<sup>2</sup>Server Load Balancing (dt. Serverlastverteilung)

<sup>3</sup>High-Availability (dt. Hochverfügbarkeit)

<sup>4</sup>Secure Socket Layer

**Manipulation der Autorisierungsanfrage:** Durch Manipulation von Autorisierungsanfragen können Entscheide von einem *PDP* beeinflusst und dementsprechend falsche Zugriffserlaubnis erreicht werden.

Möchte zum Beispiel ein Benutzer auf eine geschützte Ressource schreibend zugreifen, besitzt auf diese aber nur Leserechte, so kann er in der Autorisierungsanfrage die Aktion von schreiben auf lesen ändern. Da der Benutzer Leserechte besitzt, wird der Entscheid vom *PDP* ein *permit* sein worauf der *PEP* den Zugriff erlaubt.

Der *authorization request* muss zwingend *signiert* werden.

**Mitlesen von Informationen:** Oft enthalten Autorisierungsanfragen ergänzende Informationen über den Benutzer, wie zum Beispiel die Rolle. Zusätzlich kann durch das Mitlesen von *authorization requests* und den entsprechenden *authorization decisions* herausgefunden werden, welche Benutzer welche Rechte auf welche Ressourcen hat.

Ist es nicht erwünscht, diese Informationen preiszugeben, so müssen die Anfragen und Antworten *verschlüsselt* werden.

**Umleiten von *authorization requests*:** Gelingt es einem Angreifer, *authorization requests* von *PEPs* auf einen eigenen *PDP* umzuleiten, so kann er jede gewünschte *authorization decision* generieren und an den *PEP* zurück schicken. Dadurch erhält ein Angreifer die gesamte Kontrolle über den Autorisierungsprozess für mehrere *PEPs*. Der *PEP* muss also garantieren können, dass die Antwort auf eine Anfrage autoritativ ist.

Daher ist das *Signieren* von *authorization decisions* zwingend nötig.

Dasselbe gilt auch für das Auflösen einer *Policy*- oder *PolicySet*-Referenz zwischen zwei *PDPs*. Wird beim Auflösen einer Referenz ein falscher *PDP* angesprochen und entsprechend eine manipulierte Richtlinie zurückgeschickt, so kann der Autorisierungsentscheid verändert werden.

Aus diesem Grund muss auch die Antwort auf eine Anfrage nach einer Richtlinie *signiert* sein.

**Manipulation einer *Policy*-Referenz:** Muss ein *PDP* eine Referenz bei einem anderen *PDP* auflösen, so können durch Manipulation der Antwort eigene Richtlinien hinzugefügt und dadurch der Autorisierungsentscheid verfälscht werden.

Das Auflösen einer *Policy*-Referenz muss *signiert* werden.

***Policies* deployen:** Ein weiterer Angriffspunkt besteht beim Deployen von *Policies*. Da der *Policy Administration Point* meistens nicht auf dem selben Rechner wie der *PDP* ausgeführt wird, stellt HERAS<sup>AF</sup> im *PDP* einen Webservice für das Deployment von Richtlinien zur Verfügung. Dabei muss sichergestellt werden, dass die *Policies* beim Deployen nicht verändert werden.

Aus diesem Grund müssen die Richtlinien beim Deployen *signiert* werden. Zudem muss garantiert werden können, dass nur autorisierte Benutzer *Policies* deployen können.

Damit ein sicheres Autorisierungssystem realisiert werden kann, müssen *alle* Nachrichten, welche zwischen den Komponenten ausgetauscht werden, signiert sein. Eine Verschlüsselung der Kommunikation ist nur nötig, wenn verhindert werden soll, dass Benutzerinformationen und Zugriffsrechte preisgegeben wer-

den.

### 3.2.4 Kommunikation *PDP* zu *PDP*

*PolicySets* können Referenzen auf *Policies* oder weitere *PolicySets* beinhalten. Diese Referenzen werden durch eine *URI* bezeichnet und weisen auf Richtlinien hin, die in einem entfernten *PDP* gespeichert sind.

Wird für eine Referenz eine *URL* verwendet, so ist der Speicherort der Richtlinie durch die *URL* definiert. Der *PDP* muss Richtlinien mit Hilfe der *URL* auflösen können. Da *URLs* den Speicherort statisch angeben, hat das Verschieben einer referenzierten Richtlinie eine Anpassung von sämtlichen *PolicySets* zur Folge, welche die entsprechende *Policy* respektive das entsprechende *PolicySet* referenzieren.

Eine flexiblere Variante bietet der Einsatz von *URNs*. Diese referenzieren eine Richtlinie unabhängig vom Speicherort. Jeder *PDP* muss die *URNs* eindeutig auf eine Richtlinie abbilden können. Dabei sind zwei Topologien denkbar.

- Jeder *PDP* löst die *URNs* selbständig auf. Dies bedingt, dass jeder *PDP* entsprechend konfiguriert werden muss. Daraus ergibt sich eine *Point-to-Point* Architektur, wie sie in Abbildung 3.2 gezeigt ist. Ändert sich aber der Speicherort für bestimmte Richtlinien, so muss dies bei jedem *PDP* entsprechend neu konfiguriert werden.

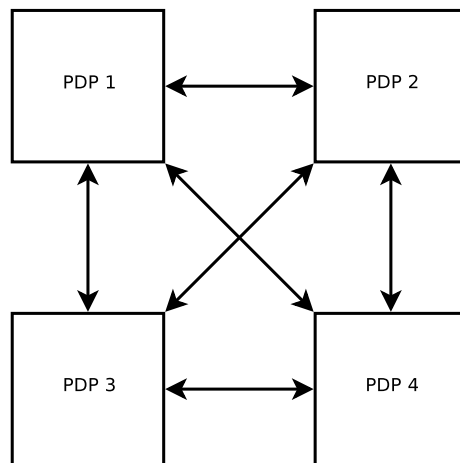


Abbildung 3.2: *Point-to-Point* Architektur

- Damit das Verschieben von Richtlinien an einen neuen Speicherort möglichst kleine Anpassungen zur Folge hat, empfiehlt sich der Einsatz einer zentralen Komponente (*HUB*), welche die Zuordnung einer *URN* zu dem Speicherort einer Richtlinie übernimmt. Dadurch kann flexibler auf eine Verschiebung einer referenzierten Richtlinie reagiert werden, da entsprechende Konfigurationen nur an einer Stelle vorgenommen werden müssen. Diese Topologie wird oft in *Service Oriented Architecture (SOA)*

verwendet und ist als *Hub-and-Spoke* oder Sternarchitektur bekannt. Abbildung 3.3 illustriert eine derartige Sternarchitektur.

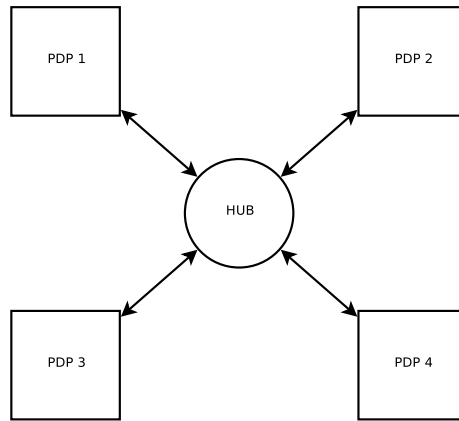


Abbildung 3.3: *Hub-and-Spoke* Architektur

# Architektur

Der HERAS<sup>AF</sup> *PDP* besteht aus den Modulen *heras-com*, *heras-int* und *heras-pdp*.

*heras-com* realisiert dabei die Kommunikation zwischen den Komponenten *PEP*, *PAP* und *PDP*. Das Modul *heras-int* stellt die Funktionalität des *PDPs* als Webservices zur Verfügung. Der HERAS<sup>AF</sup> *PDP* wird durch das Modul *heras-pdp* realisiert.

Die Trennung erlaubt die Implementation des *PDPs* unabhängig von der Kommunikation und Methode, wie darauf zugegriffen wird. Durch konsequente Verwendung des *Spring IoC<sup>1</sup>-Containers* bestehen keine direkten Abhängigkeiten zwischen den einzelnen Modulen.

## 4.1 Module

### 4.1.1 *heras-com*

Das Modul *heras-com* realisiert die Kommunikation zwischen den Modulen *PEP*, *PDP* und *PAP*. Dafür werden sowohl *authorization requests* und *authorization decision* als auch Anfragen für Richtlinien und die Richtlinie selbst (Antwort) in eine *SAML* Nachricht verpackt. Diese wiederum wird in einer *SOAP* Meldung verschickt. Abbildung 4.1 zeigt den schematischen Aufbau.

Durch diese Mehrfachverschachtelung können die Vorteile der einzelnen Sprachen genutzt werden:

### ***SOAP***

Das *Simple Object Access Protocol* abstrahiert die Transportschicht, wodurch sich das verwendete Transportprotokoll (in unserem Fall ist das *http*) austauschen lässt.

---

<sup>1</sup>Inversion of Control

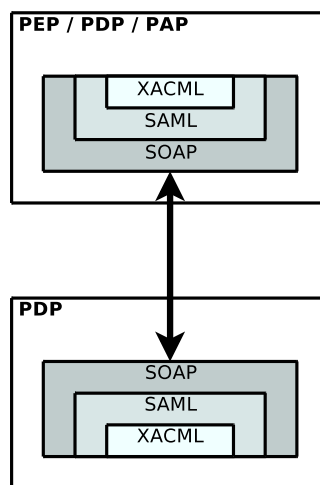


Abbildung 4.1: PDP zu PDP Kommunikation

### **SAML**

Mit Hilfen von *Security Assertion Markup Language* lassen sich zusätzliche Funktionen realisieren:

- Single Sign On
- Verteilte Transaktionen
- Authentifizierungsdienste

#### **4.1.2 heras-int**

Dieses Modul bietet die Funktionalität des PDPs als Webservices an. Durch den Einsatz von Webservices kann der PDP beliebig verteilt werden. Zusätzlich können so auch PEPs, die nicht in Java implementiert sind, die Funktionalitäten des HERAS<sup>AF</sup> PDPs verwenden.

Für das Auffinden der einzelnen PDPs empfiehlt sich die Verwendung von UDDI<sup>2</sup>.

#### **4.1.3 heras-pdp**

Die zentrale Komponente für die Autorisierung ist das Modul *heras-pdp*, welches die Aufgaben des *Policy Decision Points* übernimmt. Der HERAS<sup>AF</sup> PDP wird üblicherweise über die im Kapitel 4.1.2 erwähnten Webservices angesprochen. Wird der PDP in der selben Java VM ausgeführt, wie z. B. der PAP, so kann er auch direkt verwendet werden, ohne die Webservices zu nutzen.

<sup>2</sup>Universal Description, Discovery and Integration

Die wesentliche Aufgabe des HERAS<sup>AF</sup> *PDPs* ist das Evaluieren von *authorization requests* auf anwendbare Richtlinien. Diese Aufgabe in der aktuellen Version an den *PDP* der quelloffenen *Sun's XACML* Implementation, welche unter der Leitung von Seth Proctor entstand, delegiert. Der *Sun XACML PDP* verwendet für das Auffinden von anwendbaren Richtlinien und Attribute entsprechende *FinderModule*, welche beim Auswerten eines *authorization requests* aufgerufen werden. Das Resultat der *FinderModule* wird für die Evaluation weiterverwendet.

Die *FinderModule* sind abhängig von Speicherort und -art der entsprechenden Daten. Werden Richtlinien in einer Datenbank gespeichert, so muss ein entsprechendes *PolicyFinderModule* implementiert werden. Dasselbe gilt für die *AttributeFinderModules* und *ResourceFinderModules*.

Beinhaltet ein *PolicySet* Referenzen auf weitere *Policies* oder *PolicySets*, die in einem anderen *PDP* gespeichert sind, so muss ein entsprechender *PolicyRequestor* die Möglichkeit haben, diese aufzulösen. Dafür bietet der HERAS<sup>AF</sup> *PDP* eine entsprechende Funktionalität, durch welche Richtlinien mittels *ID* angefordert werden können.

Zusätzlich müssen die Richtlinien, welche einem *PDP* direkt zur Verfügung stehen, administriert werden können. Im Wesentlichen beinhaltet die Administration das Deployen und entfernen von Richtlinien. Diese Funktionalität wird ausschliesslich vom *Policy Administration Point* verwendet.

Obwohl der HERAS<sup>AF</sup> Prototyp die *Sun's XACML* Implementation für das Evaluieren von Richtlinien verwendet, soll diese Abhängigkeit möglichst klein sein. Dies ist vor allem für selbst entwickelte Klassen entscheidend, welche abhängig von firmeninternen Daten, wie zum Beispiel zusätzliche Benutzerinformationen, entwickelt wurden. Aus diesem Grund bietet HERAS<sup>AF</sup> eine Abstraktion, die so genannten *Requestors*, welche eine Realisierung für das Auffinden von Richtlinien, Attributen und Ressourceinformationen unabhängig der verwendeten *XACML* Implementation erlaubt. Die *Requestors* werden im Kapitel 4.3 erläutert.

## 4.2 ContextHandler

Der *ContextHandler* ist verantwortlich dafür, dass Autorisierungsanfragen, die nicht in *XACML* formuliert sind, in *XACML decision requests* umzuformen.

Ist die native Anfrage durch ein *XML*-Schema definiert, so kann die Transformation mittels *XSLT*<sup>3</sup> in eine *XACML* Autorisierungsanfrage erfolgen. In allen anderen Fällen muss die Transformation explizit implementiert werden.

Eine weitere Aufgabe des *ContextHandlers* beinhaltet das Auflösen von Attributen.

HERAS<sup>AF</sup> verzichtet auf den Einsatz eines *ContextHandlers* als eigenständige Komponente. Die Transformation in einen *XACML decision request* ist abhängig vom verwendeten *PEP* und deshalb nicht generell beschreibbar.

<sup>3</sup>Extensible Stylesheet Language Transformation

Das Auffinden von Attributen erfolgt analog zum Finden von Richtlinien und wird durch entsprechende *Requestors* erreicht (siehe Kapitel 4.3).

### 4.3 *Requestors*

Die *Requestors* wurden aus der Architektur von der *Sun's XACML* Implementation, mit dem Ziel davon unabhängig zu sein, abgeleitet. Für die drei Typen von *FinderModulen* (*PolicyFinderModule*, *AttributeFinderModule*, *ResourceFinderModule*) bietet der HERAS<sup>AF</sup> *PDP* entsprechende *Requestors* an:

- *PolicyRequestor* für das Auffinden von Richtlinien aufgrund eines *decision requests*, sowie das Auflösen von referenzierten Richtlinien.
- *AttributeRequestor* für das Auflösen von Attributen. Die *AttributeRequestors* werden für alle Attributtypen (*Subject*-, *Resource*-, *Action*- und *Environment*-Attribute) verwendet.
- *ResourceRequestor* für das Auflösen von hierarchischen Ressourcen. Die *ResourceRequestors* erlauben den Zugriff auf mehrere Ressourcen, sofern diese in einer einfachen Hierarchie gespeichert sind.

Die *Requestors* werden als Liste mittels *Spring IoC* an eine *XACML* implementationsabhängige Klasse übergeben, in unserem Fall an ein entsprechendes *FinderModule* der *Sun's XACML* Implementation. Dies bietet den Vorteil, dass realisierte *Requestors* nach einem Austausch der *XACML* Implementation ohne Änderungen weiter verwendet werden können. Dabei müssen nur die *FinderModule* neu implementiert und mittels *Spring IoC* dem *PDP* zur Verfügung gestellt werden.

Jeder *Requestor* unterstützt verschiedene Aufgaben. So kann zum Beispiel ein *PolicyRequestor* geschrieben werden, der Richtlinien aus einer Datenbank liest, ein weiterer *PolicyRequestor* ist dafür verantwortlich, Referenzen auf Richtlinien im selben *PDP* aufzulösen und ein dritter kann Referenzen auf Richtlinien, die in einem entfernten *PDP* gespeichert sind, zur Verfügung stellen.

Gleichermassen werden zum Beispiel ein *LdapSubjectAttributeRequestor* für das Auffinden von zusätzlichen Informationen über das Subjekt aus einem *LDAP*<sup>4</sup> Verzeichnis und ein weiterer *CurrentDateAttributeRequestor* für das aktuelle Datum entwickelt.

Die Anzahl der *Requestors* in einem *PDP* kann also beliebig gross werden. Für die Evaluation eines *decision requests* macht es dabei aber wenig Sinn, jeden einzelnen *Requestor* aufzurufen, zum Beispiel den oben erwähnten *LdapSubjectAttributeRequestor*, wenn das aktuelle Datum erforderlich ist. Dies hätte eine schlechte Performanz zur Folge.

Aus diesem Grund werden *RequestorSelectorStrategies* eingesetzt, Strategien, welche auf einen *decision request* entsprechend sinnvolle *Requestors* auswählen. Im eben erwähnten Beispiel würde so nur der *CurrentDateAttributeRequestor* aufgerufen, nicht aber der *LdapSubjectAttributeRequestor*.

<sup>4</sup>Lightweight Directory Access Protocol

Für den Fall, dass mehrere *Requestors* aufgerufen werden sollen, zum Beispiel zum Finden von Richtlinien in einer Datenbank (*DbPolicyRequestor*) und im Dateisystem (*FileSystemPolicyRequestor*), können die *RequestorSelector-Strategies* auch mehrere *Requestors* auswählen.

Auf diese Weise können auch Fallback-Lösungen realisiert werden, zum Beispiel für das Auswählen eines weiteren *PolicyRequestors*, nachdem ein erster eine referenzierte *Policy* nicht auflösen konnte.

## 4.4 Kommunikation *PDP* zu *PDP*

Die Kommunikation zwischen zwei *PDPs* wird für das Auflösen von referenzierten Richtlinien benötigt. Dabei wird das im Kapitel 4.1.1 beschriebene Kommunikationsmodul verwendet.

Findet der *PDP* in einem *PolicySet* eine Referenz auf eine Richtlinie, so delegiert er das Auflösen an einen entsprechenden *PolicyRequestor* weiter. Dessen Aufgabe ist es, anhand der *ID* der Richtlinie die referenzierte *Policy* oder *PolicySet* zu finden. Dabei nutzt er den entsprechenden Webservice eines anderen *PDPs*, sofern die Referenz nicht eine Richtlinie im selben *PDP* referenziert. Der Ablauf für das Auflösen von referenzierten Richtlinien wird in Abbildung 4.2 gezeigt und ist in folgende Schritte aufgeteilt:

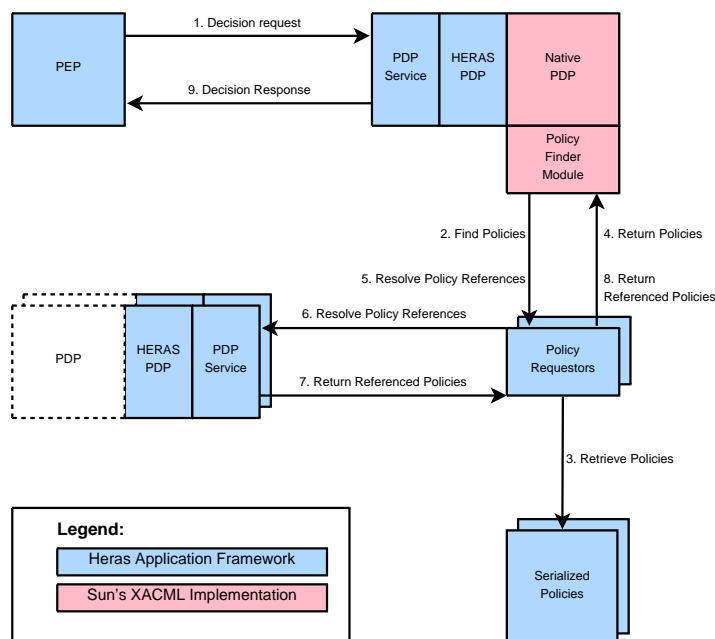


Abbildung 4.2: *PolicyRequestor*

1. Der *PEP* schickt einen *decision request* an den *PDP*. Dazu nutzt er den entsprechenden Webservice des *PDPs*. Die Anfrage wird an den verwendeten *Sun XACML PDP* weitergeleitet.

2. Der *Sun XACML PDP* sucht über das *PolicyModule* nach Richtlinien. Dabei werden die *PolicyRequestors* von HERAS<sup>AF</sup> aktiv.
3. Die *PolicyRequestors* suchen nach *Policies*.
4. Die *PolicyRequestors* geben die gefundenen Richtlinien an den *Sun XACML PDP* für die Evaluation weiter.
5. Enthält ein *PolicySet* Referenzen auf weitere Richtlinien, so werden erneut die HERAS<sup>AF</sup> *PolicyRequestors* aktiv
6. Die *PolicyRequestors* fordern die referenzierten *Policies* von entfernten *PDPs* an. Die Anfrage nach Richtlinien erfolgt mittels *SAML* und der Verwendung einer *XACMLPolicyQuery*.
7. Entfernte *PDPs* schicken die angeforderten Richtlinien mittels *SAML* in einem *XACMLPolicyStatement* an die entsprechenden *PolicyRequestors* zurück.
8. Die *PolicyRequestors* geben die angeforderten *Policies* an den *Sun XACML PDP* zur weiteren Evaluation weiter.
9. Die evaluierte *decision response* wird an den *PEP* zurück geschickt.

Die beiden erwähnten Elemente *XACMLPolicyQuery* und *XACMLPolicyStatement* sind in [OASIS05b] beschrieben.

Wie im letzten Kapitel erörtert wurde, werden *Policy*- oder *PolicySet*-Referenzen durch *URIs* definiert. Der Wert dieser *URI* entspricht dem *PolicyId*- oder *PolicySetId*-Attribut, welches beim Erstellen der Richtlinie angegeben wird.

*XACML* lässt als *ID* auch eine *URL* zu, welche direkt auf eine Richtlinie im Dateisystem oder im Web zeigt. Das direkte Referenzieren mit einer *URL* ist aber nicht zu empfehlen. Wird eine Richtlinie verschoben, so müssen alle *PolicySets* überarbeitet werden, welche eine Referenz auf die entsprechende *Policy* enthalten. Zudem lassen sich Informationen bezüglich der Version der Richtlinie nur schwer abbilden.

## Strukturierte *IDs*

Die Vergabe von *PolicyId* und *PolicySetId* ohne Struktur macht das Auffinden von referenzierten Richtlinien schwierig und dadurch unperformant. Bietet die *ID* keine Informationen darüber, in welchem *PDP* sich die referenzierte Richtlinie befindet, muss in allen bekannten *PDPs* nach der Richtlinie gesucht werden. Zudem ist die Eindeutigkeit einer *PolicyId* oder *PolicySetId* über mehrere *PDPs* kaum kontrollierbar, was beim Auflösen einer Referenz zu fehlerhaften *PolicySets* führen kann.

Um das Problem zu umgehen, in jedem *PDP* nach einer *PolicyId* oder *PolicySetId* zu suchen, kann eine zentrale Komponente hinzugefügt werden, welche die Abbildung von einer *ID* zu dem entsprechenden *PDP* realisiert. Dafür sind weitere Mechanismen nötig, die zum Beispiel beim Deployen die entsprechenden Informationen an diese zentrale Komponente liefert. Die Komponente muss

stets aktuelle Daten enthalten, da ansonsten Referenzen nicht aufgelöst werden können.

Um das Auffinden von referenzierten Richtlinien zu vereinfachen, verwendet HERAS<sup>AF</sup> für die *IDs URNs*, welche sich aus mehreren Informationen zusammensetzen. Ein Teil dieser Information besteht zum Beispiel aus einer Kennung der Richtlinie, welche manuell angegeben werden kann, aber im verwendeten *PDP* eindeutig sein muss.

Ein weiterer Teil der *URN* enthält die Identifikation des *PDPs*. Dadurch ergibt sich folgende denkbare Form der *PolicyId* und *PolicySetId*:

*urn:<pdp\_name>:<policy\_id>*

Es besteht weiterhin die Möglichkeit, zusätzliche Informationen wie zum Beispiel der Name des Unternehmens hinzuzufügen. Dies macht primär dann Sinn, wenn Richtlinien über Unternehmensgrenzen hinaus verwendet werden. Derartige Änderungen lassen sich leicht durch Anpassung der *PDP* Konfiguration und Hinzufügen eines entsprechenden *PolicyRequestors* realisieren.

Durch die zusätzliche Information über den *PDP* in der *URN* lassen sich Richtlinien einfacher auffinden. Stösst ein *PDP* bei der Evaluation eines *authorization requests* in einem *PolicySet* auf eine Referenz, so delegiert er das Auflösen an einen *PolicyRequestor*, welcher durch eine entsprechende Strategie bestimmt wird. Dieser erkennt anhand der *URN*, in welchem *PDP* die referenzierte Richtlinie gespeichert ist und fordert diese an.

Der Name des *PDPs* soll dabei beschreibend sein und nicht eine statische Information wie zum Beispiel die IP Adresse oder *URL* beinhalten, da sich diese ändern könnte. Die Abbildung vom beschreibenden Namen des *PDPs* auf die *URL* oder IP Adresse des *PDPs* kann mit einem entsprechenden *PolicyRequestor* realisiert werden.

Kommt es häufig zu Änderungen der Adressen von *PDPs*, so ist es umständlich, entsprechende *PolicyRequestors* in allen *PDPs* anzupassen. Hierfür empfiehlt sich die Einführung einer zusätzlichen zentralen Komponente mittels *JNDI*<sup>5</sup>, die das Abbilden vom Namen des *PDPs* auf eine *URL* oder IP Adresse übernimmt. So müssen Anpassungen nur an der zentralen Komponente vorgenommen werden, und nicht in jedem einzelnen *PolicyRequestor*.

#### 4.4.1 Verteilte Evaluierung

Die *XACML* Spezifikation beschreibt, dass sich ein *PolicySet*, welches eine Referenz enthält, bei der Evaluation gleich verhält, als ob die referenzierte Richtlinie im entsprechenden *PolicySet* enthalten wäre. Während dieser Arbeit wurden diesbezüglich Überlegungen für eine verteilte Evaluation gemacht. Anstatt eine referenzierte Richtlinie aufzulösen und in das *PolicySet* einzubeziehen, ist folgendes denkbar:

Der *authorization request* wird an den *PDP* übermittelt, in welchem die referenzierte Richtlinie gespeichert ist. Zusätzlich wird die *ID* der referenzier-

<sup>5</sup>Java Naming and Directory Interface

ten Richtlinie mit dem *authorization request* mitgeschickt. Der angesprochene *PDP* evaluiert den *authorization request* nur auf die referenzierte Richtlinie und schickt die *authorization decision* an den anfangs angesprochenen *PDP* zurück. Dieser fügt dem ursprünglichen *PolicySet* nur die *authorization decision* des zweiten *PDPs* hinzu.

Dadurch kann eine Verteilung des Autorisierungsentscheides erreicht werden. Jeder *PDP* evaluiert nur diejenigen Richtlinien, welche ihm direkt bekannt sind.

Dies hätte einen weiteren Vorteil: Einem *PDP* müssten nur noch die *AttributeRequestors* bekannt sein, die entsprechend referenzierte Attribute in den eigenen Richtlinien auflösen können. Denn bei der Evaluation einer referenzierten Richtlinie kann es vorkommen, dass das Auflösen von Attributen scheitert, weil ein entsprechender *AttributeRequestor* fehlt. Folgedessen müssten alle verwendeten *AttributeRequestors* jedem *PDP* zur Verfügung stehen.

Evaluiert aber ein *PDP* nur seine eigenen Richtlinien und lässt referenzierte Richtlinien von den entsprechenden *PDPs* evaluieren, so ist dies nicht nötig.

Allerdings müsste der *PDP* zwei weitere Funktionen unterstützen. Er müsste die Evaluation eines *authorization request* auf *nur eine bestimmte* Richtlinie unterstützen und bei der Evaluation eines *PolicySets* eine bereits getroffene *authorization decision* verarbeiten können.

## 4.5 Administration

Das Administrieren von Richtlinien, die einem *PDP* direkt bekannt sind, erfolgt über den *PAP*. Dazu gehören das Deployen und Entfernen von Richtlinien.

Die Abbildung 4.3 zeigt die Komponenten, die für das Deployen wichtig sind. Da die Richtlinien sinnvollerweise in serialisierter Form gespeichert werden, wird auch der native *PDP* für diesen Prozess verwendet.

Für das Deployen einer Richtlinie erstellt der *PAP* aus einer HERAS<sup>AF</sup> *Policy* eine *XACML Policy* und sendet diese an den Deploy-Service eines HERAS<sup>AF</sup> *PDPs*. Dieser serialisiert die empfangene Richtlinie mit Hilfe des nativen *PDPs* und speichert sie zum Beispiel in eine Datenbank.

## 4.6 Persistenz

Das Auffinden von persistenten Richtlinien ist ein entscheidender Faktor für einen schnellen Autorisierungsprozess.

Eine Persistierung von Richtlinien in Form von XML ist nicht optimal, da für jede Anfrage an den *PDP* die Richtlinien geparkt und in ein für den *PDP* verständliches Format gewandelt werden müssen.

Der *PDP* muss für jede Anfrage des *PEPs* die Anwendbarkeit der ihm bekannten Richtlinien überprüfen. Je weniger Richtlinien dem *PDP* für eine be-

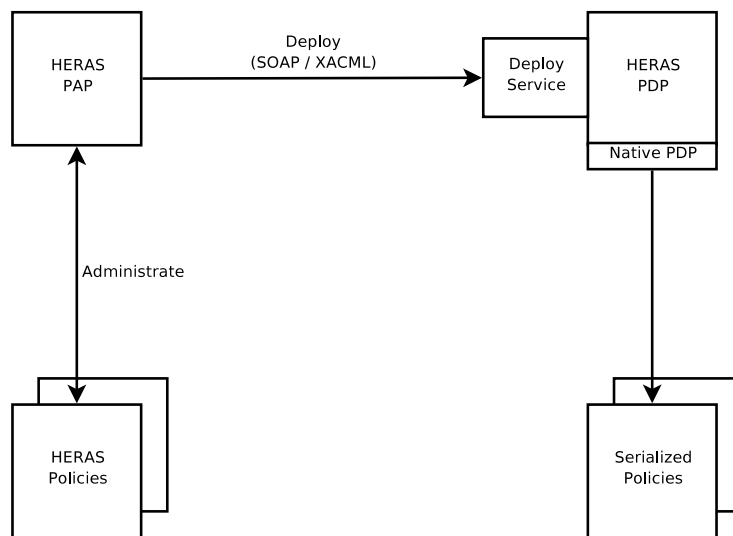


Abbildung 4.3: Policy Deployment

stimmte Anfrage zur Evaluation verfügbar sind, umso schneller kann er den Autorisierungsentscheid an den *PEP* zurückschicken. Aus diesem Grund müssen *PolicyRequestors* eine möglichst kleine Auswahl von Richtlinien finden. Diese Auswahl sollte möglichst wenig nicht anwendbare Richtlinien beinhalten. Gleichzeitig dürfen keine anwendbaren Richtlinien von den *PolicyRequestors* übersehen werden.

Die Anwendbarkeit einer Richtlinie ist durch dessen *Target*-Elemente (*Subject*, *Resource*, *Environment* und *Action*) definiert. Daher können anwendbare *Policies* und *PolicySets* mit Hilfe einer geeigneten Indexierung auf die *Target*-Elemente schnell gefunden werden.

Durch die Wahl einer schnellen Persistenzstrategie und einer geeigneten Indexierung kann das Auffinden von anwendbaren Richtlinien zusätzlich beschleunigt werden.

#### 4.6.1 Persistenzstrategien

Richtlinien können auf verschiedene Arten gespeichert werden. Aufgrund ihrer *XML* Struktur ist die trivialste Persistierung als *XML* Datei im Dateisystem denkbar. Fehlende Indexierung und beschränkte Suchmöglichkeiten machen diese Variante für eine grosse Anzahl von Richtlinien aber unbrauchbar. Geeigneter ist das Speichern von Richtlinien in Verzeichnisdienste oder relationalen Datenbanken.

##### Verzeichnisdienste

Verzeichnisdienste bieten die Möglichkeit, Daten hierarchisch zu speichern. Zudem sind sie auf schnelles Suchen und Lesen optimiert. Dadurch erfüllen sie die

wichtigsten Anforderungen an das Auffinden von persistenten Richtlinien.

Für einen optimierten Zugriff wird ein geeignetes Schema benötigt. Der Aufbau eines solchen Schemas ist umfangreich und würde den Rahmen dieser Diplomarbeit sprengen. Deshalb wurde auf die Persistierung mit Hilfe von Verzeichnisdiensten verzichtet.

[CoPa02] beschreibt eine mögliche Abbildung von Richtlinien in *LDAP*. Das Dokument hat seit Juli 2002 den Status *Draft*, was ein Indiz für die Komplexität einer derartigen Abbildung ist.

### Relationale Datenbanken

Relationale Datenbanken ermöglichen durch die relationale Strukturierung von Daten vielfältige Abfragemöglichkeiten. Dies ist für das Auffinden von anwendbaren Richtlinien ein grosser Vorteil. Dazu muss aber eine geeignete Struktur für das Speichern der Richtlinien definiert werden.

Durch die Unterstützung von *BLOB*<sup>6</sup> Daten können Richtlinien serialisiert in der Datenbank gespeichert werden und müssen so für die Evaluation nicht mehr geparkt und in ein entsprechendes *Policy*-Objekt gewandelt werden.

Leider lassen sich die Richtlinien von der *Sun's XACML* Implementation nicht serialisieren, so dass derzeit noch die XML Struktur als *BLOB* gespeichert wird.

#### 4.6.2 Derby

Der mit dieser Arbeit entstandene HERAS<sup>AF</sup> *PDP* verwendet für das Persistieren von Richtlinien die *Derby* Datenbank von *Apache*. Dadurch dass *Derby* sehr klein ist, wird sie direkt in den *PDP* eingebunden. Auf diese Weise entfällt die Kommunikation zu einem Datenbankserver. Zudem greift ausschliesslich der *PDP* auf die Datenbank zu, wodurch Lese- und Schreibsperrern ausgeschaltet werden können. Beide Tatsachen wirken sich positiv auf die Performanz aus.

Aus den genannten Gründen wird keine Indexierung auf die *Target*-Elemente realisiert.

---

<sup>6</sup>Binary Large Object

# Design und Realisierung

Aufgrund der Framework-Architektur werden hohe Anforderungen an das Design und die Implementierung von HERAS<sup>AF</sup> gestellt. Die Komponenten und Module sollen möglichst stabil und flexibel laufen sowie eine einfache Anwendbarkeit bieten.

Neben allgemein bekannten Anforderungen an das Softwaredesign sind folgende Ziele definiert:

- Abhängigkeiten zwischen Paketen sollen minimal sein.
- Das Programmieren gegen konkrete Klassen ist nur innerhalb abhängiger Strukturen erlaubt, ansonsten wird gegen Interfaces programmiert.
- Das Erzeugen von neuen Instanzen mit dem Schlüsselwort *new* ist höchstens paketintern erlaubt. Andernfalls wird der *Spring IoC Container* für die Erzeugung von Instanzen verwendet.
- Keine zyklischen Abhängigkeiten zwischen Paketen.
- Klassen werden als *POJOs*<sup>1</sup> realisiert.

Im Folgenden wird die Struktur und Implementation des *HerasPDPs* beschrieben, welcher zu dieser Arbeit entstanden ist. Sofern nicht anders erwähnt, entstand alles aus eigener Arbeit.

## 5.1 *Spring IoC Container*

Der *HerasPDP* wird vollständig über die *Spring IoC Container* Konfiguration erstellt. Dabei werden die benötigten Klassen vom *Spring IoC Container* instanziiert und zusammengefügt. Die entsprechende Konfiguration wird in einer *XML*-Datei vorgenommen.

Durch die Verwendung von *Inversion of Control* kann das *Dependency Injection* Pattern umgesetzt und so die genannten Anforderungen und Ziele an

---

<sup>1</sup>Plain Old Java Object

das Softwaredesign von HERAS<sup>AF</sup> eingehalten werden. Der folgende, unvollständige Auszug aus der verwendeten Konfiguration zeigt, wie der *Sun XACML PDP* mit einem *PolicyRequestor* erstellt wird.

---

```

1 <beans>
2
3 <!--=====
4   SunXacml PDP
5   =====>
6 <bean id="sunxacmlPdp" class="com.sun.xacml.PDP">
7   <constructor-arg>
8     <ref bean="sunxacmlPdpConfig" />
9   </constructor-arg>
10 </bean>
11
12 <bean id="sunxacmlPdpConfig"
13   class="com.sun.xacml.PDPConfig">
14
15   <constructor-arg>
16     <ref bean="sunxacmlAttributeFinder" />
17   </constructor-arg>
18   <constructor-arg>
19     <ref bean="sunxacmlPolicyFinder" />
20   </constructor-arg>
21   <constructor-arg>
22     <ref bean="sunxacmlResourceFinder" />
23   </constructor-arg>
24 </bean>
25
26 <!--=====
27   SunXacml Policy Finder
28   =====>
29 <bean id="sunxacmlPolicyFinder"
30   class="com.sun.xacml.finder.PolicyFinder">
31   <property name="modules">
32     <set>
33       <ref bean="sunxacmlPolicyFinderModule" />
34     </set>
35   </property>
36 </bean>
37
38
39 <!--=====
40   Integration
41   =====>
42 <bean id="sunxacmlPolicyFinderModule"
43   class="org.herasaf.pdp.integration.
44     sunxacml.SunXacmlPolicyFinderModule">
45   <property name="policyRequestors">
46     <list>

```

```

47         <ref
48             bean="derbyPolicyRequestor" />
49     </list>
50 </property>
51 </bean>
52
53
54 <!--=====
55     Policy Requestors
56     =====>
57 <bean id="derbyPolicyRequestor"
58     class="org.herasaf.pdp.requestor.
59     impl.DerbyPolicyRequestor">
60 </bean>
61 </beans>

```

---

## 5.2 Packages

Die Funktionalität für den *HerasPDP* ist in verschiedene Pakete aufgeteilt:

**org.herasaf.pdp** ist das Kern-Paket für die *PDP* Funktionalität und beinhaltet den *HerasPDP*, der als Wrapper für den nativen *PDP* dient. Zusätzlich findet sich hier ein Interface, welches für das Administrieren von Richtlinien im *PDP* bestimmt ist. Ein weiteres Interface beschreibt das Aussehen von Strategien, die für die Wahl von geeigneten *Requestors* auf einen gegebenen Request verwendet werden.

**org.herasaf.pdp.impl** beinhaltet Implementationen aus dem Paket *org.herasaf.pdp*.

**org.herasaf.pdp.requestor** enthält die *Requestor*-Interfaces, welche die Abstraktion von der verwendeten *XACML* Implementation ermöglichen. Zusätzlich beinhaltet dieses Paket entsprechende *Result*-Typen, die von den *Requestors* zurück geliefert werden.

**org.herasaf.pdp.requestor.impl** enthält Implementationen der *Requestor*-Interfaces.

**org.herasaf.pdp.integration.sunxacml** bietet die Integration der verwendeten *XACML* Implementation. In unserem Fall sind dies entsprechende *Sun XACML Module*, welchen die *Requestors* zur Verfügung gestellt werden.

## 5.3 Package Design

Die Abbildung 5.1 zeigt vereinfacht die verschiedenen Pakete und deren Zusammenhang mit der *Sun's XACML* Implementation.

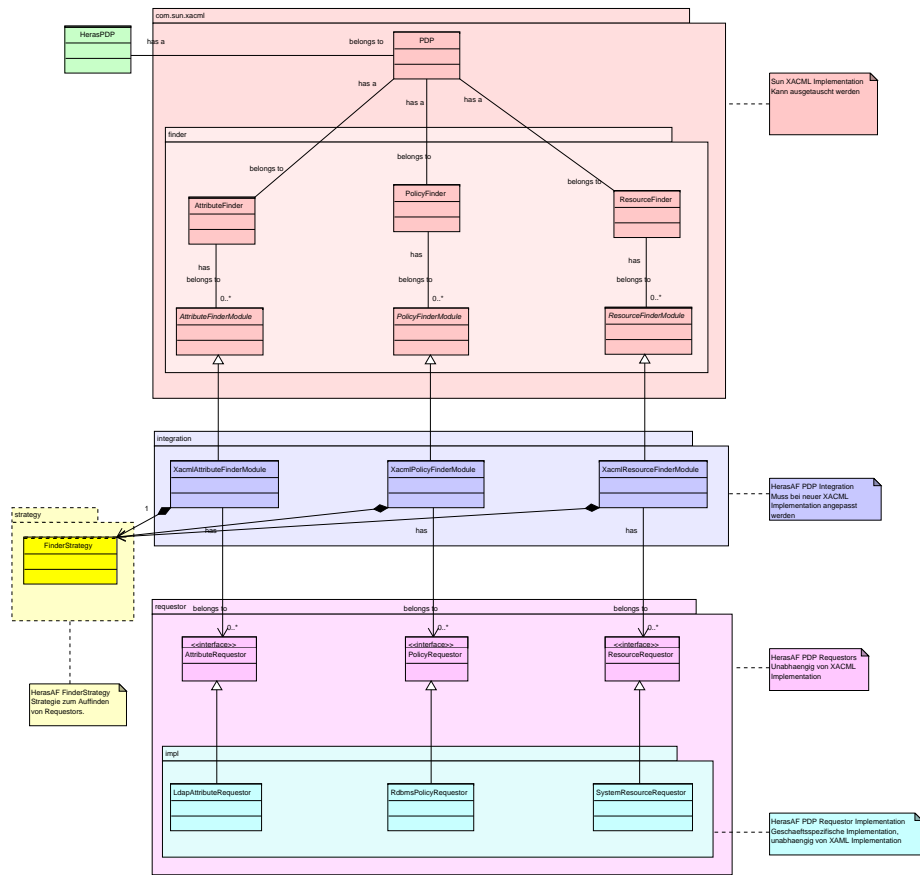


Abbildung 5.1: Package Design

Der grün gefärbte *HerasPDP* hält eine bestimmte *XACML* Implementation, in unserem Fall ist das die *Sun's XACML* Implementation, an welchen die *authorization requests* delegiert werden.

Der *Sun XACML PDP* verfügt über je ein *PolicyFinder*, *AttributeFinder* und *ResourceFinder*, welche ihrerseits entsprechende *FinderModule* enthalten. Neue *FinderModule* können für eigene Bedürfnisse implementiert und den *Finders* hinzugefügt werden. Die *FinderModule* sind *Sun XACML* spezifisch und können daher für eine andere *XACML* Implementation nicht mehr verwendet werden.

Aus diesem Grund bietet HERAS<sup>AF</sup> entsprechende *Requestors* aus dem Paket *org.herasaf.pdp.requestors*. Diese werden über die *XacmlAttributeFinderModule*, *XacmlPolicyFinderModule* und *XacmlResourceFinderModule* dem *Sun XACML PDP* über den *Spring IoC Container* zur Verfügung gestellt. Wird die Implementation von *XACML* ausgetauscht, so müssen nur die *FinderModule* neu implementiert werden, nicht aber die bestehenden *Requestors*.

Um eine sinnvolle Auswahl eines *Requestors* abhängig von *authorization requests* zu treffen, können entsprechende *RequestorSelectorStrategies* den *FinderModules* bereitgestellt werden.

## 5.4 Klassen

Dieses Kapitel beschreibt die wichtigsten Klassen und Interfaces, die von mir implementiert wurden.

**org.herasaf.pdp.HerasPdp** ist die Wrapperklasse um den nativen *PDP*.

Die Hauptaufgaben sind das Weiterleiten der *RequestContexts* und das Zurückschicken des *ResponseContexts*. Zusätzlich bietet der *HerasPdp* Zugriff auf die Objekte *RequestCtxFactory*, *ResponseCtxFactory* und *PolicyAdmin*.

**org.herasaf.pdp.RequestorSelectorStrategy** ist eine abstrakte Klasse zum Realisieren einer Strategie, um auf eine gegebene Anfrage entsprechende *Requestors* auszuwählen. Diese werden in den *SunXacmlAttribute*-, *SunXacmlResource*- und *SunXacmlPolicyFinderModule* verwendet. Über die *newInstance()* Methode wird eine einfache Realisierung einer Strategie erzeugt, welche alle verfügbaren *Requestors* auswählt.

**org.herasaf.pdp.PolicyAdmin** definiert eine Schnittstelle für das Administrieren von Richtlinien in einem *PDP*. Über den *HerasPdp* kann auf das entsprechende *PolicyAdmin* Objekt zugegriffen werden.

**org.herasaf.pdp.impl.DerbyPolicyAdmin** ist eine Realisierung von *PolicyAdmin* und erlaubt die Administration von Richtlinien in einer *Apache Derby* Datenbank.

**org.herasaf.pdp.integration.sunxacml.SunXacmlPolicyFinderModule** realisiert die Integration von *Sun's XACML PDP* in das HERAS<sup>AF</sup> *PDP* Komponente. Diese Klasse ist als *PolicyFinderModule* aus der *Sun's*

*XACML* Implementation realisiert und delegiert Anfragen an entsprechende *PolicyRequestors* weiter.

**org.herasaf.pdp.integration.sunxacml.SunXacmlAttributeFinderModule** realisiert die Integration von *Sun's XACML PDP* in das HERAS<sup>AF</sup> *PDP* Komponente. Diese Klasse ist als *AttributeFinderModul* aus der *Sun's XACML* Implementation realisiert und delegiert Anfragen an entsprechende *AttributeRequestors* weiter.

**org.herasaf.pdp.integration.sunxacml.SunXacmlResourceFinderModule** realisiert die Integration von *Sun's XACML PDP* in das HERAS<sup>AF</sup> *PDP* Komponente. Diese Klasse ist als *ResourceFinderModul* aus der *Sun's XACML* Implementation realisiert und delegiert Anfragen an entsprechende *ResourceRequestors* weiter.

**org.herasaf.pdp.requestor.Requestor** ist ein übergeordnetes Interface für alle *Requestor*-Typen. Es hat bisher noch keine Aufgabe, wurde aber für mögliche Erweiterungen bereits definiert.

**org.herasaf.pdp.requestor.PolicyRequestor** ist ein Interface und beschreibt den Zugriff auf die *PolicyRequestors*.

**org.herasaf.pdp.requestor.AttributeRequestor** ist ein Interface und beschreibt den Zugriff auf die *AttributeRequestors*.

**org.herasaf.pdp.requestor.ResourceRequestor** ist ein Interface und beschreibt den Zugriff auf die *ResourceRequestors*.

**org.herasaf.pdp.requestor.PolicyRequestorResult** speichert das Resultat einer Suche nach Richtlinien.

**org.herasaf.pdp.requestor.AttributeRequestorResult** speichert das Resultat einer Suche nach Attributen.

**org.herasaf.pdp.requestor.ResourceRequestorResult** speichert das Resultat einer Suche nach Ressourcen.

**org.herasaf.pdp.requestor.impl.DerbyPolicyRequestor** realisiert einen *PolicyRequestor* mittels der *Apache Derby* Datenbank. Der *DerbyPolicyRequestor* realisiert ein simples Caching, so dass nicht für jede Anfrage auf die Datenbank zugegriffen werden muss.

**org.herasaf.pdp.requestor.impl.DateTimeEnvironmentRequestor** realisiert einen *AttributeRequestor* und liefert aktuelle Zeitinformationen.

**org.herasaf.pdp.helpers.DerbyFactory** ist eine Hilfsklasse für den Zugriff auf die *Apache Derby* Datenbank und wird von der Klasse *DerbyPolicyAdmin* und *DerbyPolicyRequestor* verwendet.

## 5.5 Änderungen von HERAS<sup>AF</sup>

Der gesamte Autorisierungsprozess verwendet das Interface *org.smurve.heras.context.EvaluationCtx* für den Zugriff auf die Autorisie-

rungsanfrage. Dieses Interface besitzt derzeit aber noch keine Funktionalität, wodurch einige Klassen nicht fertig realisiert werden konnten.

Eine Möglichkeit, um aus einer XML-Richtlinie eine HERAS<sup>AF</sup>-Richtlinie zu erstellen habe ich leider vergebens gesucht. Aber abgesehen davon waren für die Realisierung des *PDPs* keine Änderungen am bestehenden Modell von HERAS<sup>AF</sup> nötig.



# Zusammenfassung und Ausblick

## 6.1 Zusammenfassung

Das Ziel dieser Diplomarbeit war das Beschreiben von Anforderungen an einen verteilten und unternehmenstauglichen *Policy Decision Point (PDP)* sowie die Erarbeitung entsprechender Lösungen. Zusammengefasst können folgende Ergebnisse hervorgehoben werden:

- Das Bereitstellen der *PDP* Funktionalität mittels Webservices für den Einsatz in *service-orientierten Architekturen (SOA)*.
- Eine Lösung für das Verteilen und Auffinden von Sicherheitsrichtlinien durch Verwendung von beschreibenden Bezeichnungen und entsprechenden Funktionalitäten.
- Eine Schnittstelle für das Realisieren eigener Module zum Auffinden von zusätzlichen Informationen unabhängig der verwendeten *XACML* Implementation.
- Strategien und Lösungen für Performanzoptimierungen während der Autorisierung.
- Der Einsatz von *Webservice Security* für die Sicherheit der Autorisierungslösung.
- Die Persistierung von Richtlinien mit der *Apache Derby* Datenbank.
- Hohe Interoperabilität, Erweiterbarkeit und Austauschbarkeit der Komponenten durch Unterteilung der Aufgaben (Kommunikation, Funktionalität, ...) in verschiedene Module.

Jedoch konnten nicht alle Ziele erreicht werden:

- Auf die Indexierung von persistierten Sicherheitsrichtlinien musste verzichtet werden, da ihr Einbezug in diese Arbeit den zeitlichen Rahmen gesprengt hätte.

- Die Integration des realisierten *Policy Decision Points* in den bestehenden HERAS<sup>AF</sup> Prototypen konnte nicht umgesetzt werden.

## 6.2 Ausblick

Aufgrund der stetig wachsenden Bedeutung von *SOA* sind verteilte Autorisierungslösungen ein wichtiges Thema. HERAS<sup>AF</sup> wird daher fortlaufend weiterentwickelt.

Ein erster Schritt ist die Integration des verteilten *PDPs* in die bestehende HERAS<sup>AF</sup> Architektur. In einer weiteren Phase sind zusätzliche Realisierungen denkbar:

- die Indexierung von *XACML* Richtlinien.
- die Entwicklung einer *Policy Administration Points* als *Eclipse Rich Client Platform (RCP)* Applikation.
- der Zugriffsschutz auf Klassen und Methoden mittels *Aspect Oriented Programming (AOP)*.

# Literaturverzeichnis

- [AAND05] Anne Anderson (2005): A Comparison of Two Privacy Policy Languages: EPAL and XACML. Internet: <http://research.sun.com/techrep/2005/abstract-147.html> (01.12.2006).
- [CoPa02] Cover Pages (2002): OASIS eXtensible Access Control Markup Language (XACML). Internet: <http://xml.coverpages.org/xacml-schema-policy-v15.pdf> (25.11.2006).
- [DoPe06] W. Dobmeier, G. Pernul (2006): Modellierung von Zugriffsrichtlinien für offene Systeme. Universität Regensburg.
- [EGG06] René Eggenschwiler (2006): HERAS<sup>AF</sup>. Manageable Policy-based access control for J2EE. Diplomarbeit an der Hochschule für Technik, Rapperswil.
- [IETF00] IETF (2000): A Framework for Policy-based Admission Control. Internet: <http://www.ietf.org/rfc/rfc2753.txt> (20.11.2006).
- [IETF01] IETF (2001): Terminology for Policy-Based Management. Internet: <http://www.ietf.org/rfc/rfc3198.txt> (20.11.2006).
- [NIST06] National Institute of Standards and Technology (2006): Role Based Access Control. Internet: <http://csrc.nist.gov/rbac> (12.12.2006).
- [OASIS05a] Oasis (2005): eXtensible Access Control Markup Language (XACML) Version 2.0. Internet: [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) (12..2006).
- [OASIS05b] OASIS (2005): SAML 2.0 profile of XACML v2.0. Internet: [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) (12.12.2006).
- [Secunet] secunet Security Networks AG: Policies und Richtlinien. Internet: [http://www.secunet.com/content.php?text=k\\_sicherheitsmanagement\\_policies](http://www.secunet.com/content.php?text=k_sicherheitsmanagement_policies) (05.11.2006).
- [Sun05] Sun Microsystems (2005): XACML: Access Control, Under Control. Internet: [http://research.sun.com/spotlight/2005\\_11\\_01-XACML.html](http://research.sun.com/spotlight/2005_11_01-XACML.html) (15.11.2006).



# Internetadressen

**EPAL** <http://www.w3.org/Submission/EPAL>

**IETF** <http://www.ietf.org>

**OASIS** <http://www.oasis-open.org>

**P3P** <http://www.w3.org/P3P>

**SAML** [www.oasis-open.org/committees/security](http://www.oasis-open.org/committees/security)

**SOAP** <http://www.w3.org/TR/soap>

**SPML** [www.oasis-open.org/committees/provision](http://www.oasis-open.org/committees/provision)

**Spring** <http://www.springframework.org>

**Sun Microsystems** <http://www.sun.com>

**Sun's XACML Implementation** <http://sunxacml.sourceforge.net>

**Verbreitung von XACML** <http://docs.oasis-open.org/xacml/xacmlRefs.html>

**W3C** <http://www.w3.org>

**XACML** <http://www.oasis-open.org/committees/xacml>



# Abkürzungen

**ACL** Access Control List

**BLOB** Binary Large Object

**DSS** Digital Signature Standard

**EPAL** Enterprise Privacy Authorization Language

**HA Clustering** High-Availability Clustering (dt. Hochverfügbarkeit)

**HERAS<sup>AF</sup>** Holistic Enterprise Ready Application Security Architecture Framework

**IETF** Internet Engineering Task Force

**IoC** Inversion of Control

**J2EE** Java 2 Platform, Enterprise Edition

**Java VM** Java Virtual Machine

**JNDI** Java Naming and Directory Interface

**LDAP** Lightweight Directory Access Protocol

**PAP** Policy Administration Point

**PDP** Policy Decision Point

**PEP** Policy Enforcement Point

**PIP** Policy Information Point

**PKI** Public Key infrastructure

**POJO** Plain Old Java Object

**P3P** Platform for Privacy Preferences

**PBAC** Policy Based Access Control

**RBAC** Role Based Access Control

**SAML** Security Assertion Markup Language

- SLB** Server Load Balancing (dt. Serverlastverteilung)
- SOA** Service Oriented Architecture
- SOAP** Simple Object Access Protocol
- SPML** Service Provisioning Markup Language
- SSL** Secure Socket Layer
- UDDI** Universal Description, Discovery and Integration
- URI** Uniform Resource Identifier
- URL** Uniform Resource Locator
- URN** Uniform Resource Locator
- W3C** World Wide Web Consortium
- WSS** Webservice Security
- XACML** Extensible Access Control Markup Language
- XML** Extensible Markup Language
- XSLT** Extensible Stylesheet Language Transformation