INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Diplomarbeit

Konzeption und Implementierung einer policy-basierten Privacy Management Architektur für föderierte Identitätsmanagementsysteme am Beispiel Shibboleth

Matthias Ebert

Aufgabensteller: Prof. Dr. Heinz-Gerd Hegering

Betreuer: Wolfgang Hommel
Latifa Boursas

Abgabetermin: 31. Dezember 2006

INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Diplomarbeit

Konzeption und Implementierung einer policy-basierten Privacy Management Architektur für föderierte Identitätsmanagementsysteme am Beispiel Shibboleth

Matthias Ebert

Aufgabensteller: Prof. Dr. Heinz-Gerd Hegering

Betreuer: Wolfgang Hommel
Latifa Boursas

Abgabetermin: 31. Dezember 2006

Hiermit versichere ich, dass ich die vorliegende Diplomarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

München, den 31. Dezember 2006

. . . . . .

(Unterschrift des Kandidaten)

Aufgabe des Identitätsmanagements ist die organisationsinterne und zentrale Verwaltung von Daten über Mitarbeiter, Kunden, Partner und Gäste (kurz Benutzer) sowie deren Zugriff auf lokale Dienstleisungen und Ressourcen. Diese herkömmliche Lösung des Identitätsmanagements deckt jedoch weder die steigende Notwendigkeit des Zugriffs auf organisationsexterne Dienstleistungen und Ressourcen noch eine immer wichtiger werdende effiziente Benutzerverwaltung von Partnerunternehmen (z.B. im B2B-Bereich) ab. Die Frage ist also, wie organisationsinterne sowie -externe Informationssysteme und deren Benutzer in Domänengrenzen überschreitenden Netzwerken integriert bzw. verwaltet werden können. Genau hiermit beschäftigt sich das föderierte Identitätsmanagement (FIM).

In föderierten Identitätsmanagementsystemen müssen zur Authentifizierung und Autorisierung benötigte Benutzerdaten zwischen verschiedenen Instanzen (z.B. zwischen zwei Organisationen, wenn Benutzer einer Organisation Dienste einer anderen Organisation nutzen möchten) ausgetauscht werden. Um den Austausch von Benutzerinformationen in kontrolliertem Rahmen ablaufen zu lassen, werden Attribute Release Policies (ARPs) eingesetzt. Mit Hilfe der ARPs lassen sich Richtlinien erstellen, in denen die zu übertragenden Daten eingeschränkt werden können. Indem Benutzer bzw. Administratoren durch ARPs regeln können, welche Informationen, z.B. zum Zweck der Autorisierung, freigegeben werden dürfen und welche nicht, wird der wichtige Aspekt des Datenschutzes gewährleistet. Für das Konzept der ARPs gibt es jedoch noch keine Standardisierungsansätze.

Eine vom Konsortium Internet2 entwickelte, relativ weit verbreitete und schon im praktischen Einsatz befindliche Open Source Software für föderiertes Identitätsmanagement ist Shibboleth. Shibboleth implementiert jedoch nur proprietäre ARPs, deren Ausdrucksfähigkeit nur für einfache Szenarien ausreicht.

In dieser Diplomarbeit werden, nach einer Einführung und Beschreibung von Shibboleth, Szenarien im Bereich des föderierten Identitätsmanagements betrachtet. Aus den Szenarien werden in einer Anforderungsanalyse Kriterien, welche ein ARP-System unterstützen muss, abgeleitet. Mit dem daraus entwickelten Kriterienkatalog werden bekannte Policysprachen auf ihre Eignung für ARPs hin untersucht. Die am besten passende Policysprache wird für die zu entwickelnde Architektur ausgewählt und deren Verwendung im Detail beschrieben. Auf Basis der Policysprache wird eine neue policybasierte Privacy Management Architektur für Shibboleth entwickelt und anschließend prototypisch in Java implementiert. Bei der Implementierung wird auf einen bereits vorhandenen Policy Decision Point von Sun zurückgegriffen. Des Weiteren wird mit Beispielen beschrieben, wie das neue System konfiguriert und angewandt wird. Zum Schluss wird dargestellt, welche Zielsetzungen erreicht wurden und welche Ansatzpunkte für weiterführende Arbeiten vorhanden sind.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einführung

Als Motivation wird zu Beginn der Arbeit ein Überblick über die Entwicklungsgeschichte des Identitätsmanagements gegeben. Des Weiteren wird die Aufgabenstellung und die Struktur dieser Arbeit beschrieben.

1.1 Motivation

Die Geschichte des Identitätsmanagements kann in die drei Entwicklungsstufen „Systeme ohne Identitätsmanagement“, „Systeme mit Identity und Access Management“ und „Föderiertes Identitätsmanagement“ eingeteilt werden. Im Folgenden werden neben den jeweiligen Funktionalitäten und Eigenschaften der einzelnen Entwicklungsstufen auch dessen Probleme dargestellt.

1.1.1 Systeme ohne Identitätsmanagement

Über das Internet werden verschiedenste geschützte Dienstleistungen angeboten. Betrachtet man als Beispiel die Ludwig Maximilians Universität München, kurz LMU, so gibt es dort eine Vielzahl von web-basierten Diensten. Diese Dienste liegen dezentral verteilt, d.h. jede Fakultät bietet ihre eingenen zugangsgeschützten Dienste und Ressourcen an. Beispiele hierfür sind der „ifkw.extra“ Dienst der Fakultät Kommunikationswissenschaft, welcher eine Übersicht über die bisherigen Leistungsnachweise bietet, oder das fakultätsübergreifende Internetportal „CampusLMU“ , welches umfassende und strukturierte Informationen rund um das Studium bietet. Die Dienste erfordern jeweils die Eingabe von Benutzerdaten, um sich erfolgreich anmelden und diese nutzen zu können. Nutzt man mehrere Dienste, so müssen auch mehrfache Benutzerkonten angelegt werden. Es entstehen für jeden Benutzer eine Vielzahl von unterschiedlichen Accounts, welche durch kein Managementsystem miteinander verbunden sind. Hier den Überblick zu behalten ist oft nicht einfach. Probleme eines solchen Szenarios sind:

- Redundanz der Benutzerdaten

Neben Benutzername und Passwort werden auch zusätzliche Informationen, wie z.B. die Adresse eines Benutzers, mehrfach an unterschiedlichen Orten abgelegt. Daraus folgt Redundanz und es kann ein Sicherheitsrisiko entstehen, da viele Nutzer in ihren unterschiedlichen Konten meist die gleichen Kennungsdaten benutzen. Erhält ein unbefugter Benutzer Zugriff auf eines dieser Konten, so steht ihm oft auch die Tür zu anderen Diensten offen.

- Inkonsistenz der Benutzerdaten

Müssen Benutzerinformationen, wie zum Beispiel die E-Mail Adresse oder der Wohnort aktualisiert werden, so muss dies an mehreren Stellen geschehen. Der Aufwand der Datensynchronisation ist erheblich. Es besteht die Gefahr, dass die Daten inkonsistent werden, wenn die Datenpflege nicht ordentlich betrieben wird.

- Datenqualität nicht gewährleistet

Es gibt keine Regelungen zum einheitlichen Schutz der Datenqualität. Benutzerinformationen können fehlerhaft und unvollständig sein.

1.1.2 Systeme mit Identity und Access Management

Systeme mit Identity und Access Management bekommen diese Probleme in den Griff. An der Einrichtung des Benutzers, hier die LMU, wird eine zentrale Datenbank angelegt, in der alle Benutzerinformationen abgelegt sind. Durch die Zentralisierung können die Daten leicht geprüft und aktualisiert werden. Authentifiziert sich ein Benutzer einmal an zentraler Stelle, so kann er daraufhin alle angebotenen Dienste nutzen, was als Single Sign-On bezeichnet wird. In diesem Szenario reicht die Authentifizierung, also das „Wer bin ich?“, jedoch nicht aus. Es muss auch die Frage des „Was darf ich?“ geklärt werden. Nicht alle Benutzer der LMU dürfen z.B. auf interne Informationen der Fakultät der Kommunikationswissenschaft zugreifen. Diesen Sachverhalt klärt die Autorisierung. Dazu werden Benutzerinformationen (wie Fakultät des Benutzers) untersucht, woraufhin eine Zugriffsentscheidung getroffen werden kann.

Der Nachteil an diesem System ist jedoch, dass es nur innerhalb einer Organisation funktioniert. Will ein Benutzer auch organisationsexterne Dienste nutzen, zum Beispiel von der Technischen Universität München, so benötigt er dort ein eigenes Benutzerkonto. Um dieses Problem zu lösen, wurde das sog. föderierte Identitätsmanagement eingeführt.

1.1.3 Föderiertes Identitätsmanagement

Gesucht sind also Lösungen, die einen organisationsübergreifenden Zugriff auf Dienstleistungen ermöglichen, ohne dafür mehrfache Benutzerkonten zu benötigen. Nach einmaliger Authentifizierung bei der eigenen Einrichtung soll der Benutzer neben den organisationsinternen Diensten auch organisationsfremde Dienste nutzen können. Dazu wird eine zweistufige Architektur verwendet. Die Einrichtung, in der die Benutzer verwaltet und authentifiziert werden, nennt man Identity Provider. Die Einrichtung, die Dienste anbietet und den Zugang zu diesen Diensten durch Autorisierung kontrolliert, nennt man Service Provider [Shib Architecture]. Will ein Benutzer der LMU einen Dienst der TU nutzen, so authentifiziert sich dieser zuerst am Identity Provider (LMU). Nach erfolgreicher Authentifizierung muss der Service Provider, also die TU, den Benutzer für den gewünschten Dienst „nur“ noch autorisieren. Eine Authentifizierung ist an der TU nicht mehr nötig. Ist auch die Autorisierung erfolgreich abgelaufen, so wird dem Benutzer der Zugriff auf den Dienst gewährt. Solche Systeme werden föderierte Identitätsmanagement (FIM) Systeme genannt. Eine Einrichtung kann Identity Provider und Service Provider zugleich sein, indem sie Benutzerdaten verwaltet und auch Web-Dienstleistungen anbietet. Durch die Aufteilung in Identity Provider und Service Provider ist die Benutzerverwaltung von den Dienstleistungen abgekoppelt. So können neue Organisationen Dienste bereitstellen, ohne sich um die Benutzerverwaltung kümmern zu müssen. Auf der anderen Seite ist die Benutzerverwaltung auch unabhängig von den angebotenen Dienstleistungen.

Die in föderierten Identitätsmanagement Systemen zur Authentifizierung und Autorisierung benötigten Benutzerdaten werden zwischen den Identity- und Service-Providern über dedizierte Protokolle wie SAML [SAML], Security Assertion Markup Language, ausgetauscht. Der Umfang der dabei zu übertragenden Daten wird durch ein geeignetes policy-basiertes Privacy Management System eingeschränkt. Dies ist Aufgabe des Identity-Providers und wird mit Hilfe der sogenannten Attribute Release Policies (ARPs) gelöst. Ein Benutzer muss also beispielsweise angeben können, welche seiner Informationen er an wen (z.B. Herausgabe nur an TU), zu welchem Zweck (z.B. zur Autorisierung),

unter welchen Bedingungen/Conditions (z.B. Kursdaten nur herausgeben, wenn Note besser als 2 ist) mit welchen Auflagen/Obligations (z.B. Daten müssen beim Service Provider nach Semesterende wieder gelöscht werden), herausgibt. All diese Eigenschaften sollen in den ARPs geregelt werden können.

Auf der anderen Seite (Service-Provider) muss angegeben werden können, welche Informationen für die Freigabe eines Dienstes benötigt werden. Dies wird in den Attribute Acceptance Policies (AAPs) niedergeschrieben.

Die Ressourcenfreigabe in föderierten Identitätsmanagement Systemen wird also durch Policymechanismen gesteuert. Es müssen Entscheidungen getroffen und durchgesetzt werden. Beide Handlungen werden logisch in die Komponenten „Policy Decision Point“ (PDP) und „Policy Enforcement Point“ (PEP) getrennt.

Der „Policy Decision Point“ ist die Instanz, in der Policy Entscheidungen, also ob der Benutzer autorisiert ist oder nicht, getroffen werden. Dazu werden die Benutzerattribute mit zutreffenden Policies analysiert. Das Ergebnis wird an den PEP weitergereicht.

Der „Policy Enforcement Point“ ist die Instanz, der die Implementierung zur Umsetzung der vom PDP getroffenen Policy Entscheidung enthält. Der PEP gewährt oder verhindert anhand der Entscheidung des PDP die Freigabe der Resourcen für einen Benutzer. (Siehe Abbildung 1.1).

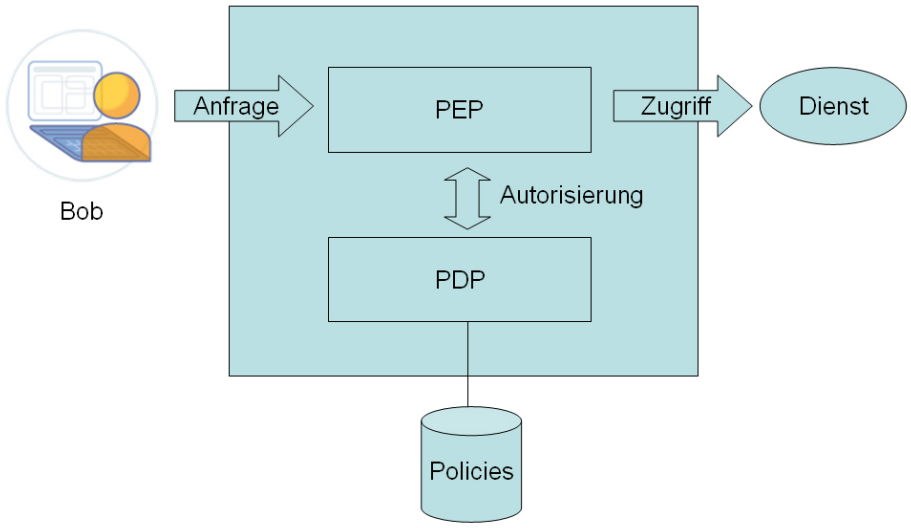

Abbildung 1.1: Policy Decision und Enforcement

Schließen sich mehrere Organisationen zur gemeinsamen Nutzung eines FIMs zusammen, so nennt man dies eine Föderation. Die Organisationen innerhalb einer Föderation werden auch Relying Party genannt. Innerhalb einer Föderation müssen Übereinkünfte über eine gemeinsame Syntax und Semantik zum Informationsaustausch getroffen werden. Daneben müssen Verträge abgeschlossen werden, die z.B. regeln, wer zu was in welchem Umfang berechtigt ist.

1.2 Aufgabenstellung

Shibboleth implementiert proprietäre ARPs, deren Ausdrucksfähigkeit für viele Szenarien des FIM nicht ausreicht. Im Rahmen dieser Diplomarbeit sollen Anforderungen an eine ARP-Sprache aus Szenarien abgeleitet werden, die u.a. auch delegierte Policy-Administration notwendig machen. Auf Basis eines Kriterienkatalogs sollen bekannte Policysprachen auf ihre Eignung für ARPs hin untersucht werden. Die am Besten geeignete Policysprache soll dann an die speziellen Bedürfnisse im FIM-Umfeld angepasst und prototypisch für Shibboleth in Java implementiert werden; hierfür ist, soweit vorhanden, auf existierende Referenzimplementierungen der Policy Decision Points der Policysprache zurückzugreifen. Für die diskutierten Szenarien sind abschließend entsprechende ARPs zu erstellen und die Funktion des Gesamtsystems zu demonstrieren.

Ziel der Diplomarbeit ist also die Konzeption und Implementierung einer policy-basierten Privacy Management Architektur für das föderierte Identitätsmanagementsystem Shibboleth, welche die bisherigen ARP-Funktionalitäten erweitert. Die Mauer in Abbildung 1.2 deutet symbolisch die neu zu entwickelnde Architektur und deren Platzierung in einem föderierten Identitätsmanagementsystem an. Die neue Architektur soll also den Datenaustausch zwischen Identity Provider und Service Provider im Kontext der Autorisierung regeln.

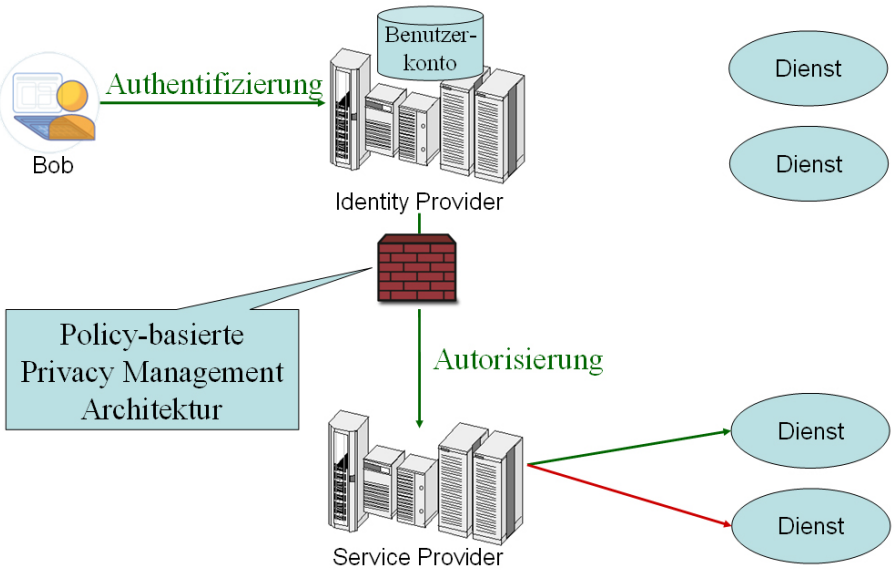

Abbildung 1.2: Ziel der Arbeit

1.3 Struktur der Ausarbeitung

Die Struktur der Ausarbeitung hält sich stark an die Aufgabenstellung. Die folgende Auflistung gibt einen Überblick über den Aufbau der Arbeit:

1. Einführung in das Thema, Beschreibung der Aufgabe (dieses Kapitel)
2. Einführung des föderierten Idemtitätsmanagementsystems Shibboleth (siehe Kapitel 2)
3. Beschreibung von Szenarien zur Ableitung von Anforderungen an eine neue policy-basierte Privacy Management Architektur (siehe Kapitel 3)
4. Herleitung der Anforderungen und Erstellung eines Kriterienkatalogs zur Bewertung der in Frage kommenden Policy Sprachen (siehe Kapitel 4)
5. Untersuchung verschiedener Policy Sprachen mit Hilfe des Kriterienkatalogs (siehe Kapitel 5)
6. Erstellung der neuen Architektur unter Verwendung der ausgewählten Policy Sprache (siehe Kapitel 6)
    a) Beschreibung, wie die ausgewählte Policy Sprache für die neue Architektur verwendet werden kann (siehe Abschnitt 6.1)
    b) Überblick über die in Shibboleth bereits vorhandene Architektur (siehe Abschnitt 6.2)
    c) Beschreibung der Komponenten und des Workflows der neuen Architektur (siehe Abschnitt 6.3)
7. Implementierung der Architektur (siehe Kapitel 7)
8. Konfiguration der Architektur mit Angabe von Verwendungsbeispielen (siehe Kapitel 8)
9. Zusammenfassung, die beschreibt, was in dieser Arbeit erreicht wurde und welche weiterführenden Punkte sich daraus ergeben (siehe Kapitel 9)
10. Anhang, welcher die im Internet veröffentlichte Patch-Seite der neu entwickelten Architektur beinhaltet (siehe dazu 10)

2 Föderiertes Identitätsmanagement am Beispiel Shibboleth

Das Web Single Sign-On System Shibboleth ist eine Software zur verteilten Authentifizierung und Autorisierung, welche einen organisationsübergreifenden Zugriff auf geschützte Webinhalte bietet. Shibboleth basiert auf einer Erweiterung der XML-basierten Auszeichnungssprache Security Assertion Markup Language (SAML), welche die Beschreibung und Übertragung sicherheitsbezogener Informationen ermöglicht. XML steht für Extensible Markup Language. Das Konzept von Shibboleth sieht vor, dass der Benutzer sich nur einmal bei seinem Identity Provider (Heimateinrichtung) authentifizieren muss, um danach auf verschiedene Dienste oder lizenzierte Inhalte eines oder mehrerer Service Provider (Anbieter) zugreifen zu können (Single Sign-On).

Um den Zugriff auf Ressourcen für bestimmte Benutzergruppen einschränken bzw. kontrollieren zu können, bedient sich Shibboleth neben der Authentifizierung der Autorisierung. Dazu werden Benutzerinformationen (Attribute) zwischen den verschiedenen beteiligten Organisationen ausgetauscht, anhand derer dann eine Berechtigungsentscheidung getroffen werden kann. Die ARPs regeln dabei die Rechteverwaltung, also welche Attribute unter welchen Bedingungen an wen herausgegeben werden dürfen.

Shibboleth basiert also auf einem föderativen Ansatz, wobei der Identity Provider seine Benutzer verwaltet und authentifiziert und der Service Provider den Zugang zu den Ressourcen kontrolliert. [Internet2 Shib][WIKI Shib]

Shibboleth wird zur Zeit schon in Projekten mehrerer Länder, wie USA, Schweiz, Australien und Großbritannien, vor allem im universitären Bereich produktiv eingesetzt [Shib Partners]. In Deutschland beschäftigt sich z.B. das Projekt DFN-AAI mit dem Einstz von Shibboleth [DFN AAI].

2.1 Vor- bzw. Nachteile

Gründe für die relativ schnelle Verbreitung und Akzeptanz von Shibboleth finden sich in folgenden Vorteilen wieder:

- Sicherheitsgewinn, da sich Benutzer nur noch ein Passwort merken müssen und dieses nur einmal übertragen wird
- Zeitersparnis, da nur eine einmalige Authentifizierung notwendig ist, um auf alle Systeme zugreifen zu können
- Ressourcen können differenziert geschützt werden
- keine Benutzerverwaltung auf Seiten des Anbieters erforderlich
- Integration in vorhandene Systeme ist häufig mit geringem Aufwand möglich
- Komponenten sind Open Source und stehen kostenfrei zur Verfügung
- Einbindung neuer Ressourcen in das eigene Angebot ist sehr einfach

- berechtigten Nutzern anderer Einrichtungen kann leicht Zugriff auf eigene geschützte Ressourcen gewährt werden
- Nutzung der Ressourcen ist unabhängig von Standort und Zugriffsweg möglich
- Datenschutzaspekte werden berücksichtigt

Jedoch gilt es auch Nachteile zu berücksichtigen:

- Kann ein Angreifer die Identität eines Benutzers entwenden, so stehen ihm sofort alle Systeme, auf die dieser Benutzer Zugriff hat, zur Verfügung.
- Geringe Mächtigkeit des ARP Systems

[AAR]

2.2 Funktionsweise im Überblick

Die Architektur von Shibboleth besteht aus drei Hauptelementen (siehe dazu Abbildung 2.1):

- der `Identity Provider` verwaltet die Benutzer und führt die Authentifizierung durch
- der `Service Provider` dient der Ressourcenverwaltung und der Zugangsberechtigung (Autorisierung)
- das `WAYF` (Where are you from?) lokalisiert den Identity Provider des Benutzers

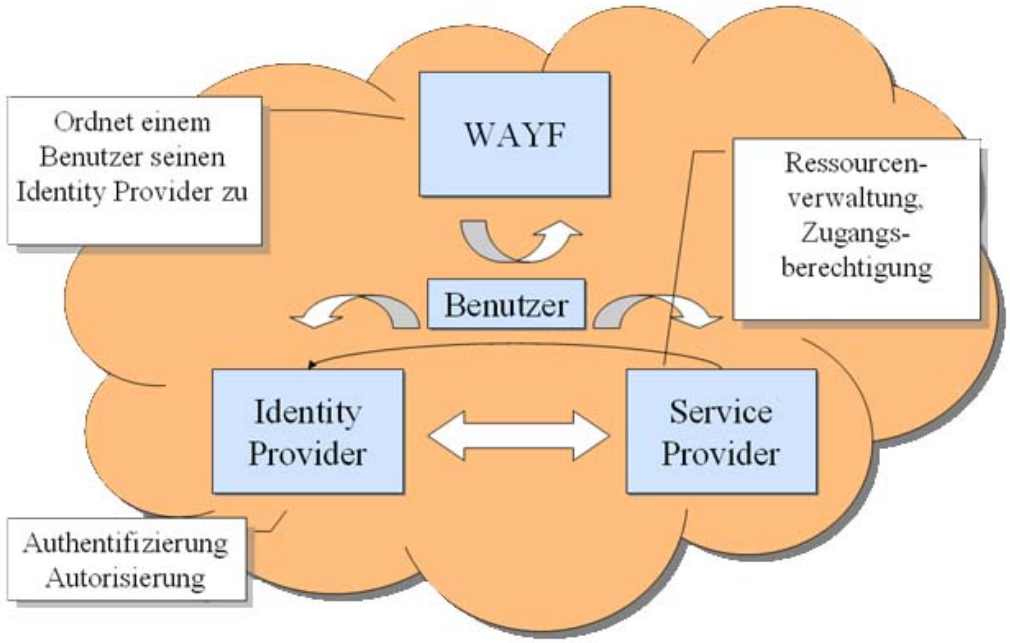

Abbildung 2.1: Übersicht über die Hauptelemente von Shibboleth

Folgendes Beispiel erklärt im Überblick die Funktionsweise von Shibboleth:

Authentifizierung (Wer bist Du?)
Ein Benutzer möchte auf einen geschützten Webinhalt zugreifen. Der Service Provider nimmt die Anfrage entgegen und prüft, ob der Benutzer bereits authentifiziert ist. Wenn nicht, wird er zu einem WAYF (Lokalisierungsdienst) weitergeleitet. Der WAYF bietet eine Auswahl von Identity Providern (Heimeinrichtungen) an. Der Benutzer wählt seinen Identity Provider aus und wird zu diesem weitergeleitet. Der Identity Provider prüft, ob der Benutzer bereits authentifiziert ist. Ist dies nicht der Fall, wird der Benutzer aufgefordert, sich zu authentifizieren. Der Identity Provider stellt einen „digitalen Ausweis“ aus und leitet den Benutzer zum Service Provider zurück, wo der Inhalt des digitalen Ausweises überprüft wird.

Autorisierung (Was darfst Du?)
Benötigt der Service Provider weitere Informationen, um zu entscheiden, ob der Benutzer auf die gewünschten Inhalte zugreifen darf (zum Beispiel die Fakultätszugehörigkeit bei einer Universität), so fragt er bei dem Identity Provider des Benutzers nach. Der Service Provider prüft über das eigene System, ob der Benutzer auf die Ressource zugreifen darf, und gestattet den Zugriff oder lehnt ihn ab. [WIKI Shib]

2.3 Zusammenspiel der Komponenten

Im Folgenden werden die einzelnen Shibbolethbausteine und deren Zusammenspiel erklärt.

2.3.1 Identity Provider

Der Identity Provider, welcher die Benutzer verwaltet und die Authentifizierung durchführt, besteht aus 4 Hauptkomponenten:

- SSO Handler mit dem Handle Service (HS)
- Attribute Query Handler mit der Attribute Authority (AA) und den Attribute Release Policies (ARP)
- Directory Service
- Authentication Mechanism

Aus der Sicht des Identity Providers ist der erste Kontakt die Weiterleitung des Benutzers an das Handle Service, welches im Anschluss den SSO Handler kontaktiert, um festzustellen, ob sich der Benutzer bereits authentifiziert hat. Falls dies noch nicht geschehen ist, muss sich der Benutzer am Authentication Mechanism (welcher z.B. durch das Single Sign-On Authentifizierungssystem „Pubcookie“ realisiert werden kann) mit Hilfe eines Directory Services (z.B. LDAP) authentifizieren, woraufhin dem ACS per Url ein Handle des Benutzers (ein Handle ist eine Benutzerreferenz, welche eine SAML Authentifizierungsbestätigung beinhaltet) geschickt wird. Als nächstes kommt bei der AA im Attribute Query Handler des Identity Providers eine Attributanfrage vom shibd (Shibboleth Daemon) an, welche das vorherige Handle wieder enthält. Nun werden mit Hilfe der ARPs alle erlaubten Attribute des Benutzers des Handles an den shibd zurückgesendet.

2.3.2 Service Provider

Der Service Provider, welcher der Ressourcenverwaltung und der Zugriffsberechtigung dient, besteht aus 3 Hauptkomponenten:

- Assertion Consumer Service (ACS)
- Shibboleth Daemon (shibd) mit dem Attribute Requester (AR)
- Resource Manager (RM) mit den AAPs (Attribute Acceptance Policies)

Aus der Sicht des Service Providers ist der erste Benutzerkontakt die Anfrage auf einen von Shibboleth geschützten Inhalt. Der RM beauftragt den ACS über das WAYF den Identity Provider des Benutzers ausfindig zu machen. Daraufhin antwortet der Identity Provider mit einem Handle, also einer Benutzerreferenz, welches das ACS an den shibd weiterleitet. shibd kontaktiert mit diesem Handle das AA des Identity Providers und erfrägt alle zu diesem Handle möglichen Attribute. An den RM weitergeleitet validiert dieser mittels den AAPs die Benutzerinformationen. Anhand dieser Informationen wird dem Benutzer dann der Zugriff gewährt oder verweigert.

2.3.3 Where Are You From

Das Where Are You From (WAYF) kann entweder im Service Provider oder davon ausgegliedert von der Föderation betrieben werden. Es stellt die Verbindung zwischen Benutzer und Identity Provider her und leitet die Anfrage des Benutzers an das HS des Identity Providers weiter.

2.3.4 Graphische Übersicht

Figur 2.2 stellt den oben beschriebenen Ablauf noch einmal graphisch dar. Die nummerierten Pfeile beziehen sich dabei auf den Informationsfluss unter Zuhilfenahme der einzelnen Shibbolethkomponenten.

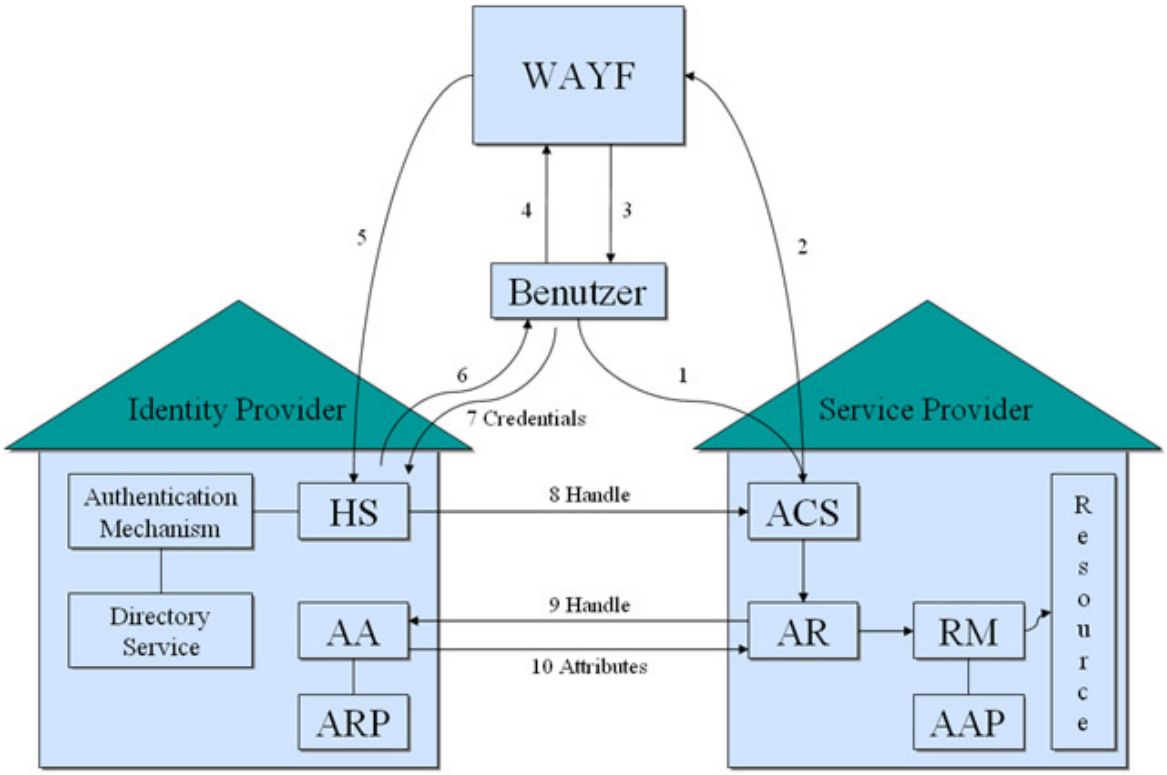

Abbildung 2.2: Zusammenspiel der Komponenten von Shibboleth

[Internet2 Shib][TWIKI]

2.4 Zusammenfassung

Das verteilte Single Sign-On System Shibboleth vereinigt „Authentifizierung“, „Autorisierung“ und „Rechteverwaltung“ und bietet so eine zentrale Verwaltung dieser Aufgaben. Shibboleth übernimmt den kompletten Schutz für Ressourcen bzw. Dienstleistungen über Organisationsgrenzen hinaus. Dadurch löst Shibboleth geschützte Anwendungen von wichtigen Aufgaben ab und reduziert so dessen Komplexität. Betrachtet man die ständig wachsende Zahl von angebotenen Diensten verschiedenster Organisationen, so erweist sich Shibboleth als gute Wahl, um die sonst immer komplexer werdende Administration erheblich zu vereinfachen.

3 Szenarien

Im ersten Schritt zur Erstellung einer neuen policy-basierten Privacy Management Architektur werden Szenarien betrachtet, aus denen Anforderungen an die neue Architektur abgeleitet werden können. Da in dieser Arbeit die ARP-Seite policy-basierter Privacy Management Systeme betrachtet wird, also die Seite, die sich mit der Verwaltung der Benutzerattribute beschäftigt, beziehen sich auch die Szenarien hauptsächlich auf diese Seite. Die folgenden Szenarien zeigen in groben Zügen auch die Entwicklungsstufen des Identity Managements, wobei die ersten sechs Szenarien aufeinander aufbauen, d.h. das jeweils vorherige Szenario erweitern.

3.1 Szenario 1: Ohne Identitätsmanagement

Der Informatikstudent Bob studiert an der LMU München. Die LMU bietet unterschiedliche zugangsgeschützte Dienste an, die eine Authentifizierung benötigen. Solche Dienste sind z.B. das Internetportal „CampusLMU“ oder einfach eine geschützte Ressource, die vorlesungsinterne Informationen enthält. Bob hat für jeden dieser Dienste einen eigenen Account, d.h. Benutzername und Passwort, was zu baldiger Verwirrung und falschen Passworteingaben führt. Neben der Authentifizierung ist keine Autorisierung nötig (siehe Abbildung 3.1). Bob ist es zur Zeit nicht möglich, auf geschützte Ressourcen einer anderen Organisation, wie der TU, zuzugreifen.

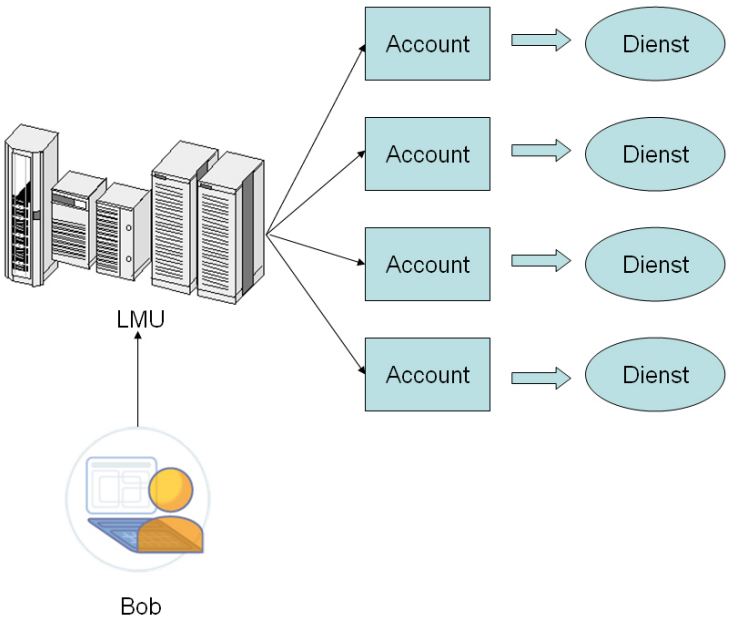

Abbildung 3.1: Szenario 1

3.2 Szenario 2: Identity und Access Management Phase 1

Die LMU hat ein Identity und Access Management System eingeführt. Dazu wurden die mehrfachen dezentralen Benutzerdatenbestände in einer zentralen Datenbank zusammengefasst. In dieser zentralen Datenbank sind nun alle Benutzerinformaitonen, wie Name, Alter, Universität, Fakultät und E-Mail Adresse, abgelegt. Probleme wie Redundanz oder Inkonsistenz der Daten wurden durch die Zentralisierung behoben. Zur Nutzung der Dienste der LMU muss sich Bob nun nur noch einmalig und zentral an der LMU authentifizieren. Die LMU fungiert also als Identity Provider, da sie zentral die Datenbestände verwaltet und die Authentifizierung regelt. Bob kann nun theoretisch nach dieser einmaligen Anmeldung alle geschützten Dienste der LMU nutzen. Da jedoch nicht jeder Student auf alle Dienste zugreifen darf, ein BWL Student darf z.B. keinen Einblick in geschützte Ressourcen der Informatik bekommen, wurde neben der Authentifizierung ein festes Autorisierungsregelwerk konzipiert. In diesem Regelwerk wird festgelegt, welche Benutzer auf welche Dienste, unter Herausgabe bestimmter Benutzerinformationen, zugreifen dürfen. Es wurde also bestimmt, welche Benutzerattribute von der zentralen Datenbank an Dienste weitergeleitet werden müssen, damit diese eine Zugriffsentscheidung treffen können. So wurde z.B. für das „ifkw.extra“ festgelegt, dass an diesen Dienst die Benutzerinformationen „Universität“ und „Fakultät“ weitergeleitet werden müssen, und Bob darauf nur Zugriff bekommt, wenn er von der LMU mit Studienfach KW kommt (siehe Abbildung 3.2).

Der Datenschutz erweist sich bei diesem Szenaio als Problem. Es wird von der LMU einfach festgelegt, welche Benutzerdaten sie an Dienste herausgibt, ohne dass die Benutzer darüber entscheiden oder eigene Einstellungen treffen können. Daher ist es notwendig ein System einzuführen, in dem Benutzer selbst bestimmen können z.B. welche Attribute sie an wen herausgeben möchten. Diese Einstellungen können durch Attribute Release Policies getroffen werden.

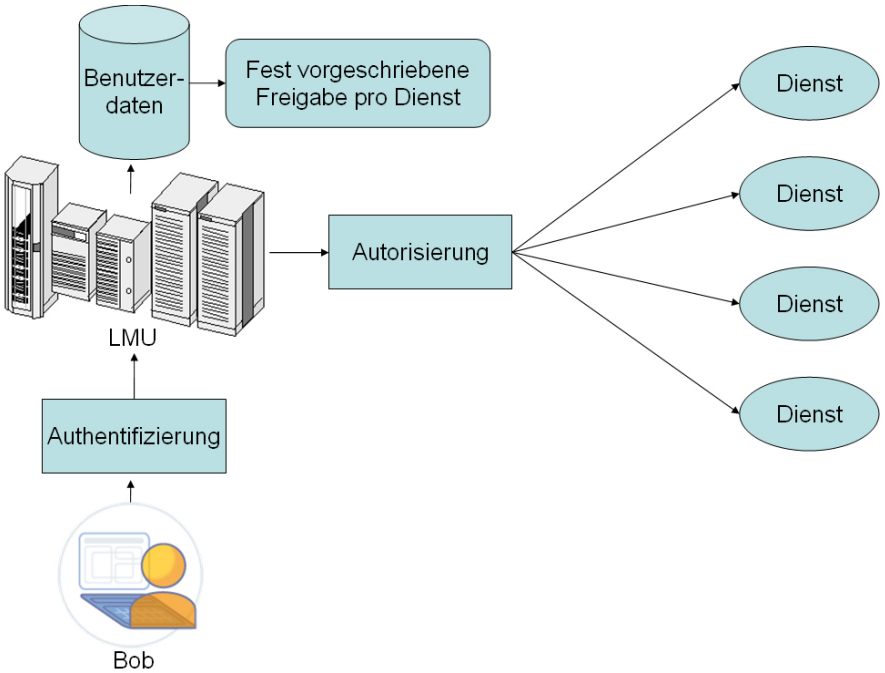

Abbildung 3.2: Szenario 2

3.3 Szenario 3: Identity und Access Management Phase 2

Die LMU hat ihr Identity und Access Management System um ARPs erweitert. Bob bzw. ein Administrator kann regeln, welche Attribute freigegeben werden dürfen. In diesem Fall vertraut Bob allen möglichen Diensten und will, dass seine gesamten Attributinformationen ohne Beschränkungen freigegeben werden (siehe Abbildung 3.3).

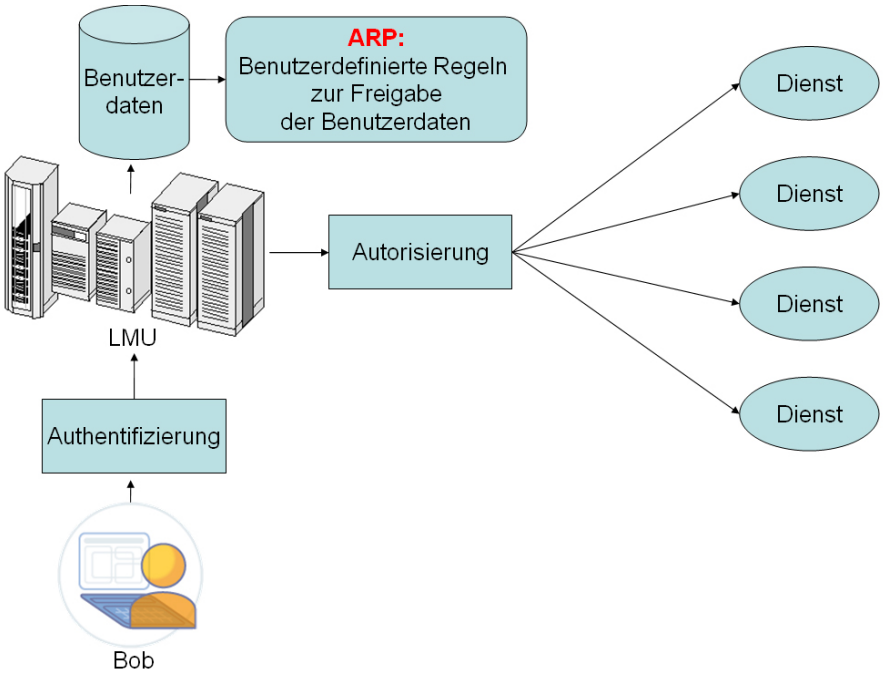

Abbildung 3.3: Szenario 3

3.4 Szenario 4: Einschränkung der Attributwerte

Bob will die Informationen einschränken, die an Dienste weitergeleitet werden dürfen. Er beschließt seine E-Mail-Adresse nur dann freizugeben, wenn diese der Domain „informatik.uni-muenchen.de“ angehört. Bob will somit verhindern, dass seine private E-Mail Adresse an Fremde weitergegeben werden könnte. Durch die Möglichkeit Informationen zu unterdrücken besteht die Gefahr, dass Bob zwar eigentlich für einen Dienst autorisiert wäre, ihm aber durch seine in den ARPs getroffenen Einstellungen der Zugriff verweigert wird.

3.5 Szenario 5: FIM

Die LMU hat ein föderiertes Identitätsmanagement System errichtet. Dadurch kann Bob nicht nur die einrichtungsinternen Dienste der LMU nutzen, sondern auch die Dienste der TU oder des LRZ (Leibniz-Rechenzentrum), ohne dort zusätzliche Benutzerdaten ablegen zu müssen. Doch bei der TU will Bob bei weitem nicht so freizügig mit seinen Informationen umgehen wie bei der LMU. Bob bestimmt, dass die TU manche Attribute überhaupt nicht bekommt, wie z.B. seine E-Mail Adresse, und

andere nur in eingeschränkter Weise. Er bestimmt, dass seine Informationen nur zum Zweck der Autorisierung freigegeben werden sollen. Bob legt auch für alle anderen teilnehmenden Organisationen und deren Dienste unterschiedliche Freigabeeinstellungen fest (siehe Abbildung 3.4).

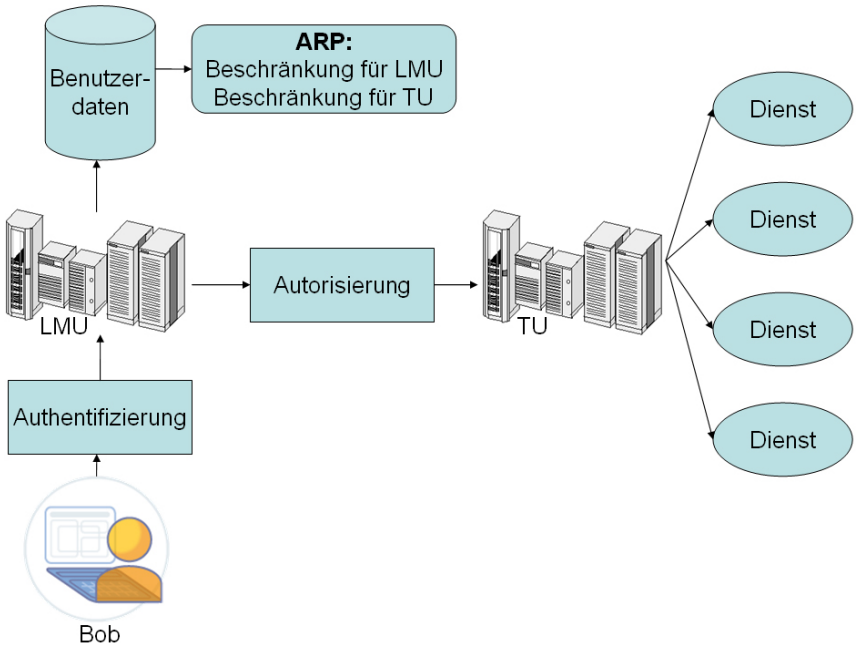

Abbildung 3.4: Szenario 5

Für organisationsübergreifende Szenarien wird zwischen den Identity Providern und den Service Providern ein Vertrauensverhältnis und ein organisatorischer Rahmen für den Austausch von Benutzerinformationen erschaffen. Dies wird in statischen Verträgen festgehalten. Es werden geeignete Übereinkünfte getroffen, dass und welche Daten ausgetauscht werden. Wichtige Fragestellungen, wie z.B. die einheitliche Regelung der Syntax und der Semantik der einzelnen Attribute, werden geklärt. Ist z.B. die Studiengangsbezeichnung eines Informatikers „Informatik“ oder „Computer Science“?

3.6 Szenario 6: Weitere Beschränkungsmöglichkeiten

Bob möchte nicht für jedes Attribut einzeln eine Regel aufstellen müssen. Deshalb können mehrere Attribute zu Attributkategorien zusammengefasst werden, auf welche dann Regeln gebildet werden können. Bobs Daten sind z.B. in unterschiedliche Kategorien wie `Personendaten` (Vorname, Nachname, Geburtsdatum, Adresse, etc.), `Online-Daten` (E-Mail, etc.), `Immatrikulationsdaten` (Martikelnummer, Fachsemester, etc.) und `Belegungsdaten` (Kurse, Noten, etc.) unterteilt.

Bob interessiert sich für einen von der VHB (Virtuelle Hochschule Bayern) angebotenen Kurs, welche auch dem FIM der LMU beigetreten ist. Bob gibt dafür die Immatrikulationsdaten und die Personendaten ohne Einschränkungen mit Hilfe einer Regel frei. Zur Anmeldung möchte die VHB jedoch aus statistischen Gründen auch die Belegungsdaten erfragen. Bob vertraut zwar der VHB, möchte sich jedoch nicht die Blöße geben und nur solche Kurse samt Noten herausgeben, in denen er besser als 2.0 ist. Seine E-Mail Adresse gibt Bob nur heraus, wenn dies auch schon mehr als 10 Ande-

re seiner Kollegen getan haben. Zusätzlich stellt er die Bedingung, dass die Belegungsdaten nach Kursende wieder gelöscht werden müssen.

3.7 Szenario 7: Kombination von Policies

Nachdem Bob das Studium erfolgreich beendet hat, hat er einen attraktiven Arbeitsplatz in einem IT-Unternehmen angenommen. Das Unternehmen fungiert als Bobs Identity Provider. Auch dort hat er eine Vielzahl von persönlichen Daten abgelegt, jedoch möchte er diese, außer an die Firma selbst, an niemanden herausgeben. Diese Forderung legt Bob in seiner eigenen Benutzer-ARP ab und überschreibt somit die Default-ARP des Identity Providers, welcher standardgemäß alle Benutzerattribute weitergeben würde.

Bobs Firma beteiligt sich an einer Ausschreibung für ein Software Projekt eines Konzerns. Im Rahmen der Ausschreibung müssen auch Profile der beteiligten Entwickler eingereicht werden. Diese Profile enthalten allgemeine Informationen zu den Personen (Name, aber auch Foto), in welcher Position sie bei der Firma tätig sind, welche Zertifizierungen sie haben und in welchen Projekten sie schon beteiligt waren. Dazu überschreibt Bobs Vorgesetzter mit einer Superior-ARP wiederum die Benutzer-ARP.

Die Privacy Policies von Bob greifen also in diesem speziellen Fall nicht. Da die Benutzerakzeptanz durch das Übergehen von Benutzereinstellungen schnell an ihre Grenzen stoßen kann, sollten solche Eingriffe jedoch die Ausnahme bleiben.

(siehe Abbildung 3.5).

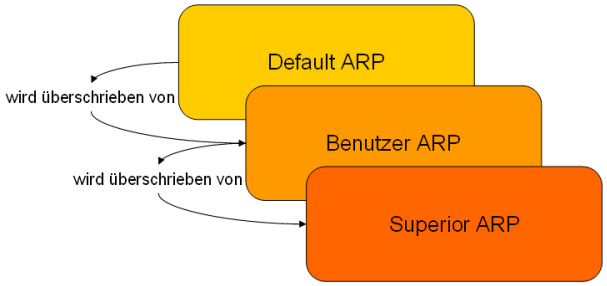

Abbildung 3.5: Szenario 7

3.8 Szenario 8: Rollenmodell

Bob möchte von seinem Arbeitsplatz aus ein Online Forum nutzen, welches den Service Privider darstellt. Meldet sich Bob am Forum während der Arbeitszeit an, so agiert er in der Rolle eines Firmenmitglieds, außerhalb der Arbeitszeit in der Rolle einer Privatperson. Für jede der Rollen hat Bob ein eigenes Benutzerprofil, also ein Arbeitsprofil und ein Freizeitprofil. Diese Profile unterscheiden sich im Umfang der enthaltenen Informationen und in der Belegung der Attribute. So steht im Arbeitsprofil die E-Mail Adresse der Arbeit, im Freizeitprofil die private E-Mail Adresse. In der Rolle einer Privatperson erhält das Forum für statistische Zwecke lesenden Zugriff auf Informationen wie Alter und Geschlecht aus dem Freizeitprofil. Nutzt Bob jedoch die Rolle eines Firmenmitglieds,

dann erhält das Forum lesenden Zugriff auf Informationen wie Abteilung, Projekte und Position in der Firma aus dem Arbeitsprofil. Des Weiteren wird Bobs E-Mail Adresse zwischen 12 Uhr und 13 Uhr nicht freigegeben (siehe Abbildung 3.6).

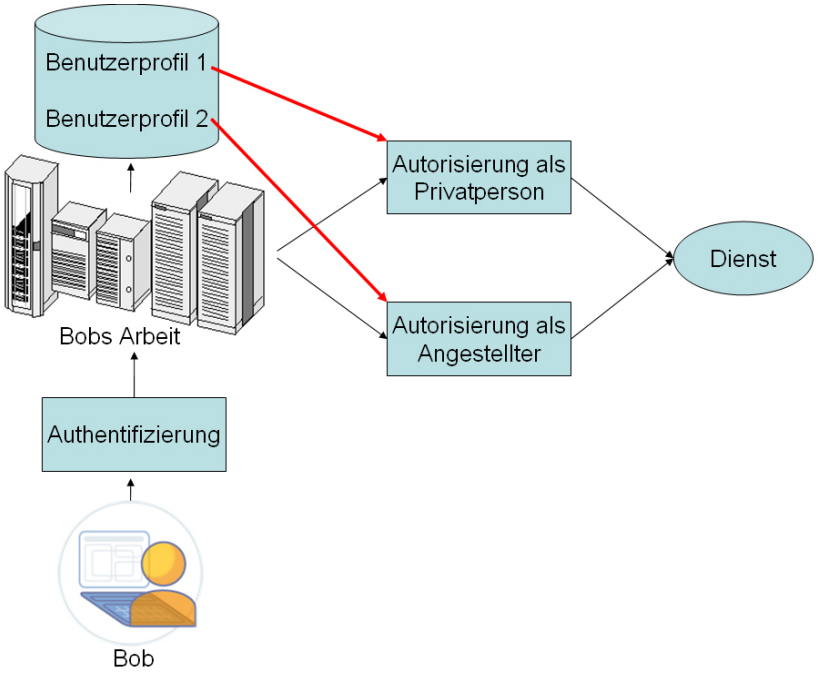

Abbildung 3.6: Szenario 8

3.9 Zusammenfassung

Durch die Szenarien wird deutlich, dass die Freigabe von Benutzerinformationen auf vielfältige Weise eingeschränkt werden kann. Je genauer die Informationsfreigabe geregelt werden kann, desto besser kann auch der gewünschte Datenschutz des Benutzers erreicht und eingehalten werden. Für eine Vielzahl von Dienstleistungen kann so ein differenziertes Regelwerk konstruiert werden. Um entsprechende Freigaberegeln beschreiben zu können, benötigt man eine ARP-Sprache, die den Anforderungen, welche aus den Szenarien hervorgehen, genügt. Welche Anforderungen dies im Speziellen sind, beschreibt das nächste Kapitel.

4 Anforderungsanalyse

Aus den beschriebenen Szenarien lassen sich eine Reihe von Anforderungen an eine Policy Sprache für ein policy-basiertes Privacy Management System herleiten. Die Anforderungen werden zunächst aus den Szenarien extrahiert, dann stichpunktartig zusammengefasst und schließlich in einem Kriterienkatalog mit Gewichtung festgehalten. Der so gewonnene Kriterienkatalog dient im weiteren Verlauf als Bewertungsgrundlage zur Auswahl geeigneter Policy-Sprachen für die neu zu entwickelnde Architektur.

4.1 Herleitung der Anforderungen

Aus Szenario 1 folgen keine Anforderungen an eine policy-basierte Privacy Management Architektur, da es kein Identitätsmanagement unterstützt. Jeder Dienst erfordert ein eigenes Benutzerkonto, auf welches der entsprechende Dienst in vollem Umfang zugreifen kann.

In den Szenarien 2 und 3 wird ein Identity und Access Management System eingeführt. Um Dienste nutzen oder auf geschützte Ressourcen zugreifen zu können reicht es nicht mehr aus sich „nur“ zu authentifizieren. Nach erfolgreicher Authentifizierung werden die Zugriffsrechte anhand von Benutzerattributen, welche beim Identity Provider abgelegt sind, überprüft. Auch aus Szenario 2 folgen noch keine Anforderungen an eine policy-basierte Privacy Management Architektur, da hier noch keine ARPs im Spiel sind, sondern Attributfreigaben fest vorgeschrieben sind. Jedoch bestehen organisatorische Anforderungen, dass z.B. im Vorfeld ausgehandelte Datenschutzbestimmungen grundsätzlich eingehalten werden. In Szenario 3 kann durch ARPs geregelt werden, auf welche Attribute zugegriffen werden darf. Somit ist eine Grundanforderung an eine ARP, dass Attribute (siehe 6.1.4 Attribut) gesperrt, oder freigegeben werden können.

In Szenario 4 werden die Attribute nicht nur allgemein beschränkt, sondern die Attributfreigabe wird abhängig vom Attributwert gemacht. Eine zweite Anforderung ist es also, Bedingungen (siehe 6.1.4 (Komplexe) Bedingung) an Attributwerte stellen zu können. Diese Bedingungen müssen zum Beispiel in Form von direkten Angaben oder regulären Ausdrücken realisiert werden können. Für Szenario 4 müssen auch sogenannte „multi-valued“ Attribute unterstützt werden, d.h. ein Attribut kann mit mehreren Attributwerten belegt werden (hier hat Bob z.B. mehrere Werte für das Attribut `e-mail-address`).

Szenario 5 verlässt die Grenzen nur einer Organisation und beschreibt ein föderiertes Identitätsmanagement. Attribute müssen organisationsübergreifend austauschbar sein. In den ARPs muss die Attributfreigabe vom Service Provider (siehe 6.1.4 Nutzer) bzw. vom Dienst und vom Verwendungszweck (siehe 6.1.4 Zweck) (Zweck im Szenario: Autorisierung) abhängig gemacht werden können. Weitere Zwecke könnten sein: Statistik, Marktforschung, etc.

In Szenario 6 werden die Benutzerdaten kategorisiert. Mit Hilfe der ARPs muss eine Zuordnung von Zugriffsrechten für ganze Attributkategorien (siehe 6.1.4 Attributkategorie) und Nutzerkategorien (siehe 6.1.4 Nutzerkategorie) (Service Provider bzw. Dienste) möglich sein. Es werden attributübergreifende Bedingungen gestellt. Die Rechtezuweisung eines Attributs muss also auch anhand anderer

Attributwerte möglich sein. Szenario 6 geht sogar noch einen Schritt weiter und macht die Freigabe vom Verhalten oder den Attributen anderer Benutzer abhängig. Daraus ergibt sich die Anforderung, unter gewissen Umständen auf Attribute anderer Benutzer zugreifen zu können. Mit Hilfe der ARPs müssen Obligationen (siehe 6.1.4 Obligation) gestellt werden können. Dies sind Bedingungen, die nach der Attributfreigabe eingehalten werden müssen. So müssen die gesammelten Informationen z.B. nach einer gewissen Zeit beim Service Provider wieder gelöscht werden.

Szenario 7 beschreibt eine delegierte Policy-Administration (Policies können von mehreren Instanzen erstellt werden) und die Kombination dieser Policies (siehe 6.1.5 Kombination von Policies). Es gibt verschiedene Ebenen von ARPs, wobei höhere Ebenen die unteren Ebenen übernimmt und gegebenenfalls überschreibt. Default ARPs legen Standardfreigaben fest, die den Benutzern eine solide Voreinstellung liefern. Wünscht ein Benutzer dennoch andere Regeln, so kann er diese in seiner Benutzer ARP festlegen. Diese ARP wirkt in ergänzender Weise. Sollte es wie in Szenatio 7 zu Fällen kommen, in denen Attribute gegen den Willen eines Benutzers behandelt werden müssen, so muss das ARP Management noch höhere Ebenen anbieten, die dann wiederum die Benutzer ARPs überschreiben. Es werden Mechanismen benötigt, welche ein geeignetes Zusammenfügen verschiedener ARPs zu einer Einzelnen ermöglichen, z.B. durch Verwendung von Prioritäten oder durch Algorithmen wie z.B. „deny overrides“, welcher die Freigabe eines Attributs verweigert, sobald eine von mehreren passenden Policies die Freigabe verweigert. Szenario 7 verdeutlicht des Weiteren, dass festgelegt werden können muss, für welchen Benutzer eine ARP erstellt wurde und gilt. Gilt die ARP also für alle Benutzer (z.B. Standard-ARP von einem Administrator), nur für Bob oder für eine bestimmte Benutzergruppe? Wie dieser Sachverhalt (welcher nicht in den Anforderungskatalog für Policy-Sprachen aufgenommen wird, da er nicht von einer speziellen Policy-Sprache abhängig ist) in der später entwickelten Architektur umgesetzt wird, zeigt Abschnitt 6.3.2.

Szenario 8 setzt ein Rollenmodell voraus. Je nach Rolle (siehe 6.1.7 Rollenmodell) eines Benutzers werden verschiedene ARPs auf ggf. verschiedene Benutzerprofile angewandt. Des Weiteren muss die Attributfreigabe von Nicht-User-Attributen (siehe 6.1.6 Zugriff auf Nicht-Benutzer-Attribute), wie z.B. dem Datum, abhängig gemacht werden können. Mit Hilfe der ARPs müssen Handlungen (siehe 6.1.4 Handlung) unterstützt werden, d.h. welche Aktionen dürfen auf die erhobenen Daten ausgeführt werden? Beispiele hierfür sind lesender Zugriff oder schreibender Zugriff auf die Benutzerdaten.

4.2 Anforderungen im Überblick

Die aus den Szenarien gewonnen Anforderungen in einer Liste zusammengefasst:

- (1) **Attribute müssen freigegeben bzw. gesperrt werden können**: Die Attributfreigabe muss z.B. mit „Permit“(Freigabe erlauben), „Deny“ (Freigabe verweigern), „Don’t care“ (wenn einem Attribut weder eindeutig ein „Permit“ noch ein „Deny“ zugeordnet werden kann), „Indeterminate“ (wenn es zu einem Attribut keinen eindeutigen Wert gibt) oder „Error“ (wenn z.B. ein Attribut nicht gefunden wird) geregelt werden können.
- (2) **Abhängigkeit der Attributfreigabe vom Attributwert / Bedingung**: Es müssen Bedingungen zur Attributfreigabe gestellt werden können, z.B. die Freigabebedingung ist, dass das Attribut einen bestimmten Attributwert hat. In einer Bedingung müssen Attributwerte mit festen Zeichenfolgen oder mit regulären Ausdrücken verglichen werden können. Daneben müssen verschiedenste Funktionen zum Vergleich unterstützt werden, wie z.B. logische Funktionen oder arithmetische Funktionen.
- (3) **Abhängigkeit vom Nutzer der Daten**: Es muss ein „Target“ angegeben werden können, welches bestimmt, wer auf die Attribute zugreifen darf. Ein „Target“ muss die Angabe eines

Service Providers oder eines Dienstes unterstützen.

- (4) **Abhängigkeit von der Handlung**: Es muss möglich sein, die Freigabe von einer Handlung abhängig zu machen. Handlungen bestimmen, wie ein Service Provider auf die Daten zugreifen darf, z.B. „read“ oder „write“.
- (5) **Abhängigkeit vom Zweck**: Es muss ein Verwendungszweck angebbar sein. Wozu dürfen Daten verwendet werden (für Statistiken, zur Autorisierung , etc.)? Z.B. in der Policy Sprache P3P standardisierte Verwendungszwecke können unter [P3P Specification], Abschnitt `3.3.4 The PURPOSE element` nachgelesen werden.
- (6) **attributübergreifende Abhängigkeiten**: In der Freigabebedingung muss auf andere Attribute verwiesen werden können, anhand derer Werte eine Entscheidung getroffen werden kann.
- (7) **Abhängigkeit zu anderen Benutzern**: es müssen geeignete Mechanismen vorhanden sein, welche es erlauben, auf benutzerfremde Attribute zugreifen zu können um damit benutzerübergreifende Bedingungen stellen zu können.
- (8) **Abhängigkeit von externen Informationen**: es muss z.B. auf das Datum oder die Uhrzeit verwiesen werden können.
- (9) **Obligationen**: Es müssen Bedingungen an die Datennutzung in Form von Obligationen möglich sein. Z.B. lösche die Daten nach einem Monat. Angebbar muss also z.B. ein Zeitraum sein und die Bedingung selbst (löschen).
- (10) **Kategorien von Attributen**: Attribute müssen kategorisierbar sein, d.h. zusammengefasst werden können, um so die Regelbildung zu erleichtern. Dies kann in mehreren Formen geschehen. Z.B. durch die Verwendung von Datenhierarchien. Durch Hierarchien ist es möglich, eine Regel für einen Hauptknoten (z.B. Adresse) zu bestimmen, die dann automatisch auch auf alle Kindknoten (z.B. Ort, Straße) angewandt wird. Eine andere Möglichkeit die Regelbildung zu vereinfachen ist die Verwendung von Gruppen, anstatt von Hierarchien. Attribute können so in „flachen“ Gruppen zusammengefasst werden, auf deren Basis dann Regeln definiert werden können.
- (11) **Kategorien von Nutzern/Service Providern**: siehe Beschreibung zu Punkt (10). Ziel ist es, mehrere Nutzer durch nur eine Regel ansprechen zu können.
- (12) **Kombination von Policies**: es müssen mehrere Policies für eine Autorisierungsentscheidung zusammengefasst werden können. Ein oder mehrere Algorithmen zur Kombination der Policies müssen spezifiziert sein, wie z.B. „Deny-overrides“ oder „Permit-overrides“, welche in [XACML Specification] Abschnitt 2.3. definiert werden.
- (13) **Rollenmodel**: Es muss ein RBAC (Role Based Access Control) definiert sein, d.h. Zugriffsentscheidungen können von bestimmten Rollen, die ein Benutzer hat, abhängig gemacht werden. Die entsprechende Rolle, für die eine Policy erstellt wurde, muss z.B. durch ein Attribut im Kopfbereich einer Policy angegeben werden können.

4.3 Benötigte Sprachelemente

Zieht man die aus den Szenarien hergeleiteten Anforderungen in Betracht, so ergeben sich eine Reihe für Policysprachen benötigte Sprachelemente. Es muss angegeben werden können:

- Attribut (z.B.: Name, Alter evtl. in Abhängigkeit der Benutzerrolle)
- Attributkategorie (z.B.: Adresse, besteht aus Ort, Straße, Hausnummer, Name)
- (komplexe) Bedingung (z.B.: Wert enthält @ifi.lmu.de oder Freigabe von Attribut Schein nur, wenn Attribut Note besser als 2 ist)
- Zweck (z.B.: Buchführung, Statistik)
- Handlung (z.B.: Lesezugriff, Schreibzugriff)
- Obligation (z.B.: nach 10 Tagen löschen)
- Nutzer (z.B.: Dienst, Service Provider)
- Nutzerkategorie (z.B.: Dienstart)

4.4 Weitere Entscheidungskriterien

Neben den funktionalen Eigenschaften einer Policy Sprache ist auch zu berücksichtigen, ob es schon eine frei verfügbare Implementierung eines entsprechenden PDP gibt, der in die zu erstellende Architektur integriert werden kann. Auch die Mächtigkeit einer solchen Implementierung muss gewertet werden. Die Frage nach einer existierenden Implementierung fällt deswegen ins Gewicht, da der Rahmen dieser Arbeit nicht gesprengt werden darf. Allein die Implementierung eines PDP mit umfassenden Funktionen würde die Zeit einer weiteren Arbeit in Anspruch nehmen.

4.5 Kriterienkatalog

Jeder Anforderung wird eine bestimmte Punktezahl bzgl. ihrer Wichtigkeit zugeordnet. So erhält jedes Sprachelement 7 Punkte. Alle 8 Sprachelemente zusammen bekommen also 56 Punkte. Wird ein Sprachelement in einer Policy Sprache nicht unterstützt, so wird von den maximalen 56 Punkten die entsprechende Punktzahl abgezogen. Da für den Rahmen dieser Arbeit ein schon vorhandener PDP wichtig ist, erhält diese Forderung 24 Punkte. Ebenfalls sehr wichtig ist die Unterstützung der Kombination von Policies, welche 10 Punkte erhält. Der Zugriff auf Nicht-Benutzer-Informationen und die Unterstützung eines Rollenmodells erhalten jeweils 5 Punkte. Wird eine Anforderung nicht in vollem Umfang unterstützt, so können von der jeweiligen maximalen Punktzahl Abzüge gemacht werden. Insgesamt können höchstens 100 Punkte erreicht werden.

Daraus ergibt sich folgender Kriterienkatalog:

ANFORDERUNG	Bewertung
Unterstützung der Sprachelemente	max 56
Implementierung eines PDP	max 24
Kombination von Policies	max 10
Zugriff auf Nicht-Benutzer-Informationen	max 5
Rollenmodell	max 5

Tabelle 4.1: Kriterienkatalog

4.6 Zusammenfassung

In diesem Kapitel wurden eine Vielzahl von Anforderungen beschrieben, die eine Policy Sprache erfüllen sollte, um den Szenarien gerecht zu werden. Dabei wird nicht nur auf Eigenschaften der Policy Sprache selbst geachtet, sondern auch auf die Tatsache, ob es dafür bereits eine PDP-Referenzimplementierung gibt, da dies den praktischen Teil der Arbeit in einen machbaren Rahmen lenkt. Mit den abgeleiteten Kriterienkatalog können nun verschiedene Policy Sprachen auf ihre Eignung für ein policy-basiertes Privacy Management System untersucht, beurteilt und ausgewählt werden.

5 Untersuchung existierender Ansätze/Policysprachen

In den folgenden Abschnitten wird der „State of the Art“ anhand der im Web Single Sign-On Systems Shibboleth integrierten ARP-Sprache untersucht. Neben der ARP-Sprache von Shibboleth werden die Policysprachen PONDER, EPAL, P3P, E-P3P und XACML auf ihre Eignung für eine policy-basiertes Privacy Management Architektur untersucht. Diese Policysprachen wurden für die Untersuchung ausgewählt, da sie bekannt, relativ weit verbreitet sind und sich in den verschiedensten Systemen produktiv im Einsatz befinden. Zur Untersuchung herangezogen wird der entwickelte Kriterienkatalog. Je mehr eine Policysprache die Punkte des Kriterienkatalogs erfüllt, desto passender ist sie im Bezug auf die Integration in das föderierte Identitätsmanagementsystem Shibboleth.

5.1 ARP-Sprache von Shibboleth

5.1.1 Einführung

Die auf XML basierenden Shibboleth-ARPs dienen als Filter für Benutzerinformationen, welcher entscheidet, welche Attribute an wen herausgegeben werden dürfen, ohne dabei selbst Informationen zu erzeugen. ARPs können nur solche Attribute freigeben, welche in einem Verzeichnisdienst abgelegt sind, der durch einen Attribute Resolver für Shibboleth zugänglich gemacht wird. ARPs werden für gewöhnlich von einem Admisistrator verwaltet, jedoch unterstützt Shibboleth auch die Möglichkeit für den Benutzer selbst, seine Attributfreigaben zu kontrollieren. Schränkt ein Benutzer seine Attributfreigaben zu stark ein, so kann es sein, dass er einen bestimmten Dienst dadurch nicht mehr nutzen kann, was jedoch vom Benutzer durchaus so gewollt sein kann.

5.1.2 Aufbau der ARPs

Jede ARP Datei enthält eine ARP. Eine ARP ist ein Container für eine Menge von ARP Regeln. Jede ARP Regel spezifiziert eine Richtlinie zur Freigabe von Attributen an einen bestimmten Dienst. Mit Hilfe der ARPs können auch Attribut/Wert Paare eingeschränkt werden.

5.1.3 Beispiele

ARPs sind XML basiert. Das Wurzelelement einer ARP ist `AttributeReleasePolicy`, gefolgt von einem oder mehreren `Rule` Elementen. Jedes `Rule` Element besteht aus einem `Target` Element und mindestens einem `Attribute` Element. Das `Target` Element bestimmt für wen die Regel gilt, die `Attribute` Elemente spezifizieren die möglicherweise freizugebenden Attributnamen und deren Werte.

Folgendes einfaches Beispiel gibt das Attribut eduPersonAffiliation (Zeilen 12 - 14) an jeden beliebigen Service Provider (Zeile 10) ohne Berücksichtigung des Attributwertes (Zeile 13) heraus:

```
<?xml version="1.0" encoding="UTF-8"?>
<AttributeReleasePolicy xmlns:xsi="http://www.w3.org/XMLSchema-instance"
xmlns="urn:mace:shibboleth:arp:1.0"
xsi:schemaLocation="urn:mace:shibboleth:arp:1.0 shibboleth-arp-1.0.xsd">

   <Description>Simplest possible ARP.</Description>

   <Rule>
      <Target>
         <AnyTarget/>
      </Target>
      <Attribute name="urn:mace:dir:attribute-def:eduPersonAffiliation">
         <AnyValue release="permit"/>
      </Attribute>
   </Rule>

</AttributeReleasePolicy>
```

Im `Target` Element kann neben `AnyTarget` auch die Service Provider Id mittels einer `matchFunction` spezifiziert werden. Folgendes Beispiel verwendet einen regulären Ausdruck und gibt TRUE zurück, falls der Service Provider aus der Domain example.edu stammt:

```
<Requester matchFunction="shibboleth:arp:matchFunction:regexMatch">
   .*\.example\.edu
</Requester>
```

Das `Attribute` Element muss entweder das `AnyValue` Element oder ein oder mehrere `Value` Elemente enthalten. Diese Elemente spezifizieren die Freigabe oder die Rückhaltung des Attributs. Folgendes Beispiel verbietet die Freigabe des Attributwertes `member@example.edu`. Andere Attributwerte dürfen herausgegeben werden.

```
<Attribute name="urn:mace:dir:attribute-def:eduPersonScopedAffiliation">
   <Value release="deny">member@example.edu</Value>
</Attribute>
```

[ARP Shib]

Die University Of Southern California hat für den Shibboleth 1.3 Identity Provider den Shibboleth ARP Rule Constraint Patch herausgegeben, welcher erlaubt, in den ARPs Bedingungen (Constraints) zur Herausgabe von Attributen in Abhängigkeit von Attributwerten anderer Attribute zu stellen. Eine solche Bedingung hat die Struktur:

```
<Constraint name= "urn:attributeURN">
        <AnyValue release="permit"|"deny"
         matchFunction urn:functionURN>attributeValue</Value>
</Constraint>
```

Ziel der Constraints ist es, die Attributfreigabe teilweise oder sogar ganz unterdrücken zu können, z.B. sollte im Vorfeld schon klar sein, dass ein Benutzer für einen gewünschten Dienst nicht autorisiert ist. Eine unnötige Freigabe der Attribute kann somit verhindert werden. In Verbindung mit einer ARP Regel ergibt sich folgendes Beispiel, in dem die Herausgabe von Attributen komplett unterdrückt wird, sollte der Benutzer für den Dienst ServiceX nicht autorisiert sein:

```
1  <AttributeReleasePolicy>
2          <Rule>
3                  <!--Release attributes only to authorized users -->
4                  <Constraint name="attribute-def:eduPersonEntitlement">
5                          <Value release="permit">urn:x:foo</Value>
6                  </Constraint>
7                  <Target>
8                          <Requester>ServiceX</Requester>
9                  </Target>
10                 <Attribute name="urn:attributeURN">
11                         <AnyValue release="permit" matchFunction
12                         rn:functionURN>attributeValue</Value>
13                 </Attribute>
14         </Rule>
15 </AttributeReleasePolicy>
```

Hat der Benutzer im obigen Beispiel nicht den Anspruch (eduPersonEntitlement), welcher durch den Attributwert `urn:x:foo` festgelegt wird, auf den Dienst ServiceX (Zeilen 7 - 9), so wird der Release Wert für die Bedingung (Zeilen 4 - 6) auf `deny` gesetzt, wodurch alle Attribute dieser Regel (Zeilen 10 - 13) gesperrt werden. [ARP Patch]

5.1.4 Eignung

In diesem Abschnitt wird untersucht, welche der genannten Anforderungen (siehe 4.2) von den ARPs in Shibboleth umgesetzt werden können und welche nicht.

Die Anforderung (1) stellt eine Grundvorraussetzung an die Funktionsweise von ARPs und wird von Shibboleth unterstützt.

Die Abhängigkeit der Attributfreigabe von seinem Attributwert wird in Shibboleth unterstützt. Durch das Tag `<AnyValue release=,permit'/>` werden alle Werte eines Attributs freigegeben. Durch `<AnyValue release=,deny'/>` werden alle Werte gesperrt. Durch das Tag `<Value>` lassen sich bestimmte Attributwerte einschränken. `<Value release=,deny'> member@example.edu </Value>` erlaubt z.B. nicht die Freigabe des Wertes `member@example.edu`. Neben genauen Wertangaben können auch reguläre Ausdrücke verwendet werden.

Auch Anforderung (3) wird von Shibboleth unterstützt. Innerhalb des `Target` Elements kann die Attributfreigabe sowohl vom anfragenden Service Provider (`<Requester>`) als auch von der angefragten URL (`<Resource>`) abhängig gemacht werden. Ab Shibboleth Version 1.2 wird nur noch `<Requester>` verwendet. Durch reguläre Ausdrücke oder durch exakte Angaben können so Ressourcen und Service Provider eingeschränkt werden. [ARP Shib]

Die Abhängigkeit von der Handlung und vom Verwendungszweck der Attribute werden in Shibboleth nicht unterstützt.

Attributübergreifende Bedingungen sind in der momentanen Shibboleth-Version 1.3c ohne externen Patch nicht möglich. In der Shibboleth-Version 2.0 werden die Constraints jedoch voraussichtlich im Umfang enthalten sein. Daher gilt diese Anforderung als erfüllt.

In Shibboleth ist es nicht möglich ARPs eines Benutzers von Attributen anderer Benutzer abhängig zu machen. Dies erschließt sich allein schon aus der Tatsache, dass Abhängigkeit (7) nicht erfüllt wird.

Es ist auch nicht möglich die Attributfreigabe von externen Informationen abhängig zu machen. So kann sich die Freigabe z.B. nicht auf die Tageszeit beziehen.

Zunehmend in das Blickfeld des Privacy Policy Managements rückt die Forderung nach Obligationen, also was darf mit den erlangten Informationen getan werden. An wen dürfen diese weitergegeben werden, oder wie lange dürfen sie gespeichert und verwendet werden. Obligationen sind in Shibboleth derzeit nicht möglich.

Die in Szenario 7 beschiebene Kombination von ARPs lässt sich nicht umsetzen, da die in Shibboleth mögliche Zweiteilung in Site- und User-ARP für diese Zwecke nicht ausreicht.

Die nicht unterstützten Anforderungen im Überblick:

- (4) Abhängigkeit von der Handlung
- (5) Abhängigkeit vom Zweck
- (7) Abhängigkeit zu anderen Benutzern
- (8) Abhängigkeit von externen Informationen, z.B. Datum
- (9) Obligationen
- (10) Kategorien von Attributen
- (11) Kategorien von Nutzern
- (12) Kombination von Policies
- (13) Rollenmodell

Bezieht man die Untersuchung auf die Sprachelemente, so können Shibboleth ARPs nicht:

- Attributkategorien
- Zweck
- Handlungen
- Obligationen
- Nutzerkategorien

Von der Liste der benötigten Sprachelemente unterstützen die Shibboleth ARPs nur drei, nämlich Attribut, Nutzer und (komplexe) Bedingung. Daraus ergeben sich drei mal sieben, also 21 Punkte.

Da die Shibboleth ARPs vollständig in Shibboleth integriert sind, gibt es bei der Implementierung eines PDP die volle Punktzahl.

Durch die eingeschränkte Unterstützung der Kombination von Policies, wird von maximal 10 Punkten die Hälfte vergeben.

Für den Zugriff auf Nicht-Benutzer-Informationen und für das Rollenmodell gibt es jeweils 0 Punkte.

Wendet man den Kriterienkatalog auf die ARPs von Shibboleth an, so ergeben sich insgesamt 50/100 Punkte:

ANFORDERUNG	Bewertung
Unterstützung der Sprachelemente	21
Implementierung eines PDP	24
Kombination von Policies	5
Zugriff auf Nicht-Benutzer-Informationen	0
Rollenmodell	0

Tabelle 5.1: Bewertung Shibboleth-ARP

5.2 PONDER

5.2.1 PONDER im Überblick

PONDER ist eine verbreitete, objektorientierte, deklarative und streng typisierte Policy Sprache, entwickelt von der Policy Research Group, für die Sicherheit und das Management verteilter Systeme. PONDER spezifiziert folgende Policy Typen:

- Autorisierungs-Policies, definieren erlaubte Handlungen
- Ereignis-gesteuerte Obligation-Policies, definieren erzwungene Obligationen
- Unterlassungs-Policies, definieren unerlaubte Handlungen
- Delegations-Policies, definieren welche Autorisierungen (an wen) delegiert werden können

Die verschiedenen Policy Typen können miteinander in zusammengesetzten Policies aggregiert werden.

5.2.2 Beispiel

In diesem Autorisierungs-Beispiel wird dem Service Provider `example.com` nicht erlaubt, zwischen 12 und 13 Uhr, mit der Methode `read()` auf das Attribut `e-mail` zuzugreifen. Die Begriffe zu Beginn jeder Zeile sind dabei in PONDER definierte Schlüsselwörter.

```
1 inst auth-     policyName {
2 subject        /example.com;
3 target         <Attribute> /e-mail;
4 action         read() ;
5 when           time.between(1200, 1300);
6 }
```

Das nächste Beispiel zeigt eine Obligation-Policy, welche festlegt, dass bei Herausgabe des Attributs `e-mail`, dieses nach Semesterende wieder gelöscht werden muss.

```
1 inst oblig     policyName {
2 on             release(e-mail);
3 do             deleteAfterTerm();
4 }
```

5.2.3 Eignung

Betrachtet man die von uns benötigten Sprachelemente, so unterstützt PONDER:

- Attribut: `target`
- Attributkategorie: durch hierarchische Pfadangaben
- Bedingung: `constraint`
- Handlung: `action`
- Obligation: `oblig`
- Nutzer: `subject`
- Nutzerkategorie: durch hierarchische Pfadangaben

Nicht spezifiziert werden kann der Verwendungszweck. Deshalb werden von möglichen 56 Punkten für die Sprachelemente sieben Punkte abgezogen.

Ein Nachteil für den praktischen Einsatz von PONDER ist die nicht XML basierte EBNF Syntax.

Für PONDER gab es bis vor Kurzem eine open source Implementierung von der Policy Research Group, die neben einem „Toolkit“ auch einen PDP enthält. Im Toolkit enthalten waren ein Editor, ein Compiler und ein Browser zur grafischen Übersicht. Leider kann die open source Implementierung nicht mehr heruntergeladen werden, da die Policy Research Group diese nicht länger unterstützt. Daher werden hier 0 Punkte vergeben.

In PONDER ist es möglich, verschiedene Policies zu kombinieren, in dem man diese z.B. zu einer Gruppe zusammenfasst (mit Hilfe des Elements `group`). Es werden jedoch keine Kombinationsalgorithmen spezifiziert, also wie diese Gruppen dann ausgewertet werden. Deshalb gibt es für diesen Punkt 5 von 10 Punkten.

PONDER unterstützt Rollen (`role`) und den Zugriff auf Nicht-Benutzer-Attribute, z.B. Zugriff auf die Systemzeit. Für die Bewertung ergeben sich daraus jeweils 5 Punkte.

Daraus ergibt sich folgende Bewertung:

ANFORDERUNG	Bewertung
Unterstützung der Sprachelemente	49
Implementierung eines PDP	0
Kombination von Policies	5
Zugriff auf Nicht-Benutzer-Informationen	5
Rollenmodell	5

Tabelle 5.2: Bewertung PONDER

Insgesamt erhält PONDER 64/100 Punkte.

[PONDER]

5.3 EPAL

5.3.1 Überblick EPAL

EPAL steht für „Enterprise Privacy Authorization Language“.

EPAL, entwickelt von IBM, ermöglicht eine automatische Umsetzung von Datenschutzregeln. Mit Hilfe von EPAL kann festgelegt werden, wer welche personenbezogenen Daten zu welchem Zweck wie verarbeiten oder nutzen darf. Hierfür können Datenkategorien, Nutzerkategorien, Verarbeitungszwecke und Gruppen von Handlungen, Obligationen und Bedingungen definiert und Regeln gebildet werden.

EPAL ermöglicht ein automatisiertes Datenschutzmanagement über Applikations-, Abteilungs- und sogar Firmengrenzen hinaus. Dabei können die Anwender personenbezogene Daten weiterhin auf unterschiedlichen Plattformen und mit diversen Anwendungen dezentral erheben, speichern und verarbeiten.

EPAL-Datenschutzinformationen werden in XML ausgedrückt. EPAL ist zur Anerkennung als internationaler Standard des W3C eingereicht.

5.3.2 Beispiel

Folgendes Beispiel beschreibt die Elemente einer EPAL Policy. Das `epal-policy` Element ist das Hauptelement und kann Unterelemente enthalten, wie z.B.:

- `policy-information` : beschreibt die Policy (Zeilen 3 - 9)
- `epal-vocabulary-ref` : bezieht sich auf das zu verwendende EPAL Vokabular (Zeilen 11/12)
- `condition` : beschreibt Bedingungen (Zeile 14)
- `rule` : beschreibt die Autorisierungsregeln der Policy (Zeilen 16 - 28)

Das EPAL-Vokabular legt alle Elemente fest, die später in den EPAL Policies verwendet werden können. Das EPAL-Vokabular besteht z.B. aus:

- einem `vocabulary-information` Element, welches das Vokabular beschreibt
- keinem oder mehreren `user-category` Elementen, welche die Nutzerkategorien des Vokabulars festlegen (Zeile 17)
- keinem oder mehreren `data-category` Elementen, welche die Datenkategorien (Attributkategorien) des Vokabulars festlegen (Zeile 18)
- keinem oder mehreren `purpose` Elementen, welche die Zwecke des Vokabulars definiert (Zeile 19)
- keinem oder mehreren `action` Elementen, welche die Handlungen des Vokabulars definiert (Zeile 20)
- keinem oder mehreren `container` Elementen, die abstrakte Daten des Vokabulars definieren, welche von Bedingungen ausgewertet werden können
- einem oder mehreren `obligation` Elementen, welche Obligationen definieren, die vom Vokabular zurückgegeben werden können (Zeilen 22 - 26)

```
<epal-policy default-ruling="allow" ...>

  <policy-information>
    <issuer>
      <organization>LMU</organization>
      <e-mail>example@lmu.de</e-mail>
    </issuer>
    <location>http://www.lmu.de</location>
  </policy-information>

  <epal-vocabulary-ref id="id1" location="http://www.lmu.de"
      revision-number="1.0"/>

  <condition id="allowedReleaseValues">...</condition>

  <rule id="Rule1" ruling="allow">
      <user-category refid="Students"/>
      <data-category refid="StudentInformation"/>
      <purpose refid="Registration"/>
      <action refid="Read"/>

      <obligation refid="DeleteAfterTerm">
        <parameter refid="oblig1">
          <value>StudentInformation</value>
        </parameter>
      </obligation>

  </rule>

</epal-policy>
```

5.3.3 Eignung

EPAL beschreibt alle für die Szenarien benötigten Sprachelemente. Somit erhält EPAL 56 Punkte für die Sprachelemente.

Für EPAL existiert keine open source Implementierung eines PDP. Daher 0 Punkte.

EPAL spezifiziert nicht, wie Policies kombiniert werden können, erhält somit keine Punkte.

Es werden Funktionen unterstützt, die Vergleiche mit dem Datum oder der Systemzeit anstreben, daher werden für den Zugriff auf Nicht-Benutzer-Informationen 5 Punkte vergeben.

EPAL unterstützt kein Rollenmodell, somit wieder 0 Punkte. [EPAL Specification]

Daraus ergibt sich folgende Bewertung:

ANFORDERUNG	Bewertung
Unterstützung der Sprachelemente	56
Implementierung eines PDP	0
Kombination von Policies	0
Zugriff auf Nicht-Benutzer-Informationen	5
Rollenmodell	0

Tabelle 5.3: Bewertung EPAL

Insgesamt erhält EPAL 61/100 Punkte.

5.4 P3P

5.4.1 Überblick P3P

Das XML-basierte P3P, entwickelt von W3C, steht für „Platform for Privacy Preferences“. Während EPAL personenbezogene Datenflüsse in Organisationen kontrolliert, klärt P3P Internet-Nutzer darüber auf, wie Organisationen ihre Daten verarbeiten. P3P ist eine Frontend-Lösung, die Versprechungen durch Datenschutzbestimmungen gibt. Die Umsetzung dieser Bestimmungen fällt nicht in den Bereich von P3P, sondern ist z.B. Aufgabe von EPAL (Backend-Lösung).

Mit dem vom internationalen Standardisierungsgremium W3C standardisierten P3P lassen sich Datenschutz-Richtlinien von Websites automatisch auswerten. Der Browser eines Benutzers holt sich vor dem Zugriff auf eine Website dessen Richtlinien und vergleicht diese mit den lokalen Benutzerrichtlinien. Bei Übereinstimmung der Richtlinien gewährt der Browser den Zugriff, andernfalls wird der Benutzer darauf hingewiesen, dass eine Unstimmigkeit vorliegt. Ändert ein Unternehmen seine Richtlinien, wird dies von clientseitigen P3P-Tools (User Agents) dem Internet-Nutzer bei dessen nächsten Besuch der Website angezeigt. P3P klärt Fragen wie: Werden die persönlichen Nutzerangaben weiterverarbeitet? Werden die Daten an Dritte weitergegeben? Zu welchem Zweck werden die Daten verarbeitet? Die Spezifikation von P3P beschränkt sich auf die Seite des Service Providers. Wie User Agents implementiert werden sollen, wird nicht spezifiziert.

5.4.2 Beispiel

Im folgenden Beispiel werden die Hauptelemente einer P3P Policy dargestellt. Das Hauptelement `POLICIES` fasst mehrere `POLICY` Elemente zusammen. ein `POLICY` Element enthält eine komplette P3P Policy, bestehend aus:

- `ENTITY`: beschreibt den Service Provider (Zeilen 5 - 9)
- `ACCESS`: beschreibt den Zugriff auf Informationen (Zeile 11)
- `STATEMENT`: beschreibt, was mit den Benutzerinformationen geschieht (Zeilen 13 - 21)
  - `PURPOSE`: beschreibt den Zweck (Zeile 14)
  - `RECIPIENT`: gibt die Nutzer der Informationen an (Zeile 15)
  - `RETENTION`: beschreibt Obligationen (Zeile 16)

- `DATA-GROUP`: beschreibt die zu sammelnden Datentypen (Zeilen 17 - 20)

Die TU München (Zeile 5 - 9) fordert die Attribute Fakultät und E-Mail Adresse (Zeilen 18, 19) an, um mit diesen Daten den Benutzer des Dienstes anschreiben zu können (Zeile 14):

```
1  <POLICIES>
2
3   <POLICY>
4
5    <ENTITY>
6     <DATA-GROUP>
7      <DATA>TU Muenchen</DATA>
8     </DATA-GROUP>
9    </ENTITY>
10
11   <ACCESS><nonident/></ACCESS>
12
13   <STATEMENT>
14    <PURPOSE><contact/></PURPOSE>
15    <RECIPIENT><ours/></RECIPIENT>
16    <RETENTION><stated-purpose/></RETENTION>
17    <DATA-GROUP>
18     <DATA ref="faculty"/>
19     <DATA ref="e-mail"/>
20    </DATA-GROUP>
21   </STATEMENT>
22
23  </POLICY>
24
25 </POLICIES>
```

[P3P Specification]

5.4.3 Eignung

P3P spezifiziert ein festes Vokabular, d.h. Elemente haben vordefinierte Inhalte. Das Element `RECIPIENT` kann z.B. nur mit den Tags `ours`, `delivery`, `same`, `other-recipient`, `unrelated` oder `public` (siehe dazu [P3P Specification]) belegt werden. Fügt man benutzerdefinierte Elemente hinzu, so entsprechen diese nicht mehr der Spezifikation. Beim organisationsübergreifenden Einsatz von P3P stellt dies ein Problem dar, da neue Elemente nur in Absprache mit allen teilnehmenden Organisationen mit teils erheblichem Aufwand eingeführt werden können. Wünschenswert wäre jedoch ein Standard, der alle gewünschten Funktionalitäten unterstützt, damit keine nachträglichen Umstrukturierungen getroffen werden müssen. In P3P können weder Handlungen, also z.B. Daten dürfen nur gelesen werden, noch (komplexe) Bedingungen beschrieben werden, wodurch der Zugriff auf Nicht-Benutzer-Informationen, um komplexere Bedingungen zu formulieren, ebenfalls entfällt. Unterstützt werden Attribute (Zeilen 18, 19) und Attributgruppen (Zeilen 17 - 20), der Zweck (Zeile 14), Obligationen (Zeile 16) und Nutzer (Zeilen 5 - 9 und Zeile 15). Für jeden dieser Elemente gibt es 7 Punkte, insgesamt also 35 Punkte.

P3P-Funktionalität erhält der Benutzer durch die Installation eines P3P-Agent. P3P-Agents können über unterschiedliche technische Konzepte realisiert sein:

- Im Browser eingebaute P3P-Agents
  - Mozilla Browser ab Version 1.4
  - Netscape Navigator ab Version 7.0
  - Microsoft Internet Explorer ab Version 6.0
- Plug-In für Microsoft Internet Explorer ab Version 5.01 (AT&T Privacy Bird)
- Proxy-Server (P3P-Agent JRC Proxy)

[P3P]

Die oben genannten P3P-Agenten sind frei zugänglich und ermöglichen durch unterschiedliche Technologien die Nutzung von P3P. Daher werden für die P3P Implementierung 24 Punkte vergeben.

In P3P wird die Kombination von Policies nicht unterstützt. Die Spezifikation geht davon aus, dass für jede geschützte Seite genau eine Policy vorliegt. Sollten in Ausnahmefällen zwei Policies für eine Seite vorliegen, so MÜSSEN beide, für einen erfolgreichen Zugriff, eine positive Auswertung liefern. Für die Kombination von Policies werden hier also keine Punkte vergeben.

Für das Fehlen eines Rollenmodells gibt es ebenfalls 0 Punkte.

Es ergibt sich folgende Bewertung:

ANFORDERUNG	Bewertung
Unterstützung der Sprachelemente	35
Implementierung eines PDP	24
Kombination von Policies	0
Zugriff auf Nicht-Benutzer-Informationen	0
Rollenmodell	0

Tabelle 5.4: Bewertung P3P

Insgesamt erhält P3P 59/100 Punkte.

5.5 E-P3P

5.5.1 Überblick E-P3P

Mit P3P lassen sich privacy Aussagen treffen, welche fairen Umgang mit Informationen versprechen. Mit P3P werden bloße Versprechen gegeben, für dessen Umsetzung jedoch keine wohl definierte privacy Architektur und Semantik im Standard definiert ist. E-P3P (The Platform for Enterprise Privacy Practices) wurde von IBM entwickelt und wird zur XML-basierten Beschreibung unternehmensinterner privacy Policies verwendet. E-P3P definiert ein feingranulares privacy policy Model. Im Gegensatz zu P3P hat E-P3P eine wohl definierte Architektur und Semantik, womit die Umsetzung der gegebenen privacy Versprechen erzwungen werden kann. P3P und E-P3P können zusammen verwendet werden, indem mit P3P Datenschutz-Richtlinien veröffentlicht werden, welche mit E-P3P erzwungen werden. Im Gegensatz zu P3P verwendet E-P3P kein festes Vokabular. E-P3P ist das algebraische Gegenstück und Vorgänger von EPAL. E-P3P ist eine Untermenge von EPAL.

5.5.2 Beispiel

Eine E-P3P Policy ist definiert durch das Tupel „Policy := (Terms, Rules)“ . Die Terms stellen das Vokabular der Policy dar. Das Vokabular besteht aus:

- Hierarchien von Datenkategorien
- hierarchische Verwendungszwecke
- hierarchische Daten-Nutzer
- Handlung
- Obligation
- Bedingung

Die Rules stellen eine Menge von Autorisierungsregeln und default Regeln dar, welche Handlungen auf Daten erlauben oder verhindern. Eine Autorisierungsregel besteht aus den Sprachelementen:

- Priorität (Regeln können andere Regeln überschreiben): `precedence`
- Datenkategorie (hierarchische Datenstruktur): `data-category`
- Zweck: `purpose`
- Datennutzer: `data-user`
- Ruling (allow/deny): `ruling`
- Handlung (z.B. read/write): `operation`
- Obligationen: `obligation`
- Bedingungen (boolsche Ausdrücke in XSLT-Format): `condition`

Das Beispiel erlaubt es der LMU (Zeile 3) auf die E-Mail Daten des Benutzers (Zeile 5) zum Zweck der Autorisierung (Zeile 7) zuzugreifen. Die LMU hat lesenden Zugriff (Zeile 9) auf die Daten und muss diese spätestens nach Kursende löschen (Zeilen 14 - 16). Als Bedingung wird die E-Mail Adresse nur herausgegeben, wenn sie von der Domain informatik.uni-muenchen.de stammt (Zeile 11).

```
1  <rule id="rule" precedence="5" ruling="allow">
2
3          <data-user id="LMU"/>
4
5          <data-category id="email"/>
6
7          <purpose id="authorization"/>
8
9          <operation id="read"/>
10
11         <condition id="@informatik.uni-muenchen.de"/>
12         <condition .../>
13
14         <obligation id="retention">
15                 <parameter id="term">endOfCourse</parameter>
16         </obligation>
17
18 </rule>
```

5.5.3 Eignung

Von E-P3P werden alle Anforderungen an Sprachelemente erfüllt. Daraus folgt volle Punktzahl.

Für E-P3P gibt es keine open source Implementierung eines PDPs, daher 0 Punkte.

E-P3P unterstützt Prioritäten auf Policies (Zeile 1 `precedence`), d.h. auch die Anforderung an die Kombination von Policies ist erfüllt und erhält volle Punkzahl. Obwohl E-P3P eine Untermenge von EPAL darstellt, erlangt E-P3P dadurch insgesamt eine höhere Gesamtpunktzahl als EPAL, da EPAL die Kombination von Policies nicht unterstützt.

Wie mit Hilfe von E-P3P Policies externe Informationen zugegriffen werden kann, ist im Standard selbst nicht spezifiziert. Da in E-P3P jedoch XSLT verwendet werden kann (und somit auch X-Path), kann z.B. durch X-Path Funktionen wie `fn:current-date()` oder `fn:current-time()` auf Datum oder Zeit zugegriffen werden. Daher erhält E-P3P 5 Punkte.

Rollen werden in Verbindung mit E-P3P zwar angesprochen, jedoch nicht explizit spezifiziert. Daher 0 Punkte. [E-P3P]

Daraus ergibt sich folgende Bewertung:

ANFORDERUNG	Bewertung
Unterstützung der Sprachelemente	56
Implementierung eines PDP	0
Kombination von Policies	10
Zugriff auf Nicht-Benutzer-Informationen	5
Rollenmodell	0

Tabelle 5.5: Bewertung E-P3P

Insgesamt erhält E-P3P 71/100 Punkte.

5.6 XACML

5.6.1 XACML im Überblick

XACML, entwickelt von OASIS, steht für „extensible access control markup language“. Mit XACML können plattformunabhängig feingranulare Autorisierungsentscheidungen getroffen werden. XACML steht in engem Zusammenhang mit SAML (Security Assertion Markup Language) und ist eine Erweiterung dieser Sprache. XACML legt fest, wie Policy-Informationen für die Zugriffskontrolle aussehen und übermittelt werden. Die Spezifikation von XACML definiert sowohl die Syntax und Semantik von Policies, als auch die Anfragen an den PDP und die Antworten vom PDP. Weiterhin wird spezifiziert, wie ein PDP eine Policy auswerten muss. [XACML Specification]

5.6.2 Beispiel

Jede XACML `Policy` besteht aus Regeln (`Rules`). `RuleCombiningAlgortithms` bestimmen, wie mehrere Regeln zusammen ausgewertet werden, z.B. `PermitOverrides`. Regeln und Policies können ein Ziel (`Target`) spezifizieren, welche Eigenschaften mit dem aktuellen Anforderer von Attributen übereinstimmen muss, um eine erfolgreiche Auswertung zu erzielen. Jede Regel

hat entweder den Effekt `deny` oder `permit`. Regeln beschreiben die zu schützenden Ressourcen (`Resources`), Subjekte (`Subjects`), deren Aktionen `Actions` und haben optional eine Bedingung (`Condition`).

In einer XACML Attribute Release Policy kann typischerweise festgelegt werden:

- Kombinationsalgorithmus: `RuleCombiningAlg=,permit-overrides‘`
- Gruppierungen von Attributen: `Resources`
- eindeutige Attributspezifikationen: `AttributeId`
- erlaubte Anfragende: `Subjects`
- Aktionen: `Actions`
- Bedingungen: `Conditions`
- Obligationen: `Obligations`

[XACML FIM] [XACML Specification]

Folgendes Beispiel gibt einen Einblick in die Syntax der Policies von XACML. Die Regel erlaubt dem Subjekt TU (Zeilen 6 - 20) lesenden Zugriff (Zeilen 36 - 50) auf die E-Mail Adresse des Benutzers (Zeilen 22 - 34), falls die Anfrage von der Fakultät Informatik stammt (Zeilen 54 - 65). Sollten für diese Anfrage, also Zugriff auf E-Mail Adresse, mehrere Policies existieren, so wird der Kombinierungsalgorithmus „permit-overrides“ verwendet (Zeile 1). Mit dem AttributeDesignator (z.B. Zeilen 15 - 17) werden die Attribute jeweils eindeutig bestimmt.

```
1  <Policy RuleCombiningAlg="permit-overrides">
2
3    <Rule>
4
5      <Target>
6        <Subjects>
7          <Subject>
8            <SubjectMatch
9             MatchId="function:string-equal">
10
11           <AttributeValue DataType="XMLSchema#string">
12             TU
13           </AttributeValue>
14
15           <SubjectAttributeDesignator
16            AttributeId="subject:subject-id"
17            DataType="http://www.w3.org/2001/XMLSchema#string"/>
18           </SubjectMatch>
19         </Subject>
20       </Subjects>
21
22       <Resources>
23         <Resource>
24           <ResourceMatch MatchId="function:anyURI-equal">
25
26             <AttributeValue DataType="XMLSchema#anyURI">
27               //identifier/of/email/address
28             </AttributeValue>
29
30             <ResourceAttributeDesignator DataType="XMLSchema#anyURI"
```

```
31                AttributeId="resource:resource-id"/>
32              </ResourceMatch>
33            </Resource>
34          </Resources>
35
36          <Actions>
37            <Action>
38              <ActionMatch
39               MatchId="function:string-equal">
40
41                <AttributeValue   DataType="XMLSchema#string">
42                  read
43                </AttributeValue>
44
45                <ActionAttributeDesignator
46                 AttributeId="action:action-id"
47                 DataType="XMLSchema#string"/>
48              </ActionMatch>
49            </Action>
50          </Actions>
51
52      </Target>
53
54      <Condition  FunctionId="function:string-equal">
55
56          <Apply  FunctionId="string-one-and-only">
57            <SubjectAttributeDesignator  DataType="XMLSchema#string"
58             AttributeId="group"/>
59          </Apply>
60
61          <AttributeValue  DataType="http://www.w3.org/2001/XMLSchema#string">
62            computer science
63          </AttributeValue>
64
65      </Condition>
66
67    </Rule>
68
69 </Policy>
```

5.6.3 Eignung

XACML kann als Obermenge von EPAL betrachtet werden. Beide sind plattformunabhängige Sprachen, die direkt-erzwingbare Policies unterstützen, d.h. auf eine Anfrage eine Policyentscheidung treffen können. Tabelle 5.6 vergleicht XACML mit EPAL, woraus die Mächtigkeit von XACML ersichtlich wird.

EIGENSCHAFT	UNTERSCHIED zwischen EPAL und XACML
Entscheidungsanfragen	EPAL unterstützt eine Untermenge von XACML
Regeln	EPAL unterstützt eine Untermenge von XACML
Anwendbarkeit von Regeln	EPAL unterstützt eine Untermenge von XACML
Bedingungen	Äquivalent
Verschachtelte Policies	Von EPAL nicht unterstützt
Ergebniskonflikt	EPAL unterstützt eine Untermenge von XACML
Policyreferenzen	Von EPAL nicht unterstützt
Vokabular	XACML unterstützt eine Untermenge von EPAL
Attributmapping	EPAL unterstützt eine Untermenge von XACML
Attributauffindung	Äquivalent
XML Attributwerte	Von EPAL nicht unterstützt
hierarchische Rollen	Von EPAL nicht unterstützt
hierarchische Kategorien	Von XACML nicht unterstützt
hierarchische Ressourcen	EPAL unterstützt eine Untermenge von XACML
Attributkategorien	EPAL unterstützt eine Untermenge von XACML
Attribute mehrerer Subjekte	Von EPAL nicht unterstützt
Fehlerbehandlung	Von EPAL nicht unterstützt
Datentypen	EPAL unterstützt eine Untermenge von XACML
Funktionen	EPAL unterstützt eine Untermenge von XACML
Obligationen	EPAL unterstützt eine Untermenge von XACML
mehrfache Antworten	Von EPAL nicht unterstützt

Tabelle 5.6: Vergleich: EPAL XACML

Die Haupteigenschaften von XACML, welche EPAL nicht unterstützt, sind:

- Kombination von Policies unterschiedlicher Herausgeber
- fremde Policies in einer eigenen Policy referenzierbar
- Abhängigkeit zu anderen Benutzern (über Conditions)
- Jeder Knoten bei Anfrage einer hierarchischen Ressource ist einzeln herausgebbar
- Unterstützung von Subjekten, welche gleichzeitig in unterschiedlichen Rollen agieren
- Fehlerbehandlung, sollten z.B. Attribute fehlen
- Unterstützung von Attributwerten, welche Instanzen von XML Schema Elementen sind
- Unterstützung von zusätzlichen primitiven Datentypen, wie IP Adresse

[XACML Comparison]

XACML unterstützt bis auf den Zweck alle geforderten Sprachelemente. Für den Verwendungszweck der Datennutzung kann kein explizites Zweckattribut angegeben werden. Trotzdem kann der Zweck mittels des Action Elements beschrieben werden. Von der vollen Punktzahl werden wegen der indirekten Unterstützung 7 Punkte abgezogen.

Für einen XACML PDP gibt es eine komplett in JAVA geschriebene open source Implementierung von Sun (Version 1.2). Die Implementierung stellt Schnittstellen zur Erstellung von Anfragen und zur Behandlung und Auswertung von Policies bereit. Neben dem Quelltext wird auch eine umfangreiche Dokumentation mit Beispielen angeboten (siehe dazu auch [XACML PDP]). Der Punkt Implementierung eines PDP erhält volle Punktzahl.

Die XACML-Spezifikation definiert mehrere Kombinierungsalgorithmen. Daher wird die volle Punktzahl vergeben.

In der Spezifikation von XACML wird auch der Zugriff auf Nicht-Benutzer-Informationen festgelegt. Mit dem Attribut `urn:oasis:names:tc:xacml:1.0:environment:current-date` kann z.B. auf das momentane Datum zugegriffen werden. Daher 5 Punkte.

Rollen werden in XACML unterstützt und mit 5 Punkten gewertet. Autorisierungsentscheidungen können also anhand der Rolle eines Benutzers getroffen werden. Das Beispiel zeigt die Angabe der Rolle „Informatiker“ unter Verwendung des Attributelements:

```
1 <Attribute
2  AttributeId="attribute:role"
3  DataType="XMLSchema#string" Issuer="example.com">
4
5    <AttributeValue>
6       Informatiker
7    </AttributeValue>
8
9 </Attribute>
```

Daraus ergibt sich folgende Bewertung:

ANFORDERUNG	Bewertung
Unterstützung der Sprachelemente	49
Implementierung eines PDP	24
Kombination von Policies	10
Zugriff auf Nicht-Benutzer-Informationen	5
Rollenmodell	5

Tabelle 5.7: Bewertung XACML

Insgesamt erhält XACML 93/100 Punkte.

5.7 Zusammenfassung und Bewertungsüberblick

Fasst man die Bewertungsergebnisse aller untersuchten Policy Sprachen zusammen, so ergibt sich folgende Tabelle:

Sprache	Sprachelemente	PDP	Kombination	Nicht-Benutzer-Attribute	Rolle	Summe
Shib-ARP	21	24	5	0	0	**50**
PONDER	49	0	5	5	5	**64**
EPAL	56	0	0	5	0	**61**
P3P	35	24	0	0	0	**59**
E-P3P	56	0	10	5	0	**71**
XACML	49	24	10	5	5	**93**

Tabelle 5.8: Überblick Bewertungsergebnisse

Betrachtet man in dieser Tabelle nur die von den Policy Sprachen unterstützten Sprachelemente, so liefern sich PONDER, EPAL, E-P3P und XACML ein knappes Rennen und kämen theoretisch alle

vier im Bezug auf die Sprachelemente für unsere Zwecke in Frage. Der mit hohem Punktabzug verbundene Schwachpunkt der drei Erstgenannten ist jedoch das Fehlen eines geeigneten Open Source PDP, also der Komponente, die für die Umsetzung der beschriebenen Regeln zuständig ist und die Sprache einsetzbar macht. Betrachtet man die Gesamtpunktzahl, so hat XACML 93/100 Punkte erreicht und ist somit die beste Wahl um Policies im FIM Umfeld zu modellieren und diese umzusetzen. XACML wird somit als Policy Sprache für die zu entwickelnde policy-basierte Privacy Management Architektur verwendet.

6 Konzeption der Architektur

Dieses Kapitel beschreibt die neue policy-basierte Privacy Management Architektur für Shibboelth. Zu Beginn wird XACML und dessen Verwendungsweise im Bezug auf die abgeleiteten Anforderungen dargestellt. Danach wird neben der in Shibboleth bereits vorhandenen Architektur die neu entwickelte Architektur mit ihren Komponenten und dessen Workflow beschrieben.

6.1 XACML-basierte Policies

In diesem Abschnitt wird die Policy Sprache XACML im Detail beschrieben. Neben der Begriffsklärung werden die Hauptmerkmale von XACML erläutert und das Policy Modell von XACML vorgestellt. Im Anschluss wird die Verwendung der aus den Szenarien abgeleiteten Sprachelemente und Anforderungen durch XACML Elemente erklärt. Eine Übersicht zur Vorgehensweise der ARP-Erstellung mittels XACML rundet das Kapitel ab.

6.1.1 Begriffe in XACML

In XACML gibt es eine Vielzahl verwendeter Ausdrücke, welche hier alphabetisch aufgelistet werden:

- Aktion/Handlung (Action): eine Operation auf eine Ressource, z.B. lesen.
- Anwendbare Policy (Applicable Policy): Die Menge von Policies und Policymengen, welche für eine Entscheidungsanfrage relevant sind. D.h. alle Policies, welche im Target-Bereich mit der Entscheidungsanfrage übereinstimmen.
- Attribut: Eigenschaften eines Subjekts, einer Ressource oder einer Umgebung, welche in Prädikaten oder Zielen referenziert werden können.
- Autorisierungsentscheidung (authorization decision): Das Ergebnis der Auswertung einer anwendbaren Policy. Ein Ergebnis könnte z.B. `Permit` oder `Deny` sein.
- Bedingung (Condition): Ein Ausdruck von Prädikaten. Eine Funktion, die Wahr, Falsch oder Nichtdeterministisch zurückliefert.
- Effekt (Effect): Konsequenz einer Regel (Permit oder Deny).
- Entscheidung (Decision): Das Ergebnis der Auswertung einer Regel, Policy oder Policymenge.
- Entscheidungsanfrage (Decision request): Die Anfrage vom PEP zum PDP eine Autorisierungsentscheidung zu treffen.
- Kontext (Context): Die kanonische Repräsentation (eine auf die nötigsten Informationen reduzierte Darstellung, ohne die Allgemeingültigkeit zu reduzieren) einer Entscheidungsanfrage und einer Autorisierungsentscheidung.

- Obligation: Eine in einer Policy oder Policymenge spezifizierte Operation, welche vom PEP in Verbindung mit der Durchsetzung der Autorisierungsentscheidung ausgeführt werden soll.
- PAP (Policy Administration Point): Die Systemeinheit, die eine Policy oder eine Policymenge erstellt und pflegt.
- PDP (Policy Decision Point): Die Systemeinheit, die anwendbare Policies auswertet und eine Autorisierungsentscheidung trifft.
- PEP (Policy Enforcement Point): Die Systemeinheit, die die Zugangskontrolle überwacht, indem sie Entscheidungsanfragen stellt und Autorisierungsentscheidungen erzwingt.
- PIP (Policy Information Point): Die Systemeinheit, die als Quelle für Attributwerte dient (z.B. ein LDAP-Server).
- Policy: Eine Menge von Regeln, eine Angabe des Kombinierungsalgorithmus und optional Obligationen.
- Policymenge (PolicySet): Menge von Policies oder anderen Policymengen, Angabe eines Kombinierungsalgorithmus und optional Obligationen.
- Prädikat (Predicate): Eine Aussage über Attribute, dessen Richtigkeit ausgewertet werden kann.
- Regel (Rule): ein Ziel, ein Effekt und eine Bedingung.
- Ressource (Resource): Daten, Dienste oder Systemkomponenten.
- Subjekt (Subject): Ein Akteur, dessen Attribute von Prädikaten referenziert werden können. In den Shibboleth ARPs wird das Subjekt mit Target bezeichnet.
- Umgebung (Environment): Menge von Attributen, die für eine Autorisierungsentscheidung relevant und unabhängig von einem speziellen Subjekt, Ressource oder Aktion ist.
- Ziel (Target): Die Menge der Entscheidungsanfragen, identifiziert durch Definitionen für Ressource, Subjekt und Aktion, welche eine Regel, Policy oder Policymenge auswerten.

6.1.2 Merkmale von XACML

Die Policysprache XACML, entwickelt zur Standardisierung von Zugriffsentscheidungen und Policy Management, unterstützt folgende Hauptmerkmale:

Regel- und Policykombinierung

Regeln und Policies können in XACML zu einer Policymenge zusammengefasst werden, welche auf eine Entscheidungsanfrage des PEP nur die dazu passenden Regeln und Policies enthält. XACML unterscheidet die drei Hauptelemente `<Rule>`, `<Policy>` und `<PolicySet>`, welche die verschiedenen Kombinierungsebenen darstellen. Das Element `<Rule>` kann in Isolation ausgewertet werden und liefert einen booleschen Ausdruck. Regeln sind jedoch nicht dafür vorgesehen, von einem PDP in Isolation ausgewertet zu werden. Vielmehr können Regeln in verschiedenen Policies wiederverwendet werden, welche die Grundeinheit zur Auswertung im PDP darstellen. Das `<Policy>` Element kann eine Menge von `<Rule>` Elementen enthalten und spezifiziert eine Prozedur, wie die Auswertungsergebnisse der Regeln zu kombinieren sind. Das `<PolicySet>` Element wiederum kann eine Menge von `<Policy>` Elementen oder andere `<PolicySet>` Elemente enthalten. Auch in

diesem Element gibt es eine spezifizierte Prozedur, die die Auswertungsergebnisse kombiniert. Werden, wie bei XACML, die Auswertungsergebnisse kombiniert, so spricht man von Policykombinierung. Zu erwähnen ist hier auch das Konzept der Policyintegration, welches von XACML jedoch nicht unterstützt wird. Bei der Policyintegration werden nicht die Auswertungsergebnisse miteinander verglichen, sondern die auf eine Anfrage passenden Policies zu einer einzigen kombinierten Policy zusammengefasst. Vorteil dieser Methode wäre z.B., dass passende ARPs und AAPs bei einer Anfrage zu einer Policy kombiniert werden, aus der dann schon im Vorfeld (vor einer Übertragung von Attributen) eine Vorentscheidung getroffen werden kann.

Unterstützung von Kombinierungsalgorithmen

XACML unterstützt eine Menge von Kombinierungsalgorithmen, welche durch die Attribute `RuleCombiningAlgId` oder `PolicyCombiningAlgId` der Elemente `<Policy>` oder `<PolicySet>` identifiziert werden. Eine Kombination kann also auf Regelebene und auf Policyebene stattfinden. Definierte Standardalgorithmen sind:

- Deny-overrides
- Permit-overrides
- First-applicable (Reihenfolge der Policies wichtig)
- Only-one-applicable

Wird bei Deny-overrides ein einzelnes `<Rule>` oder `<Policy>` Element als „deny “ ausgewertet, so ist auch das kombinierte Ergebnis „deny “. Bei Permit-overrides ist die Auswertung analog. Wird der First-applicable Algorithmus angewandt, so ist das kombinierte Ergebnis, das Ergebnis der ersten anwendbaren Regel, Policy oder Policymenge. Anwendbar bedeutet, dass die Regel für eine Anfrage in Betracht gezogen werden muss. Der Only-one-applicable Algorithmus kann nur auf Policyebene verwendet werden. Gibt es aus der Menge der Policies keine anwendbare Policy oder Policymenge, so liefert der Kombinierungsalgorithmus als Ergebnis „NotApplicable“ , trifft genau eine Policy oder Policymenge zu, so liefert der Algorithmus das Auswertungsergebnis der Policy oder Policymenge. Sind mehrere Policies anwendbar, so liefert dar Algorithmus „Indeterminate“. Zusätzlich können zu Kombinierungsalgorithmen auch Parameter angegeben werden. In den eben vorgestellten Standardalgorithmen werden jedoch keine Parameter verwendet. Wenn nötig, kann XACML durch benutzerdefinierte Algorithmen erweitert werden, welche z.B. die Priorität einzelner Regeln, Policies oder Policymengen bestimmen.

Policies basierend auf Attributen

In XACML kann die Autorisierungsentscheidung u.a. von Attributen von Subjekten und von Attributen von Ressourcen abhängig gemacht werden. Attribute von Subjekten, welche im Kontext enthalten sind, können durch das Element `<SubjectAttributeDesignator>` identifiziert werden. Dieses Element enthält eine Id, welche das Attribut im Kontext eindeutig identifiziert (siehe Beispiel). Alternativ dazu kann mit Hilfe des `<AttributeSelector>` Elements ein XPath Ausdruck angegeben werden, welcher auf Attribute, im Bezug auf ihre Position im Kontext, verweisen kann. Nachdem das Attribut im Kontext identifiziert wurde, kann dessen Wert mit dem angegebenen Attributwert in der Policy verglichen werden. Attribute von Ressourcen, also im Kontext enthaltene Benutzerattribute, können durch das Element `<ResourceAttributeDesignator>` identifiziert werden. Dieses Element enthält wieder eine Id, welche das Attribut einfeutig identifiziert. Auch

hier kann mit Hilfe des Elements `<AttributeSelector>` alternativ eine XPath Anfrage gestellt werden.

```
1  <Subjects>
2    <Subject>
3      <SubjectMatch MatchId='string-equal'>
4
5        <AttributeValue DataType='#string'>
6          TU
7        </AttributeValue>
8
9        <SubjectAttributeDesignator
10         AttributeId='subject-id'
11         DataType='#string'/>
12
13     </SubjectMatch>
14   </Subject>
15 </Subjects>
```

Operationen auf Attributen

XACML bietet eine Menge von build-in arithmetischen Operationen auf Attribute von Subjekten und auf Attribute von Ressourcen. Diese Operationen können durch benutzerspezifische Operationen erweitert werden. Die Operationen können auch ineinander zu komplexeren Operationen verschachtelt werden. Operationen werden mit Hilfe des `<Apply>` Elements angegeben. Durch das Attribut `FunctionId` kann eine Operation identifiziert werden, welche auf den Inhalt des Elements angewandt werden soll. Ebenfalls unterstützt werden Operationen auf Zeit, Datum und Dauer. Daneben gibt es auch vergleichende Operationen (z.B. <) und Operationen auf boolsche Datentypen. Folgendes Beispiel gibt an, dass die gerundete Kursnote (Zeilen 9 - 13) eine 2.0 (Zeilen 4 - 7) sein muss.

```
1  <Condition>
2    <Apply FunctionId='double-equal'>
3
4      <Function FunctionId='double-one-and-only'/>
5      <AttributeValue DataType='#double'>
6        2.0
7      </AttributeValue>
8
9      <Apply FunctionId='round'>
10       <ResourceAttributeDesignator
11         AttributeId='grade'
12         DataType='#double'/>
13     </Apply>
14
15   </Apply>
16 </Condition>
```

Mehrwertige Attribute

In XACML werden mehrwertige Attribute unterstützt. D.h. wenn ein XACML PDP einen Attributwert anfrägt, kann das Ergebnis mehrere Werte beinhalten. Die Ergebniswerte werden in einem Bag gehalten. Ein Bag ist eine Menge, in der auch Duplikate enthalten sein können. Diese Situation kann

unterschiedlich behandelt werden, z.B. in dem irgendeiner der im Bag enthaltenen Attributwerte eine Regel erfüllt. XACML stellt Funktionen bereit, mit denen die Handhabung von Mehrwertigen Attributwerten geregelt werden kann. Im Beispiel bestimmt die Funktion „any-of“ (Zeile 1), dass einer der Attributwerte des explizit angegebenen Bags (Zeilen 6 - 10) den Wert „Bob“ (Zeile 4) haben muss. Dieses Beispiel wird positiv ausgewertet, da „Bob“ im Bag enthalten ist.

```
1  <Apply FunctionId='any-of'>
2    <Function FunctionId='string-equal'/>
3
4      <AttributeValue DataType='#string'>Bob</AttributeValue>
5
6      <Apply FunctionId='string-bag'>
7        <AttributeValue DataType='#string'>Alice</AttributeValue>
8        <AttributeValue DataType='#string'>Bob</AttributeValue>
9        <AttributeValue DataType='#string'>Malice</AttributeValue>
10     </Apply>
11
12 </Apply>
```

Auffinden von Policies

In XACML können Policies verteilt abgelegt werden. Für eine Entscheidungsanfrage muss die entsprechende anwendbare Policy, im Bezug auf das `<Target>` Element, gefunden werden. Dazu werden zwei Vorgehensweisen unterstützt:

- Policy Aussagen können in einer Datenbank gespeichert werden. Der PDP stellt eine Datenbankanfrage, die nur solche Policies zurückliefert, welche auf die Entscheidungsanfrage anwendbar sind.
- Der PDP wird mit allen verfügbaren Policies geladen und wertet dann dessen `<Target>` Elemente im Kontext einer Entscheidungsanfrage aus, um die relevanten Policies herauszufiltern.

Abstraktionsschicht

XACML ist durch den XACML Kontext von umgebenden Anwendungen unabhängig. Der XACML Kontext beschreibt Entscheidungsanfragen vom PEP an den PDP und Antworten vom PDP an den PEP, welche beide in einer XACML Syntax formuliert werden. In dem Fall, dass das Anfrage/Antwort Format vom PEP jedoch z.B. SAML ist, wird dieses zuerst in den XACML Kontext, z.B. mit Hilfe eines XSLT (Extensible Stylesheet Language Transformation), für den PDP übersetzt. Dieser Mechanismus hat den Vorteil, dass die Syntax der Entscheidungsanfragen/Antworten vom PEP auf kein festes Format beschränkt ist. Somit können auch PEPs zum Einsatz kommen, welche die Syntax von XACML nicht unterstützen. XACML ist somit durch den XACML Kontext von umgebenden Anwendungen unabhängig (siehe auch Abbildung 6.1). Im grauen Bereich wird XACML Syntax verwendet. In der XACML Kontext Anfrage müssen alle für die Auswertung relevanten Informationen (dies beinhaltet auch alle relevanten Attribute und Attributwerte) bereits enthalten sein, d.h. der PDP selbst muss von sich aus keine weiteren Informationen beziehen, um eine Auswertung durchführen zu können.

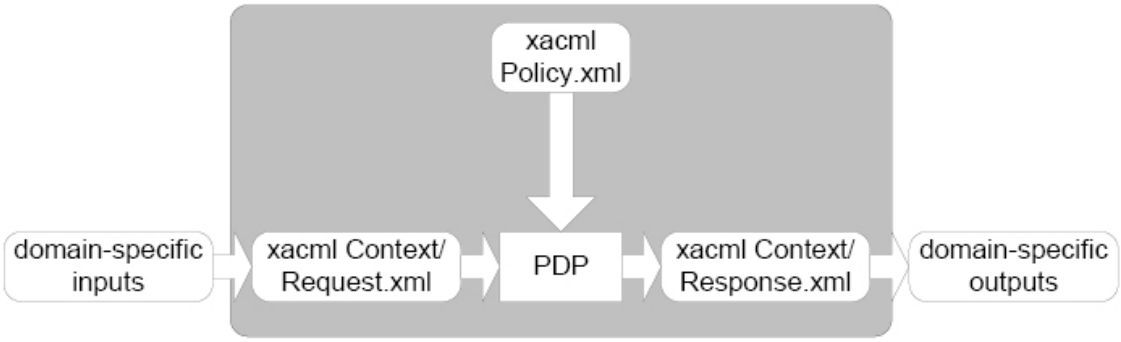

Abbildung 6.1: XACML Kontext

Obligationen

Durch das `<Obligations>` Element werden in XACML Obligationen unterstützt, also Aktionen, die in Konjunktion mit der Policyerzwingung ausgeführt werden müssen. In XACML gibt es für Obligationen keine Standarddefinitionen. Deshalb müssen zwischen dem PAP und dem PEP beidseitige Abkommen getroffen werden, damit die Obligationen richtig interpretiert werden. Eine Obligation könnte z.B. sein, dass bei einer Attributfreigabe eine E-Mail an den Benutzer verendet werden muss, um diesen darauf hinzuweisen. Verstehen PEPs eine Obligation nicht, so wird der Zugriff laut Spezifikation verweigert.

[XACML Specification]

6.1.3 Policy Modell

Abbildung 6.2 beschreibt das Policy Modell von XACML. Die Hauptkomponenten sind:

- `<Rule>`
- `<Policy>`
- `<PolicySet>`

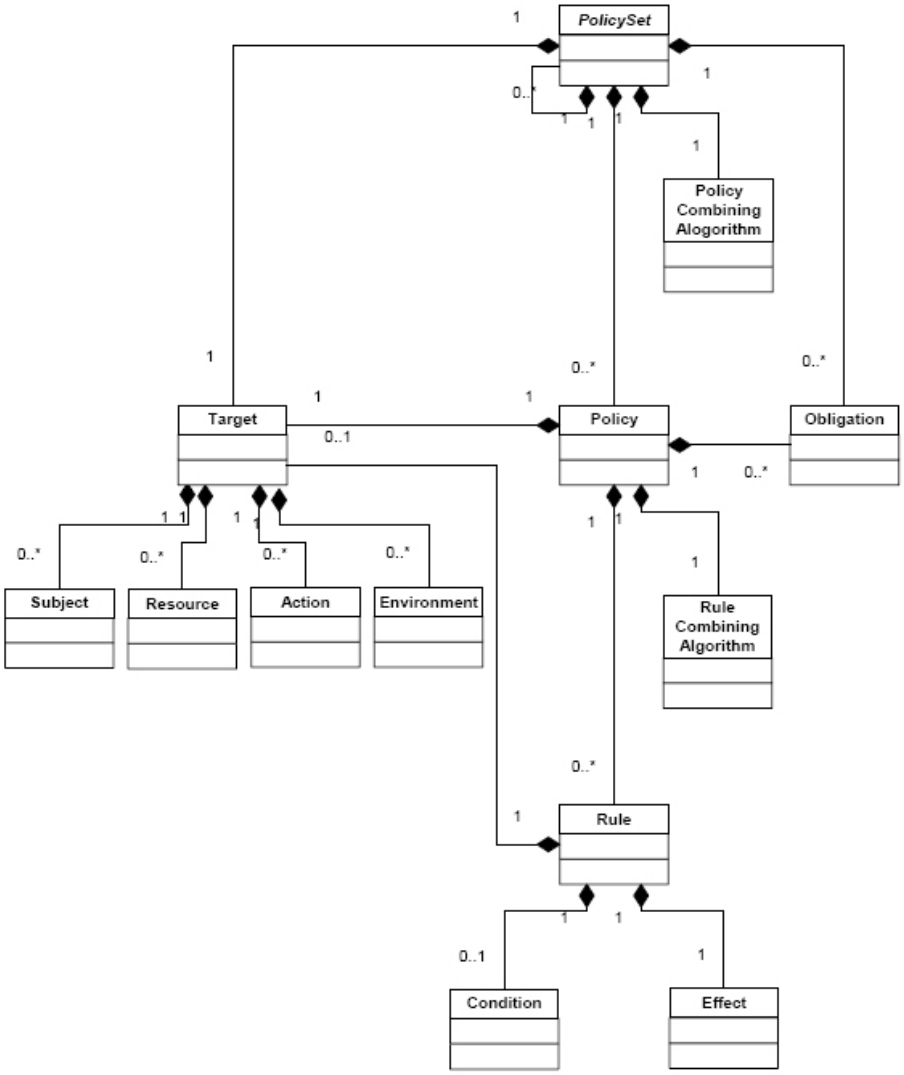

Abbildung 6.2: Policy Modell

Rule

Das Element <Rule> stellt die Grundeinheit einer Policy dar. Mehrere <Rule> Elemente werden in einem <Policy> Element gekapselt. Die Unterelemente von <Rule> sind:

- <Target> (0..1)
  - <Resources> (0..*)
  - <Subjects> (0..*)
  - <Actions> (0..*)
  - <Environment> (0..*)
- <Effect> (1)
- <Condition> (0..1)

Das <Target> Element in einer Regel gibt an, worauf eine Regel angewandt wird. Ein XACML PDP prüft nach, ob das <Target> Element einer Regel durch die im Anfragekontext mitgelieferten Attribute (Subject, Resource, Action, Environment) zufriedengestellt werden kann, und identifiziert so anwendbare Regeln. Wird das <Target> Element in einer Regel weggelassen, so wird statt dessen auf das <Target> Element des <Policy> Elements zurückgegriffen.

Das <Effect> Element gibt die Konsequenz einer zu wahr ausgewerteten Regel an. <Effect> kann entweder den Wert „Permit“ oder den Wert „Deny“ haben.

Eine Regel kann ein <Condition> Element beinhalten. Das <Condition> Element beschreibt einen booleschen Ausdruck, welcher die Anwendbarkeit einer Regel im Bezug auf das <Target> Element verfeinert.

Policy

Durch das <Policy> Element können <Rule> Elemente kombiniert werden. Es besteht aus den Komponenten:

- <Target> (1)
- rule-combining algorithm-identifier (1)
- <Rule> (0..*)
- <Obligation> (0..*)

Das <Target> Element einer Policy spezifiziert, genauso wie das <Target> Element einer Regel oder einer Policymenge, die Menge von Subjekten, Ressourcen, Aktionen und Umgebungen, auf welche die Policy angewandt wird. <Target> Elemente einer Policy oder einer Policymenge können entweder direkt angegeben werden, oder sie können aus den <Target> Elementen ihrer Kindelemente berechnet werden. XACML spezifiziert jedoch keine Komponente, welche diese Berechnung durchführt. Wird in einer Policy ein <Target> Element angegeben, und hat das Kindelement <Rule> das selbe <Target> Element, so kann dieses weggelassen werden. In diesem Fall übernimmt das <Rule> Element das <Target> der Policy.

Mit dem rule-combining algorithm-identifier kann der Kombinierungsalgorithmus von Regeln angegeben werden. Dieser berechnet, wie die Ergebnisse mehrerer Regeln einer Policy kombiniert werden. Die Algorithmen können mit Parametern versehen werden.

In einer Policy können Obligationen mit Hilfe des <Obligation> Elements hinzugefügt werden. Wertet ein PDP eine Policy mit Obligationen aus, so sendet er diese weiter an den PEP, welcher diese umsetzen muss.

PolicySet

Eine Policymenge besteht aus den vier Komponenten:

- <Target> (1)
- policy-combining algorithm-identifier (1)
- <Policy> (0..*)
- <Obligation> (0..*)

Eine Policymenge kann wiederum aus Policymengen bestehen.

Der policy-combining algorithm-identifier gibt den zu verwendenden Kombinierungsalgorithmus für Policies an. Dieser berechnet, wie die Ergebnisse mehrerer Policies einer Policymenge kombiniert werden. Die Algorithmen können mit Parametern versehen werden.

In einer Policymenge können, zusätzlich zu evtl. in untergeordneten `<Policy>` Elementen oder Policymengen enthaltenen Obligationen, Obligationen angegeben werden.

[XACML Specification]

6.1.4 Sprachelemente

In diesem Abschnitt wird gezeigt, wie die von den Szenarien geforderten Sprachelemente in XACML umgesetzt werden können. Die Beispiele geben gleichzeitig an, auf welche Weise XACML für unsere Zwecke (ARPs) verwendet werden soll.

Attribut

Benutzerattribute werden durch das Element `<Resource>` beschrieben. Im Beispiel ist die gewünschte Ressource Bobs E-Mail Adresse.

```
1  <Resources>
2    <Resource>
3      <ResourceMatch MatchId='anyURI-equal'>
4
5        <AttributeValue DataType='#anyURI'>
6          e-mail-address
7        </AttributeValue>
8
9        <ResourceAttributeDesignator
10         AttributeId='resource-id'
11         DataType='#anyURI'/>
12
13     </ResourceMatch>
14   </Resource>
15 </Resources>
```

Das Element `<ResourceMatch>` spezifiziert eine Matchfunktion, welche durch das Attribut `MatchId` identifiziert wird (Zeile 3). Das Element `<AttributeValue>` identifiziert eindeutig den Attributnamen, bei dem die Matchfunktion den Wert „True“ zurückliefert (Zeilen 5 - 7). Durch das Element `<ResourceAttributeDesignator>` (Zeilen 9 - 11) bzw. das Attribut `AttributeId` wird das zu vergleichende Element aus dem Kontext, also aus der Entscheidungsanfrage, angegeben.

Attributkategorie

In XACML gibt es zwei Möglichkeiten, Attributkategorien festzulegen. Entweder man bestimmt Attributhierarchien, auf welche man dann mit XPath Ausdrücken zugreift, oder man legt Attributgruppen fest. Ziel beider Ansätze ist die Vereinfachung der Regelbildung, da so nicht für jedes Attribut einzeln Regeln definiert werden müssen. In dieser Arbeit wird die Bildung von Attributgruppen verwendet.

In dem Beispiel wird eine Attributgruppe mit Hilfe des Elements <VariableDefinition> definiert, auf die in Regeln zurückgegriffen werden kann:

```
1 <VariableDefinition VariableId='attribute-group'>
2     e-mail-address
3   | registration-number
4   | faculty
5 </VariableDefinition>
```

Die Attributgruppe besteht aus den Attributen E-Mail Adresse, Matrikelnummer und Fakultät, welche durch den regulären Ausdruck „ODER“ verknüpft sind (Zeilen 2 - 4). Folgendermaßen kann eine einzige Regel für die verschiedenen Attribute der Attributgruppe festgelegt werden:

```
1  <Rule>
2  ...
3    <Resource>
4      <ResourceMatch MatchId='anyURI-regexp-match'>
5
6        <AttributeValue DataType='#anyURI'>
7
8          <VariableReference VariableId='attribute-group'/>
9
10       </AttributeValue>
11
12       <ResourceAttributeDesignator
13         AttributeId='resource-id'
14         DataType='#anyURI'/>
15
16     </ResourceMatch>
17   </Resource>
18 ...
19 </Rule>
```

In Zeile 8 wird die Attributgruppe durch das Element <VariableReference> referenziert. Die Regel wird mit „True “ ausgewertet, falls der Wert des Kontextelements, referenziert durch resource-id (Zeile 13), e-mail-address oder registration-number oder faculty ist.

(Komplexe) Bedingung

Mit XACML können mit dem Element <Condition> (attributübergreifende) Bedingungen an Attribute gestellt werden. Alle Benutzerattribute und deren Werte sind in der Entscheidungsanfrage im Element <ResourceContent> abgelegt, auf den mit XPath zugegriffen werden kann. Besitzt Bob mehrere E-Mail Adressen, so wird für jede Adresse eine Anfrage an den PDP gestellt. Das Beispiel zeigt, dass Bob nur die Adresse bob@example.edu freigibt:

```
1  <Condition>
2    <Apply FunctionId='string-equal'>
3
4      <Apply FunctionId='string-one-and-only'>
5        <AttributeSelector
6          RequestContextPath='//e-mail-address/text()'
7          DataType='#string'/>
8      </Apply>
9
10     <Apply FunctionId='string-one-and-only'>
11       <AttributeValue DataType='#string'>
12         bob@example.edu
13       </AttributeValue>
14     </Apply>
15
16   </Apply>
17 </Condition>
```

Das Element <Apply> gibt an, dass eine mathematische Funktion aufgerufen wird. Verschiedene <Apply> Elemente können ineinander verschachtelt werden. Durch das erste <Apply> (Zeile 2 - 16) wird die Funktion „string-equal“ aufgerufen, welche die Strings der beiden folgenden <Apply> Elemente vergleicht. Im <Apply> Element von Zeile 4 - 8 wird mit XPath auf den Attributwert der „e-mail-address“ zugegriffen, welcher im <ResourceContent> der Entscheidungsanfrage abgelegt ist. Im <Apply> Element von Zeile 10 - 14 wird der String bob@example.edu definiert.

Will man attributübergreifende Bedingungen stellen, so kann dies durch entsprechende Schachtelung mehrerer <Apply> Elemente, welche auf verschiedene Attribute des <ResourceContent> zugreifen, geschehen.

Zweck

Der Verwendungszweck von angefragten Benutzerdaten kann in XACML im Element <Action> angegeben werden. Das Beispiel gibt an, dass die angefragten Daten nur zur Autorisierung verwendet werden dürfen:

```
1  <Action>
2    <ActiontMatch MatchId='string-equal'>
3
4      <AttributeValue DataType='#string'>
5        authorization
6      </AttributeValue>
7
8      <ActionAttributeDesignator
9        AttributeId='purpose'
10       DataType='#string'/>
11
12   </ActionMatch>
13 </Action>
```

Wie es bei dem <Resource> Element das <ResourceMatch> Element gibt, so gibt es bei <Action> das <ActionMatch> Element. Hier kann der Verwendungszweck mittels <AttributeValue> (Zeilen 4 - 6) und <ActionAttributeDesignator> (Zeilen 8 - 10) angegeben

werden. In diesem Beispiel muss das Attribut der Entscheidungsanfrage, welches durch „purpose“ identifiziert wird (Zeile 9), den Wert „authorization“ (Zeile 5) haben.

Handlung

Erlaubte Handlungen auf Benutzerdaten können in XACML ebenfalls durch das Element `<Action>` beschrieben werden. Im folgenden Beispiel wird nur lesender Zugriff auf die Benutzerdaten gewährt:

```
1  <Action>
2    <ActionMatch MatchId='string-equal'>
3
4      <AttributeValue DataType='#string'>
5        read
6      </AttributeValue>
7
8      <ActionAttributeDesignator
9        AttributeId='action-id'
10       DataType='#string'/>
11
12   </ActionMatch>
13 </Action>
```

Hat das Element der Entscheidungsanfrage, welches durch „action-id“ identifiziert wird (Zeile 9), den Wert „read“ (Zeile 5), so liefert das `<Action>` Element „True“ zurück.

Obligation

Obligationen werden in XACML durch das Element `<Obligation>` beschrieben. Das Beispiel beschreibt eine Obligation, die verlangt, dass der Service Privoder, welcher Benutzerdaten (e-mail address) anfordert, in einer Log-Datei festgehalten wird, falls die dazugehörige Regel zu „Permit“ ausgewertet wird.

```
1  <Obligation ObligationId='log' FulfillOn='Permit'>
2
3    <AttributeAssignment
4      AttributeId='text'
5      DataType='#string'>
6
7      E-mail address released to:
8
9      &lt;SubjectAttributeDesignator
10       AttributeId='service-provider'
11       DataType='#string'/&gt;
12
13   </AttributeAssignment>
14
15 </Obligation>
```

Eine Obligation wird durch das Attribut `ObligationId` (Zeile 1) identifiziert. Das Attribut `FulfillOn` (Zeile 1) gibt an, wann eine Obligation ausgeführt werden muss. In diesem Fall muss die Obligation umgesetzt werden, falls die dazugehörige Regel auf eine Entscheidungsanfrage das Ergebnis „Permit“ liefert. Das Attribut `FulFillOn` kann neben „Permit“ auch mit allen anderen

möglichen Policyergebnissen belegt werden (z.B. „Deny“). Das Element <AttributeAssignment> (Zeilen 3 - 13) wird verwendet, um Argumente in der Obligation in Form von Attributen anzugeben. Durch das Attribut AttributeId wird es eindeutig identifiziert. Das Element <AttributeAssignment> hat einen komplexen Typ, d.h. es können beliebige, der XACML Syntax entsprechende, Unterattribute gebildet werden. Im Beispiel besteht das Element aus einem String, welcher in seiner Auswertung vom PEP dazu verwendet werden kann, den Service Provider mittels <SubjectAttributeDesignator> Element zu bestimmen (Zeilen 9 - 11).

Nutzer

Nutzer, bzw. Service Provider oder Dienste, welche Benutzerinformationen anfragen dürfen, werden in XACML durch das Element <Subject> beschrieben. Wie bei den Elementen <Resource> und <Action> gibt es auch hier als Unterelemente ein Match-Element, nämlich <SubjectMatch>, und ein AttributeDesignator-Element, nämlich <SubjectAttributeDesignator>. Durch Kombination mehrerer <SubjectMatch> Elemente können z.B. auch Service Provider und Dienst angegeben werden. Im Beispiel wird ein Dienst „login“ vom Service Provider „LRZ“ beschrieben.

```
1  <Subject>
2    <SubjectMatch MatchId='string-equal'>
3
4      <AttributeValue DataType='#string'>
5        login
6      </AttributeValue/>
7
8      <SubjectAttributeDesignator
9        AttributeId='service'
10       DataType='#string'/>
11
12   </SubjectMatch>
13
14
15   <SubjectMatch MatchId='string-equal'>
16
17     <AttributeValue DataType='#string'>
18       LRZ
19     </AttributeValue/>
20
21     <SubjectAttributeDesignator
22       AttributeId='service-provider'
23       DataType='#string'/>
24
25   </SubjectMatch>
26 </Subject>
```

Das erste <SubjectMatch> Element (Zeilen 2 - 12) beschreibt den entsprechenden Dienst, welcher Benutzerinformationen anfordern darf. Das zweite <SubjectMatch> Element (Zeilen 15 - 25) beschreibt den Service Provider. In dem Beispiel wird jeweils der Wert des Elements <AttributeValue> mit dem Wert des Kontextelements, welcher das Element <SubjectAttributeDesignator> liefert, durch die Funktion string-equal verglichen.

Nutzerkategorie

Nutzerkategorien können in XACML auf die selbe Weise wie Attributkategorien beschrieben werden. Durch das Element `<VariableDefinition>` können z.B. mehrere Service Provider zu einer Gruppe zusammengefasst werden. Für diese Gruppe kann dann eine Regel definiert werden, ohne dies für jeden Service Provider einzeln durchführen zu müssen. Folgendes Beispiel fasst die Service Provider LRZ, LMU und TU zu einer Gruppe zusammen, welche im Subjekt einer Regel referenziert werden kann.

```
1  <Policy>
2
3    <VariableDefinition VariableId='sp-group'>
4      LRZ|LMU|TU
5    </VariableDefinition>
6
7    <Rule>
8    ...
9      <Subject>
10       <SubjectMatch MatchId='string-regexp-match'>
11
12         <AttributeValue DataType='#string'>
13
14           <VariableReference VariableId='sp-group'/>
15
16         </AttributeValue>
17
18         <SubjectAttributeDesignator
19           AttributeId='sp-id'
20           DataType='#string'/>
21
22       </SubjectMatch>
23     </Subject>
24   ...
25   </Rule>
26
27 </Policy>
```

In den Zeilen 3 - 5 wird die Variable mit dem regulären Ausdruck `LRZ|LMU|TU` gebildet. Durch `VariableId` in Zeile 3 wird diese eindeutig bestimmt und wird wiederum durch `VariableId` in Zeile 14 referenziert. Liefert das Element `<SubjectAttributeDesignator>` (Zeilen 18 - 20) einen der Werte der Variable zurück, so gilt das Subject als befriedigt.

[XACML Specification]

6.1.5 Kombination von Policies

Wie im Abschnitt „Merkmale von XACML“ bereits erwähnt, können sowohl Regeln als auch Policies bzw. Policymengen miteinander kombiniert werden, indem z.B. mehrere Regeln in einer Policy als Unterelemente oder mehrere Policies in einer Policymenge als Unterelemente zusammengefasst und mit Kombinierungsalgorithmen versehen werden.

In der neuen Architektur werden alle Policies mit gleichem Target und übereinstimmender Rolle zu einer Policymenge zusammengefasst. Diese Policymenge wird dann zur Auswertung herangezogen.

Wie dabei die Ergebnisse der einzelnen Elemente miteinander kombiniert werden, regeln die Kombinierungsalgorithmen, welche teils in der XACML Spezifikation vordefiniert sind, aber auch erweitert oder neu entwickelt werden können.

Als Policy-Kombinierungsalgorithmus wird ein neuer Prioritätsalgorithmus implementiert, durch den es möglich wird, Policies mit Hilfe von Parametern zu gewichten, um so das Ergebnis mit höchster Priorität zu ermitteln.

Das Beispiel zeigt eine Policymenge mit dem neu implementierten Prioritätsalgorithmus und zwei unterschiedlich gewichteten Policies.

```
1  <PolicySet
2    PolicySetId='combining-policies'
3    PolicyCombiningAlgId=
4    'urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:priority'>
5     <Policy  PolicyId='user-ARP'>
6       <CombinerParameters>
7         <CombinerParameter>
8           100
9         </CombinerParameter>
10      </CombinerParameters>
11    ...
12    </Policy>
13
14    <Policy  PolicyId='default-ARP'>
15      <CombinerParameters>
16        <CombinerParameter>
17          50
18        </CombinerParameter>
19      </CombinerParameters>
20    ...
21    </Policy>
22     ...
23  </PolicySet>
```

Der Policy-Kombinierungsalgorithmus `<urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:priority>` (Zeilen 3 - 4) gibt der Policy mit der höchsten Gewichtung Vorrang. Angenommen, dass beide Policies in diesem Beispiel (Zeilen 5 - 12 und Zeilen 14 - 21) das selbe Target spezifizieren, d.h. beide sind für eine Entscheidungsanfrage relevant. Die erste Policy hat eine Priorität von 100 zugewiesen bekommen (Zeilen 6 - 10) und die zweite Policy hat eine geringere Priorität von 50 erhalten (Zeilen 15 - 19). `<urn:oasis:names:tc:xacml:1.0:-policy-combining-algorithm:priority>` würde in diesem Fall der höher priorisierten Policy, in diesem Beispiel der Benutzer Policy (Zeilen 5 - 12), Vorrang gewähren.

[XACML Specification]

6.1.6 Zugriff auf Nicht-Benutzer-Attribute

In XACML kann auf „externe Informationen“, wie Zeit und Datum, zugegriffen werden. „Externe Informationen“ bilden insofern eine Ausnahme, da sie nicht bereits im XACML Kontext enthalten sein müssen. XACML bietet für „externe Informationen“ eine Reihe von vordefinierten Attribut-Ids, Datentypen und Funktionen an. Folgendes Beispiel wird nur positiv ausgewertet, falls das aktuelle Datum den Wert des Benutzerattributs `valid-until` nicht überschritten hat.

```
1  <Condition>
2    <Apply FunctionId='date-less-or-equal'>
3
4      <Apply FunctionId='date-one-and-only'>
5        <EnvironmentAttributeDesignator
6          AttributeId='current-date'
7          DataType='#date'/>
8      </Apply>
9
10     <Apply FunctionId='date-one-and-only'>
11       <AttributeSelector
12         RequestContextPath='//valid-until/text()'
13         DataType='#date'
14     </Apply>
15
16   </Apply>
17 </Condition>
```

In dieser Bedingung werden zwei Daten durch die Funktion `date-less-or-equal` (Zeile 2 - 16) verglichen. Das erste Datum ist das aktuelle Datum und wird durch das Attribut `current-date` (Zeile 5 - 7) aus der Umgebung (`<Environment>`) der Entscheidungsanfrage bestimmt. Das zweite Datum wird aus dem `<ResourceContent>` der Entscheidungsanfrage mit Hilfe von XPath erfragt.

[XACML Specification]

6.1.7 Rollenmodell

Im Bezug auf die Umsetzung eines Rollenmodells könnte man sich zwei Möglichkeiten zur Verwirklichung vorstellen, welche der Vollständigkeit halber auch beide aufgeführt werden:

Sollen in XACML Entscheidungen anhand verschiedener Benutzerrollen getroffen werden, so lässt sich dies z.B. bewerkstelligen, indem man den Attributnamen der `Resource` Elemente einen Rollennamen mit anhängt (z.B. `at-work/e-mail-address` in Zeile 9). Auf diese Weise kann für jede mögliche Benutzerrolle eine eigene dafür vorgesehene Policy erstellt werden.

Angenommen ein Benutzer hat zwei verschiedene E-Mail Adressen. Eine gibt er nur zur Arbeitszeit frei, die andere nur in seiner Freizeit. In der Arbeit hat er die Rolle `at-work`, sonst `free-time`. Das Beispiel zeigt eine Regel, die auf die Benutzerrolle `at-work` angewandt wird.

```
1  <Rule RuleId='role1' Effect='Permit'>
2    <Target>
3      <Resources>
4        <Resource>
5
6          <ResourceMatch MatchId='string-equal'>
7
8            <AttributeValue DataType='#string'>
9              at-work/e-mail-address-1
10           </AttributeValue>
11
12           <ResourceAttributeDesignator
13             AttributeId='resource-id'
14             DataType='#string'/>
```

```
15
16            </ResourceMatch>
17
18          </Resource>
19        </Resources>
20        ...
21      </Target>
22      ...
23  </Rule>
```

In der neuen Architektur wird das Rollenmodell von XACML jedoch ausgelagert und nicht wie eben beschrieben verwendet. So wird nicht direkt in einer XACML-Policy angegeben, für welche Rolle diese gilt, sondern extern in einem LDAP-Attribut, welches neben der Policy selbst definiert werden muss (siehe dazu Beispiel 8.3.6). Ein Vorteil dieser externen Speicherung besteht z.B. darin, dass relevante Policies so schneller, mit Hilfe von LDAP-Anfragen, aufgefunden werden können. Ein weiterer Perfomance-Vorteil besteht darin, dass Policies nur dann überhaupt zur Auswertung hinzugezogen werden, wenn die in LDAP gefundene Rolle mit der Rolle des Benutzers übereinstimmt. Ob eine Policy für eine Anfrage relevant ist, kann somit schon im Vorfeld bestimmt werden, ohne dabei die Policy selbst zu analysieren.

[XACML FIM]

6.1.8 XACML-ARP HowTo

In diesem Abschnitt wird das schrittweise Vorgehen der Erstellung einer ARP erklärt. Die zu erstellende Beispiel ARP sagt aus, dass das Attribut `idp.example.com/bob/defaultrole/-e-mail-address` an den Service Provider `lmu` zum Zweck der Autorisierung (`authorization`) nur zum Lesen (`read`) herausgegeben werden darf, wenn dessen Attributwert entweder mit `@in-formatik.uni-muenchen.de` oder mit `@ifi.lmu.de` endet. Als Obligation wird angegeben, dass die Daten nach Semesterende beim Service Provider wieder gelöscht werden müssen. Im Beispiel werden auch die erforderlichen Namespaces und die kompletten URN-Bezeichnungen mit angegeben, wodurch alle nicht in eine Zeile passenden Zeichenketten mit Leerzeichen markiert wurden. Die Syntax und Semantik der einzelnen Elemente wird hier nicht mehr erklärt, da dies bereits in den vorhergehenden Abschnitten geschehen ist.

Folgende Liste schildert das schrittweise Vorgehen:

1. Zeile 1: XML-Prolog angeben
2. Zeilen 2 - 8: Policy mit Namespaces, PolicyId und Regel-Kombinierungsalgorithmus (hier `ordered-permit-overrides`) angeben
3. Zeilen 10 - 28: textuelle Beschreibung der Policy
4. Zeilen 30 - 34: muss in der Policy mit XPath auf Informationen im Kontext der Entscheidungsanfrage zugegriffen werden, muss die XPath Version im Tag `<PolicyDefaults>`angegeben werden.
5. Zeilen 36 - 71: Bestimmung des `<Target>` Elements der Policy. Im `<Target>` enthalten sind `<Subjects>`, `<Resources>` und `<Actions>`. Auf Policy Ebene wird für gewöhnlich das Subjekt und die Ressource spezifiziert. Die Aktion wird auf dieser Ebene in diesem Beispiel noch nicht spezifiziert `<AnyAction/>` (Zeile 69). Dies hat den Vorteil, dass für eine

auf Policy Ebene spezifizierte Ressource (in Verbindung mit dem entsprechenden Service Provider) verschiedene Regeln mit verschiedenen Aktionen definiert werden können. Siehe dazu nächsten Punkt.

6. Zeilen 73 - 134: Bestimmung der <Rule> Elemente der Policy. Im <Rule> Element enthalten sind <Target> (Zeilen 74 - 109) und <Condition> (Zeilen 111 - 129). Im <Target> wird das <Actions> Element spezifiziert, welches die Aktion (<read>) und den Zweck (<authorization>) beschreibt. <Subjects> und <Resources> wurden bereits in der Policy spezifiziert und werden daher mit <AnySubject> (Zeile 76) bzw. mit <AnyResource> (Zeile 80) belegt. In der Bedingung können mehrere Funktionen geschachtelt angegeben werden. Zur genauen Verwendung siehe [XACML Specification]. Die Regel in Zeile 134 gibt an, dass wenn die obige Regel nicht greift, das Attribut nicht freigegeben werden soll.

7. Zeilen 136 - 145: Bestimmung der <Obligations> der Policy. Mit Hilfe des Elements <AttributeAssignment> können die für die Obligation relevanten Attribute angegeben werden.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <Policy xmlns="urn:oasis:names:tc:xacml:1.0:policy"
3          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4          xmlns:context="urn:oasis:names:tc:xacml:1.0:context"
5          xmlns:condition="urn:mace:dir:attribute-def"
6          PolicyId="ARP2"
7          RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-
8  ␣␣␣␣␣␣␣␣combining-algorithm:ordered-permit-overrides">
9
10   <Description>
11     EINFACHE BEDINGUNG
12
13     Attribut: e-mail-adress
14     Bedingung: Einschränkung der Attributwerte
15     Zweck: authorization
16     Handlung: read
17     Obligation: Daten nach Semesterende löschen
18     Nutzer: lmu
19     Kombination: -
20     Nicht-Benutzer-Attribute: -
21     Rolle: -
22
23     Attribut e-mail-adress zur Autorisierung
24     nur zum LESEN freigeben, wenn
25     der SP die lmu ist und der Attributwert entweder
26     *.informatik.uni-muenchen.de oder
27     *.ifi.lmu.de ist.
28   </Description>
29
30   <PolicyDefaults>
31     <XPathVersion>
32       http://www.w3.org/TR/1999/Rec-xpath-19991116
33     </XPathVersion>
34   </PolicyDefaults>
35
36   <Target>
37     <Subjects>
38       <Subject>
```

```
39          <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:
40 ..........function:string-equal">
41            <AttributeValue
42              DataType="http://www.w3.org/2001/XMLSchema#string">
43               lmu
44            </AttributeValue>
45            <SubjectAttributeDesignator
46              DataType="http://www.w3.org/2001/XMLSchema#string"
47              AttributeId="service-provider"/>
48          </SubjectMatch>
49        </Subject>
50      </Subjects>
51
52      <Resources>
53        <Resource>
54          <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:
55 ........function:anyURI-equal">
56            <AttributeValue
57              DataType="http://www.w3.org/2001/XMLSchema#anyURI">
58               e-mail-adress
59            </AttributeValue>
60            <ResourceAttributeDesignator
61              DataType="http://www.w3.org/2001/XMLSchema#anyURI"
62              AttributeId=
63              "urn:oasis:names:tc:xacml:1.0:resource:resource-id"/>
64          </ResourceMatch>
65        </Resource>
66      </Resources>
67
68      <Actions>
69        <AnyAction/>
70      </Actions>
71    </Target>
72
73    <Rule RuleId="Rule1" Effect="Permit">
74      <Target>
75        <Subjects>
76          <AnySubject/>
77        </Subjects>
78
79        <Resources>
80          <AnyResource/>
81        </Resources>
82
83        <Actions>
84          <Action>
85            <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:
86 ..........function:string-equal">
87              <AttributeValue
88                DataType="http://www.w3.org/2001/XMLSchema#string">
89                 read
90              </AttributeValue>
91              <ActionAttributeDesignator
92                DataType="http://www.w3.org/2001/XMLSchema#string"
93                AttributeId=
94                "urn:oasis:names:tc:xacml:1.0:action:action-id"/>
```

```
95            </ActionMatch>
96
97            <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:
98 ␣␣␣␣␣␣␣␣␣␣␣function:string-equal">
99              <AttributeValue
100               DataType="http://www.w3.org/2001/XMLSchema#string">
101                 authorization
102             </AttributeValue>
103             <ActionAttributeDesignator
104               DataType="http://www.w3.org/2001/XMLSchema#string"
105               AttributeId="purpose"/>
106           </ActionMatch>
107         </Action>
108       </Actions>
109     </Target>
110
111     <Condition
112       FunctionId="urn:oasis:names:tc:xacml:1.0:function:any-of-any">
113     <Function FunctionId=
114       "urn:oasis:names:tc:xacml:1.0:function:regexp-string-match"/>
115       <Apply
116         FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-bag">
117         <AttributeValue
118           DataType="http://www.w3.org/2001/XMLSchema#string">
119             @informatik.uni-muenchen.de
120         </AttributeValue>
121         <AttributeValue
122           DataType="http://www.w3.org/2001/XMLSchema#string">
123             @ifi.lmu.de
124         </AttributeValue>
125       </Apply>
126       <AttributeSelector RequestContextPath=
127         "//context:ResourceContent/condition:e-mail-adress/text()"
128         DataType="http://www.w3.org/2001/XMLSchema#string"/>
129     </Condition>
130   </Rule>
131
132   <!-- ... weitere Regeln ... -->
133
134   <Rule RuleId="sonstNichtsReleasen" Effect="Deny"/>
135
136   <Obligations>
137     <Obligation ObligationId=
138       "data-has-to-be-deleted-after-end-of-course"
139       FulfillOn="Permit">
140       <AttributeAssignment AttributeId="resource"
141         DataType="http://www.w3.org/2001/XMLSchema#anyURI">
142           urn:oasis:names:tc:xacml:1.0:resource:resource-id
143       </AttributeAssignment>
144     </Obligation>
145   </Obligations>
146
147 </Policy>
```

6.2 In Shib bereits vorhandene ARP Architektur

Um eine neue policy-basierte Privacy Management Architektur für Shibboleth entwickeln zu können, muss zunächst der Workflow der bisherigen Shibboleth-ARP-Architektur untersucht werden.

6.2.1 Architektur/Workflow im Überblick

Erhält der Identity Provider eine Attributanfrage, erstellt die AA (Attribute Authority) eine effective ARP, welche die Menge aller ARP Regeln aus allen ARP Containern (Site ARP, User ARP) für einen bestimmten Benutzer darstellt. Diese effective ARP wird dann auf die Attributwerte des Verzeichnisdienstes angewandt, woraus dann eine Antwort (Assertion) erstellt wird.

Der ARP Workflow besteht aus folgenden Schritten:

- Identifiziere alle ARPs (Site ARPs, User ARPs), die auf einen Benutzer angewandt werden sollen.
- Finde alle für die Anfrage relevanten Regeln:
  - Alle standard (default) Regeln aus den identifizierten ARPs werden in die effective ARP übernommen
  - Weiterhin werden die manuellen (non default) Regeln übernommen, welche im Targetbereich TRUE liefern.
- Erstellung des Attribut Filters:
  - Für jedes Attribut wird eine Liste mit allen Attributwerten erstellt, welche in den entsprechenden Regeln den Herausgabewert `PERMIT` haben
  - Davon werden alle Attributwerte abgezogen, welche in den Regeln den Herausgabewert `DENY` haben. `DENY` überschreibt also `PERMIT`. Die daraus resultierenden Attributwerte stellen alle möglichen freizugebenden Werte dar und dienen für die vom Attribute Resolver gelieferten Werte als Maske.
- Mit Hilfe der Maske und den vom Attribute Resolver gelieferten Attribute wird eine Antwort erstellt

[ARP Shib]

6.2.2 Workflow in Java

In diesem Abschnitt wird der Workflow der Shibboleth-ARP-Architektur unter Berücksichtigung der wichtigsten verwendeten Java Klassen und Methoden dargestellt (siehe dazu auch Abbildung 6.3 (Parameter aus Gründen der Übersicht mit `x:x` dargestellt)). Es wird von der Shibboleth Standardkonfiguration ausgegangen, wobei die Benutzerinformationen mit Hilfe von LDAP gespeichert und angefragt werden.

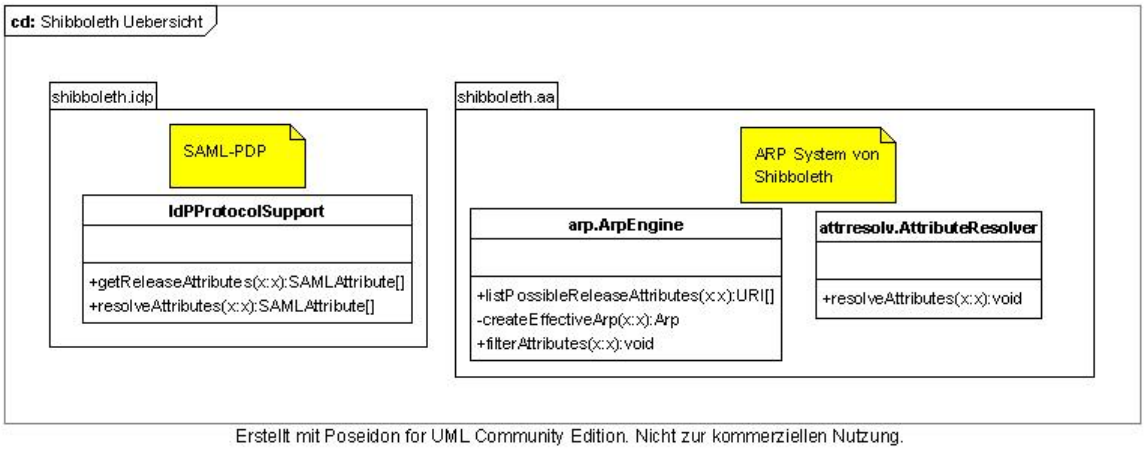

Abbildung 6.3: Klassendiagramm Shibboleth

Die Ausgangsmethode ist `public void doGet(HttpServletRequest request, HttpServletResponse response)` und befindet sich in der Klasse edu.internet2.middleware.shibboleth.idp.IdPResponder. Dort wird die Methode `public SAMLResponse processRequest(HttpServletRequest request, HttpServletResponse response, SAMLRequest samlRequest, IdPProtocolSupport support)` der Klasse edu.internet2.middleware.shibboleth.idp.provider.SAMLv1_AttributeQueryHandler aufgerufen, dessen Ziel es ist, eine SAML-Antwort aus den freizugebenden Attributen zu generieren. Diese Methode ruft eine Reihe weiterer Methoden auf, welche zur Erstellung einer SAML-Antwort nötig sind, von denen die wichtigsten in folgender Liste hierarchisch dargestellt werden:

- Klasse: edu.internet2.middleware.shibboleth.idp.IdPProtocolSupport
- Methode: public SAMLAttribute[] getReleaseAttributes(Principal principal, RelyingParty relyingParty, String requester, URL resource)
- Beschreibung: Beschaffung aller freizugebenden Attribute

  - Klasse: edu.internet2.middleware.shibboleth.aa.arp.ArpEngine
  - Methode: public URI[] listPossibleReleaseAttributes(Principal principal, String requester, URL resource)
  - Beschreibung: listet alle möglichen freizugebenden Attribute und deren Attributwerte mit Hilfe der effective ARP auf

    * Klasse: edu.internet2.middleware.shibboleth.aa.arp.ArpEngine
    * Methode: private Arp createEffectiveArp(Principal principal, String requester, URL resource)
    * Beschreibung: Erstellung einer effective ARP

  - Klasse: edu.internet2.middleware.shibboleth.idp.IdPProtocolSupport
  - Methode: SAMLAttribute[] resolveAttributes(Principal principal, String requester, String responder, URL resource, AAAttributeSet attributeSet)

- Beschreibung: Parameter beinhaltet alle möglichen freizugebenden Attribute. Ziel ist die Auflösung aller möglichen freizugebenden Attribute und das Filtern dieser Attribute

  * Klasse: edu.internet2.middleware.shibboleth.aa.attrresolv.AttributeResolver
  * Methode: public void resolveAttributes(Principal principal, String requester, String responder, ResolverAttributeSet attributes)
  * Beschreibung: löst die Attribute auf, d.h. sucht die Attribute in LDAP und liefert deren Werte zurück

  * Klasse: edu.internet2.middleware.shibboleth.aa.arp.ArpEngine
  * Methode: public void filterAttributes(ArpAttributeSet attributes, Principal principal, String requester, URL resource)
  * Beschreibung: filtert die Attribute unter Verwendung der effective ARP

    · Klasse: edu.internet2.middleware.shibboleth.aa.arp.ArpEngine

    · Methode: private Arp createEffectiveArp(Principal principal, String requester, URL resource)

    · Beschreibung: erstellt effective ARP

6.3 Neu entwickelte ARP Architektur

Dieser Abschnitt führt die neu entwickelte XACML-basierte Architektur für Shibboleth ein. Neben den neuen Komponenten wird auch der Java-Workflow beschrieben.

6.3.1 Architektur/Workflow im Überblick

Die neu entwickelte Architektur besteht aus einem XACML-PEP, einem XACML-PDP und einem neuen PAP. Shibboleth bietet mehrere Möglichkeiten, einen PIP zu realisieren, von denen wir in unserer Architektur die openLDAP Variante wählen. Die Abkürzungen PEP, PDP, PAP und PIP können in Abschnitt 6.1.1 nachgeschlagen werden. Da Shibboleth auf SAML basiert, erhält der Shibboleth Identity Provider Attributanfragen in Form von SAML. Die Systemeinheit, die diese Anfragen empfängt wird SAML-PDP genannt. Der SAML-PDP leitet die Anfragen weiter an den XACML-PEP, dessen Aufgabe darin besteht, die Anfragen von SAML in eine entsprechende XACML Kontext Anfrage zu konvertieren, diese an den (bereits vorimplementierten) XACML-PDP weiterzuleiten, XACML Kontext Antworten vom XACML-PDP entgegenzunehmen, nach SAML zurück zu konvertieren und wieder an den SAML-PDP weiter zu reichen. Eine weitere Aufgabe des XACML-PEP ist das Ermitteln und Auflösen (mit Hilfe des PIP) aller relevanten Attribute. Aufgaben des XACML-PDP sind das Auffinden der zu einer Anfrage passenden Attribute Release Policies im PAP, die Auswertung dieser Policies und das Treffen einer Autorisierungsentscheidung. Der PAP ist durch openLDAP realisiert. Dort werden Informationen, welche zur Autorisierung nötig sind (Policies), in einem geeigneten openLDAP Verzeichnis abgelegt und Shibboleth so zur Verfügung gestellt. Der PAP (Policies) und der PIP (Benutzerinformationen) können örtlich getrennt, also auf

verschiedenen openLDAP-Servern, aber auch mit nur einem openLDAP Server realisiert werden. Des Weiteren kann der PAP über mehrere openLDAP Server verteilt werden, d.h. Policies können dezentral, auf verschiedenen Servern, abgelegt werden. Damit ist ein Grundstein für die delegierte Policy-Administration gelegt, also dass Policies an mehreren Stellen von unterschiedlichen Instanzen erstellt und bearbeitet werden können.

Abbildung 6.4 stellt die Architektur, welche sich ausschließlich in der Identity-Provider Komponente von Shibboleth befindet, im grafischem Überblick dar.

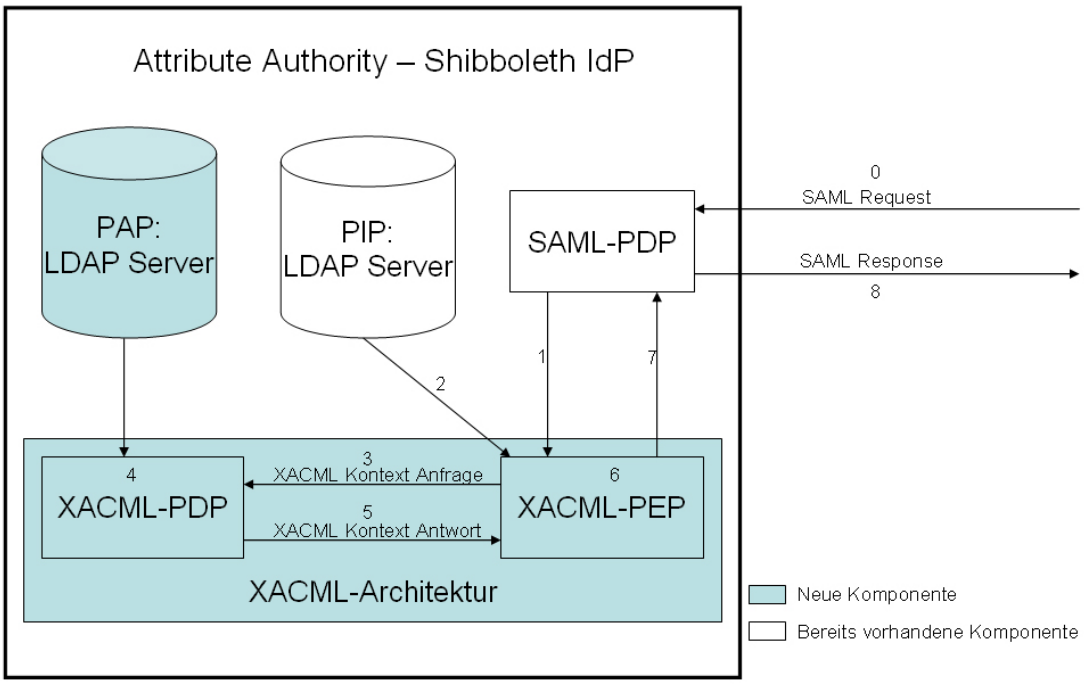

Abbildung 6.4: XACML Architektur

Der ARP Workflow besteht aus folgenden Schritten:

1. Der SAML-PDP gibt Metadaten (z.B. Service Provider und Benutzer, von dem Informationen erfragt werden sollen) der SAML Anfrage weiter an den XACML-PEP. Es wird davon ausgegangen, dass in der SAML Anfrage keine speziellen Attribute angefragt werden, sondern alle möglichen Attribute.
2. Der XACML-PEP ermittelt alle möglichen freizugebenden Attribute und Attributwerte:
   - Finde alle zur Anfrage passenden ARPs aus den dafür vorgesehenen LDAP Servern unter zu Hilfenahme des Benutzernamens (von dem Informationen erfragt werden sollen) und der Benutzerrolle (welche ein Benutzer für die jeweils gefundene Policy haben muss).
   - Identifiziere mit Hilfe der zur Anfrage passenden ARPs alle möglichen freizugebenden Attribute (Resources).
   - Löse diese Attribute mit Hilfe des definierten AttributeResolvers auf.

3. Pro möglichen Attribut sendet der XACML-PEP eine XACML Kontext Anfrage an den XACML-PDP. Es können nicht alle Attribute auf einmal in einer XACML Kontext Anfrage gesendet und ausgewertet werden, da sonst die Auswertung nicht auf Attributebene sondern auf Policyebene stattfinden würde. Die Freigabeverweigerung eines Attributs würde die Freigabeverweigerung aller Attribute nach sich ziehen. Da jedoch für jedes Attribut einzeln entschieden werden soll, muss auch jedes Attribut einzeln an den XACML-PDP gesendet werden. In jeder XACML Kontext Anfrage des XACML-PEP werden alle für den XACML-PDP benötigten Informationen mitgeliefert, d.h. neben den angefragten Attributen und Attributwerten werden auch solche Attribute im `ResourceContent` mitgeliefert, welche zur Auswertung von Bedingungen in ARPs nötig sind.

4. Jede XACML Kontext Anfrage wird vom XACML-PDP ausgewertet:

   - Finde erneut alle zur Anfrage passenden ARPs, die neben dem Benutzernamen und der Benutzerrolle auch im Target-Bereich mit der Anfrage übereinstimmen.
   - Kombiniere die gefundenen ARPs zu einer Policy Menge.
   - Werte die Policy Menge bezüglich der vom XACML-PEP gesendeten Anfrage und entsprechenden Kombinierungsalgorithmen aus.
   - Erstelle eine XACML Antwort.

5. Das Ergebnis (XACML Kontext Antwort), bestehend aus Entscheidung und Obligationen, wird an den XACML-PEP zurückgeliefert.

6. Sind in der XACML Kontext Antwort Obligationen enthalten, so müssen diese vom XACML-PEP erfüllt werden, bevor die freizugebenden Attribute weiter verarbeitet werden. Die Erfüllung von Obligationen wird in dieser Arbeit nicht umgesetzt. Dazu muss eine neue Teilarchitektur erschaffen werden, welche mögliche Obligationen definiert und beschreibt, wie diese umgesetzt werden müssen.

7. der XACML-PEP wandelt die freizugebenden Attribute in eine SAML Antwort um und sendet diese an den SAML-PDP.

8. der SAML-PDP liefert die SAML Antwort, also die freizugebenden Attribute, weiter an den anfragenden Service Provider. Folgendes Beispiel zeigt eine mögliche SAML Antwort, in der das Attribut `eduPersonAffiliation` (Zeilen 47 - 54) mit dem Wert `student` (Zeilen 50 - 53) an den Service Provider `https://lxssp02.lrz-muenchen.de` (Zeile 22) herausgegeben wird:

```
1  <Response
2  xmlns='urn:oasis:names:tc:SAML:1.0:protocol'
3  xmlns:saml='urn:oasis:names:tc:SAML:1.0:assertion'
4  xmlns:samlp='urn:oasis:names:tc:SAML:1.0:protocol'
5  xmlns:xsd='http://www.w3.org/2001/XMLSchema'
6  xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
7  InResponseTo='9da23503a06d7bafc3849e607e895e5f'
8  IssueInstant='2006-11-02T07:37:53.517Z'
9  MajorVersion='1'
10 MinorVersion='1'
11 ResponseID='584ab607c026062f09c26d9dfab51590'>
12
13   <Status>
14     <StatusCode Value='samlp:Success'>
15     </StatusCode>
```

```
16    </Status>
17
18    <Assertion
19     xmlns='urn:oasis:names:tc:SAML:1.0:assertion'
20     AssertionID='e3662dde86569351efd70abacf1ba72a'
21     IssueInstant='2006-11-02T07:37:53.517Z'
22     Issuer='https://lxsidp02.lrz-muenchen.de'
23     MajorVersion='1'
24     MinorVersion='1'>
25
26       <Conditions
27        NotBefore='2006-11-02T07:37:53.517Z'
28        NotOnOrAfter='2006-11-02T08:07:53.517Z'>
29          <AudienceRestrictionCondition>
30            <Audience>
31               https://lxssp02.lrz-muenchen.de
32            </Audience>
33            <Audience>
34               urn:mace:inqueue
35            </Audience>
36          </AudienceRestrictionCondition>
37       </Conditions>
38
39       <AttributeStatement>
40          <Subject>
41            <NameIdentifier
42             Format='urn:mace:shibboleth:1.0:nameIdentifier'
43             NameQualifier='https://lxsidp02.lrz-muenchen.de'>
44               02d4dfa2e1b6264ce0885d441718d1fc
45            </NameIdentifier>
46          </Subject>
47          <Attribute
48           xmlns:typens='urn:mace:shibboleth:1.0'
49           AttributeName='urn:mace:dir:attribute-def:eduPersonAffiliation'>
50            <AttributeValue
51             xsi:type='typens:AttributeValueType'>
52               student
53            </AttributeValue>
54          </Attribute>
55       </AttributeStatement>
56
57    </Assertion>
58
59 </Response>
```

6.3.2 Zugrunde liegendes Informationsmodell des PAP

Alle Informationen, welche sowohl der XACML-PEP, als auch der XACML-PDP benötigen, werden über openLDAP Server zur Verfügung gestellt. Ziel des neuen Informationsmodells ist es, die für einen Benutzer relevanten Policies schnell und effizient aufzufinden. Dazu muss für jede Policy angegeben werden können, für welche(n) Benutzer diese erstellt wurde und gilt. Ein weiteres Ziel ist es, dass Policies nicht mehr an einer zentralen Stelle abgelegt werden müssen, sondern dezentral gespeichert und administriert werden können. Zur Ermittlung der zu einer Anfrage passenden Policymenge können also verschiedene LDAP Server angefragt werden.

Zur Speicherung der Policies wird folgendes Informationsmodell zu Grunde gelegt (die dazugehörigen LDAP-Schemata werden in Abschnitt 7.1 angegeben):

- Policies werden in einer LDAP Baumstruktur abgelegt
- Nach einer geeigneten LDAP Wurzel teilt sich der Baum in `ou=groups` und `ou=policies`
  - Unter `ou=groups` werden flach Gruppen von Benutzern gespeichert, bestehend aus den Attributen `XACMLgroupName` (definiert den Namen der Gruppe) und `XACMLmembers` (mehrwertiges Attribut zur Speicherung der Gruppenmitglieder)
  - Diese Gruppen können in LDAP Policy Einträgen referenziert werden, wodurch festgelegt wird, für welche Benutzer bzw. Benutzergruppen eine Policy gilt. Dadurch können die für einen Benutzer relevanten Policies effizient vorgefiltert werden, ohne die Policy selbst untersuchen zu müssen.
  - Der Knoten `ou=policies` besteht aus den zwei Kindknoten `ou=sitearps` und `ou=userarps` zur übersichtlicheren Unterscheidung von Policies, welche für bestimmte Benutzer angelegt wurden und Policies, welche für ganze Benutzergruppen angelegt wurden
  - Sowohl unter `ou=sitearps` als auch unter `ou=userarps` werden flach Policies gespeichert (pro Eintrag eine Policy), bestehend aus den Attributen:
    - `XACMLpolicyId` (eindeutiger Name einer Policy, zwingende Information)
    - `XACMLpolicy` (Speicherung der Policy im String-Format, zwingende Information)
    - `XACMLusers` (hier werden einer oder mehrere Benutzer gespeichert, für die eine Policy gilt. Alternative oder Ergänzung zur Angabe einer Gruppe)
    - `XACMLgroupNames` (Referenzierung einer Gruppe, für die eine Policy gilt. Es muss entweder eine Gruppe, ein Benutzer oder beides angegeben werden)
    - `XACMLtargets` (Speicherung der Targets (Service Provider), für die eine Policy gilt. Optionale Information für eine bessere Übersicht, da das Target in der Policy selbst gespeichert wird)
    - `XACMLroles` (Speicherung der Rollen, für die eine Policy gilt. Dadurch wird das Rollenkonzept umgesetzt. Die Rollen werden also nicht in der Policy selbst angegeben. Dies hat u.a. den Vorteil, dass durch eine externe LDAP-Rollenangabe passende Policies schneller und effizienter aufgesucht werden können. Zwingende Information)

– Das Schema für die `siteaprs` und für die `userarps` ist das Selbe. Theoretisch könnte also auch ein Benutzer Gruppen für seine ARPs definieren. Will man, dass ein Benutzer für eine ARP z.B. keine Gruppe festlegen darf, und dass im Attribut `XACMLusers` nur der eigene Benutzername gespeichert werden darf, so muss dies durch eine geeignete Benutzerschnittstelle realisiert werden, welche nicht Teil dieser Arbeit ist.

Abbildung 6.5 zeigt die eben beschriebenen Struktur im Überblick, wobei in der Abbildung auch ein Vorschlag gegeben wird, wie der PIP, also die Quelle für Benutzerinformationen (z.B. bestehend aus den Attributen `uid` und `password`), mit integriert werden kann:

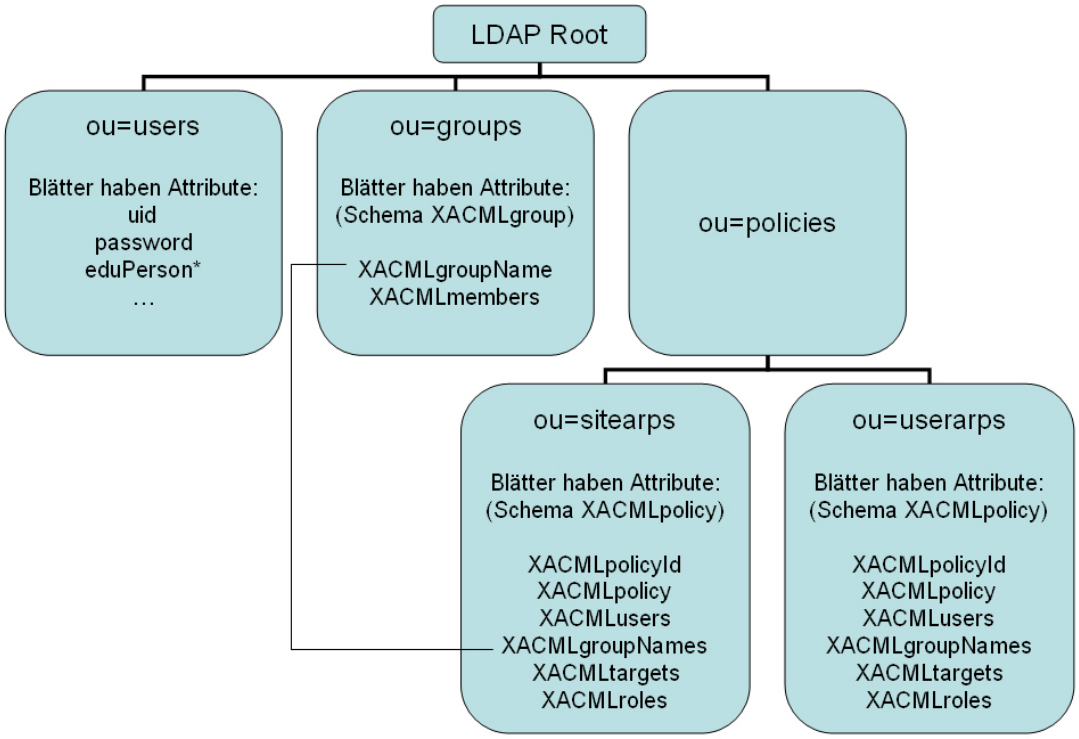

Abbildung 6.5: PAP Struktur

6.3.3 PEP- und PDP-Workflow in Java

Die neuen bzw. abgeänderten Komponenten von Shibboleth 1.3c und des Sun-XACML-PDP 1.2 sind auf folgende Pakete aufgeteilt:

- Der XACML-PDP befindet sich im Java Paket:
  edu.internet2.middleware.shibboleth.xacml.com.sun.xacml.*
- Der XACML-PEP befindet sich im Java Paket:
  edu.internet2.middleware.shibboleth.aa.*

Abbildung 6.6 zeigt die neue Architektur mit den wesentlichen Java-Klassen und -Methoden. Da die Parameter der Methoden das Diagramm sprengen würden, werden diese lediglich durch `x:x` dargestellt.

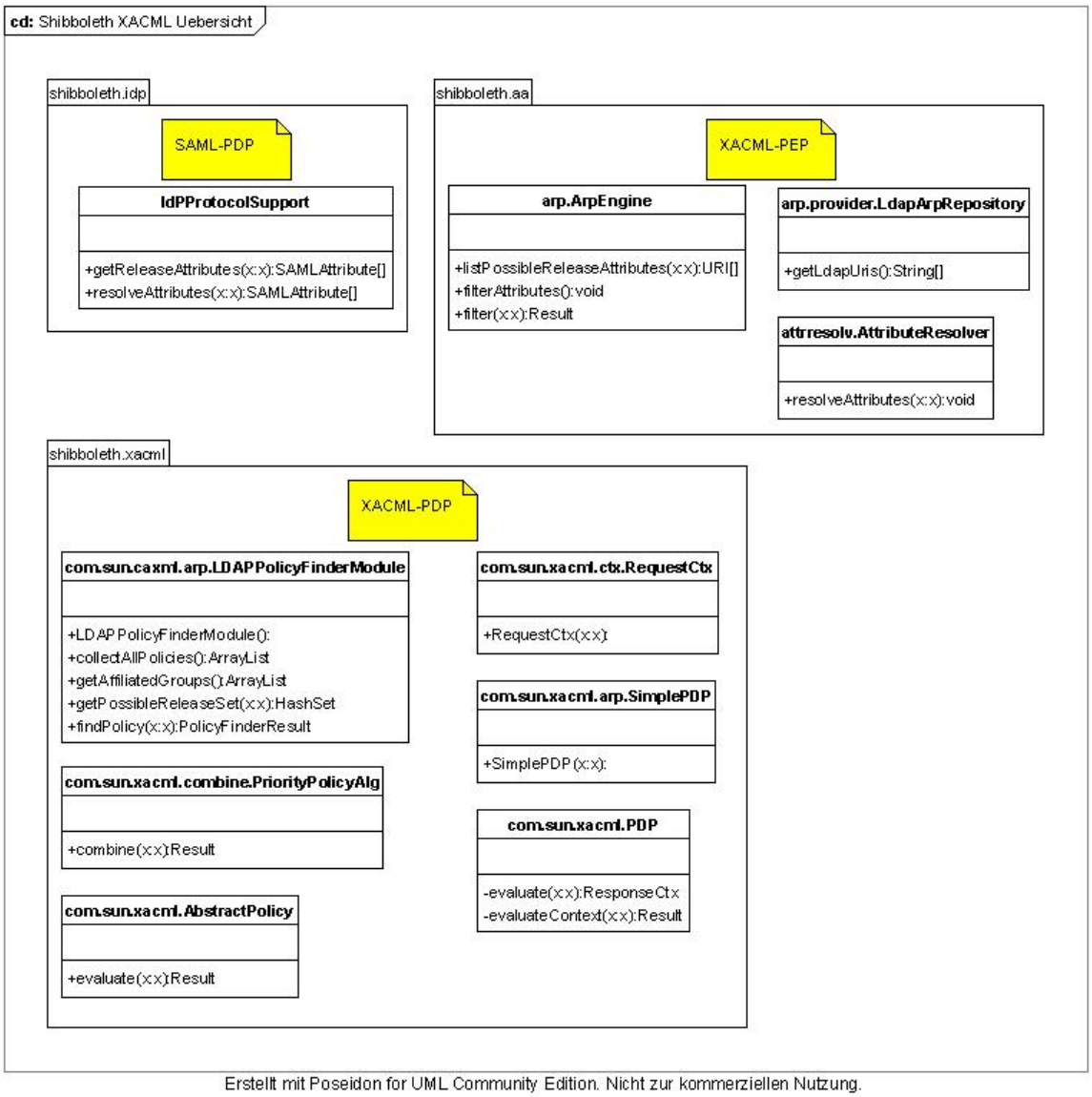

Abbildung 6.6: Klassendiagramm Shibboleth XACML

Abbildung 6.7 gibt zusätzlich einen hierarchischen und geordneten Überblick (übergeordnete Methoden rufen die untergeordneten, weiter eingerückten Methoden auf) über die verwendeten Java Methoden. Methoden, welche für die neue Architektur nicht wesentlich geändert werden mussten oder den Workflow nicht direkt beeinflussen, werden nicht betrachtet, da die Beschreibung jeder Java-Methode den Rahmen dieser Arbeit sprengen würde.

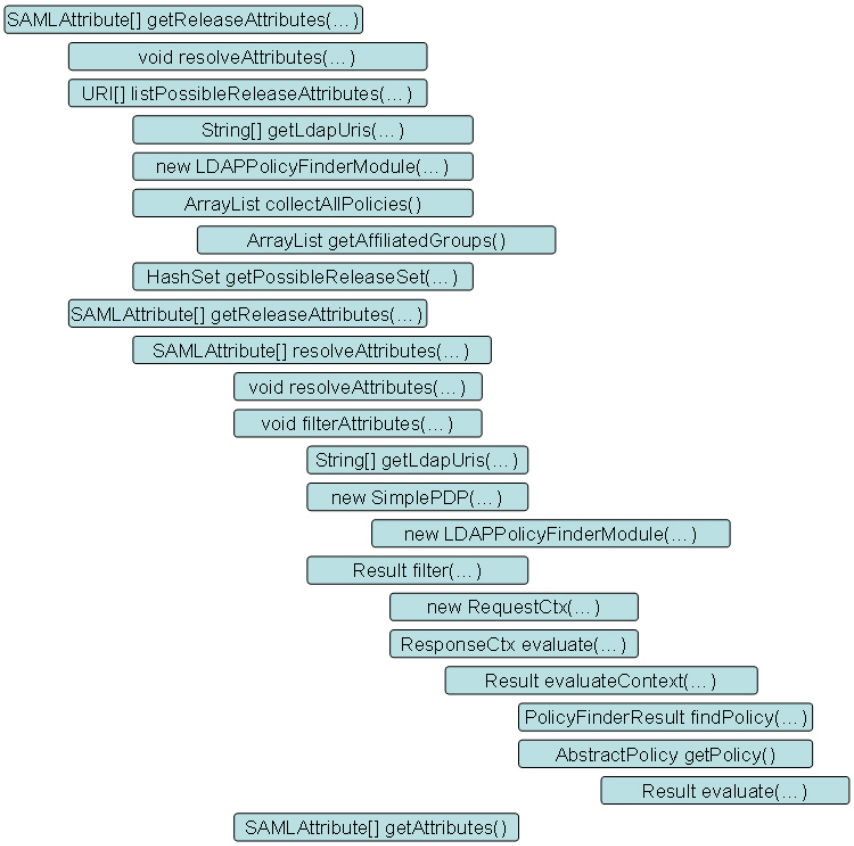

Abbildung 6.7: Java XACML Workflow

Der Workflow der neuen Architektur wird ab der Methode `public SAMLAttribute[] getReleaseAttributes(...)` im Paket edu.internet2.middleware.shibboleth.idp betrachtet, da hier das policy-basierte Privacy Management System angestoßen wird. Im Folgenden werden Klassen, in denen wesentliche Änderungen vorgenommen oder neu entwickelt wurden, mit Referenzen auf ihre Implementierung versehen, im Gegensatz zu Methoden, welche ohne größere Änderungen in den Workflow übernommen wurden.

public SAMLAttribute[] getReleaseAttributes(...)

- Klasse: edu.internet2.middleware.shibboleth.idp.IdPProtocolSupport.java (siehe 7.2.1)
- Methode: public SAMLAttribute[] getReleaseAttributes(...)
- Beschreibung: Beschaffung aller freizugebenden Attribute

Ziel der Methode public SAMLAttribute[] getReleaseAttributes(Principal principal, RelyingParty relyingParty, String requester, URL resource) ist die Beschaffung aller freizugebenden Attribute. Zu

Beginn dieser Methode wird die Rolle des Benutzers mittels einer LDAP-Anfrage aufgelöst. Danach werden mit Hilfe der Methode public URI[] listPossibleReleaseAttributes(Principal principal, String requester, URL resource, String roleName) alle möglichen freizugebenden Attribute ermittelt. Durch den folgenden Aufruf von SAMLAttribute[] getReleaseAttributes(Principal principal, RelyingParty relyingParty, String requester, URL resource, URI[] attributeNames) werden die Attributwerte ermittelt und anschließend mit Hilfe der ARPs gefiltert.

public void resolveAttributes(...)

- Klasse: edu.internet2.middleware.shibboleth.aa.attrresolv.AttributeResolver.java
- Methode: public void resolveAttributes(...)
- Beschreibung: Auflösen von Attributen

Die Methode public void resolveAttributes(Principal principal, String requester, String responder, ResolverAttributeSet attributes) wird hier verwendet, um die Rolle des Benutzers mittels LDAP-Zugriff abzufragen. Die Rolle des Benutzers wird benötigt, um entsprechende Policies zu finden, welche für die Rolle erstellt wurden.

public URI[] listPossibleReleaseAttributes(...)

- Klasse: edu.internet2.middleware.shibboleth.aa.arp.ArpEngine.java (siehe 7.2.2)
- Methode: public URI[] listPossibleReleaseAttributes(...)
- Beschreibung: listet alle möglichen freizugebenden Attribute auf, indem alle für einen Benutzer relevanten Policies auf Attribute untersucht werden

Die Methode public URI[] listPossibleReleaseAttributes(Principal principal, String requester, URL resource, String roleName) ermittelt alle möglichen freizugebenden Attribute, indem alle für einen Benutzer relevanten Policies gesammelt und daraus die möglichen Attribute extrahiert werden. Um alle relevanten Policies zu erhalten, werden zunächst mit der Methode public String[] getLdapUris() die LDAP-Verzeichnisse angefragt, in denen Policies abgelegt wurden. Danach wird ein neues Modul LDAPPolicyFinderModule(String [] LDAPuris, Principal principal, Logger log, String roleName) initialisiert, welches dafür zuständig ist, Policies mit Hilfe der Uris, des Benutzers und der Benutzerrolle ausfindig zu machen. Dies ist Aufgabe der Methode public ArrayList collectAllPolicies(). Nachdem nun alle relevanten Policies ermittelt wurden, findet die Methode public HashSet getPossibleReleaseSet(ArrayList pol) alle möglichen freizugebenden Attribute.

public String[] getLdapUris()

- Klasse: edu.internet2.middleware.shibboleth.aa.arp.provider.LdapArpRepository.java (siehe 7.2.3)
- Methode: public String[] getLdapUris()
- Beschreibung: liefert LDAP-Uris, unter denen Policies abgelegt sind

Die Methode public String[] getLdapUris() ist eine Methode des neu entwickelten LdapArpRepositories. Aufgabe des Repositories ist das Auslesen und Zurückgeben der LDAP-Uris, welche in der idp.xml konfiguriert wurden.

public LDAPPolicyFinderModule()

- Klasse: edu.internet2.middleware.shibboleth.xacml.com.sun.xacml.arp.LDAPPolicyFinderModule.java (siehe 7.2.4)
- Konstruktor: public LDAPPolicyFinderModule()
- Beschreibung: Modul, um Policies zu finden und daraus ein PolicySet zu erstellen

Der Konstruktor public LDAPPolicyFinderModule(String [] LDAPuris, Principal principal, Logger log, String roleName) erstellt eine neue Instanz, mit der Policies im Bezug auf LDAP-Uri, Benutzer und Rolle vorgefiltert und mit Hilfe eines Requests die relevanten Policies gefunden werden können. Die relevanten Policies können weiterhin zu einem PolicySet zusammengefasst werden. In dieser Klasse wird auch der für eine Policy Menge verwendete Kombinierungsalgorithmus angegeben.

public ArrayList collectAllPolicies()

- Klasse: edu.internet2.middleware.shibboleth.xacml.com.sun.xacml.arp.LDAPPolicyFinderModule.java (siehe 7.2.4)
- Methode: public ArrayList collectAllPolicies()
- Beschreibung: findet alle Policies im Bezug auf Benutzer und dessen Rolle

Die Methode public ArrayList collectAllPolicies() findet mit Hilfe der Benutzerrolle und des Benutzernamens alle für einen Benutzer relevanten Policies. Somit erhält man eine Vorauswahl aller relevanten Policies. In dieser Methode werden die Policies noch nicht anhand des zu einem Request passenden Targets gefiltert. D.h. es wird eine Obermenge aller relevanten Policies zurückgegeben. Ob eine Policy für einen Benutzer im Bezug auf dessen Benutzernamen relevant ist, wird durch die LDAP-Attribute XACMLusers und XACMLgroupNames bestimmt. Unter den angegebenen LDAP-Uris werden sowohl die User-Policies als auch die Site-Policies auf Relevanz untersucht.

private ArrayList getAffiliatedGroups()

- Klasse: edu.internet2.middleware.shibboleth.xacml.com.sun.xacml.arp.LDAPPolicyFinderModule.java (siehe 7.2.4)
- Methode: private ArrayList getAffiliatedGroups()
- Beschreibung: liefert alle Gruppennamen zurück, zu denen ein Benutzer gehört

Die Methode private ArrayList getAffiliatedGroups() ist notwendig, um alle Gruppen, denen ein Benutzer angehört zu bestimmen. Dies ist notwendig, um relevante Policies für einen Benutzer anhand einer angegebenen Gruppe bestimmen zu können.

public HashSet getPossibleReleaseSet(ArrayList pol)

- Klasse: edu.internet2.middleware.shibboleth.xacml.com.sun.xacml.arp.LDAPPolicyFinderModule.java (siehe 7.2.4)
- Methode: public HashSet getPossibleReleaseSet(ArrayList pol)
- Beschreibung: filtert Attribute aus allen gefundenen relevanten Policies eines Benutzers

Die Methode public HashSet getPossibleReleaseSet(ArrayList pol) findet zu allen gefundenen Policies die Resourcen, also die Attribute, um die es geht. Somit werden alle Attribute gesammelt, welche im weiteren Verlauf aufgelöst und mit denen Anfragen an den PDP gestellt werden.

public SAMLAttribute[] getReleaseAttributes(...)

- Klasse: edu.internet2.middleware.shibboleth.idp.IdPProtocolSupport.java (siehe 7.2.1)
- Methode: public SAMLAttribute[] getReleaseAttributes(...)
- Beschreibung: Beschaffung aller freizugebenden Attribute

Die Methode public SAMLAttribute[] getReleaseAttributes(Principal principal, RelyingParty relyingParty, String requester, URL resource, URI[] attributeNames) wird von der gleichnamigen Methode aufgerufen, nachdem alle relevanten Attribute für einen Benutzer gesammelt wurden. Ziel dieser Methode ist das Selbe wie bereits oben beschriebene gleichnamige Methode, mit dem Unterschied, dass nun schon die relevanten Attribute bekannt sind. Nachdem alle Attribute in einem AttributeSet zusammengefasst wurden, wird die Methode public SAMLAttribute[] resolveAttributes(Principal principal, String requester, String responder, URL resource, AAAttributeSet attributeSet) aufgerufen, welche wiederum die Methode public void resolveAttributes(Principal principal, String requester, String responder, ResolverAttributeSet attributes) ausführt, um das AttributSet des Benutzers aufzulösen. Wurden alle Attributwerte bestimmt, so werden mit Hilfe der Methode public void filterAttributes(ArpAttributeSet attributes, Principal principal, String requester, URL resource, String roleName) alle freizugebenden Attribute bestimmt. Dies geschieht durch die Erzeugung einer Instanz des PDPs, der erneuten Erstellung einer relevanten Policy Menge, der Generierung einer Anfrage pro Attribut und der Auswertung dieser Anfrage bezüglich der Policy Menge.

public SAMLAttribute[] resolveAttributes(...)

- Klasse: edu.internet2.middleware.shibboleth.idp.IdPProtocolSupport.java (siehe 7.2.1)
- Methode: public SAMLAttribute[] resolveAttributes(...)
- Beschreibung: löst Attribute auf und ruft Methoden auf, welche diese filtern

Die Methode public SAMLAttribute[] resolveAttributes(Principal principal, String requester, String responder, URL resource, AAAttributeSet attributeSet) ruft zuerst die Methode public void resolveAttributes(Principal principal, String requester, String responder, ResolverAttributeSet attributes) auf, welche alle relevanten Attribute auflöst. Danach wird die Methode public void filterAttributes(ArpAttributeSet attributes, Principal principal, String requester, URL resource, String roleName) aufgerufen, welche die Attribute filtert.

public void resolveAttributes(...)

- Klasse: edu.internet2.middleware.shibboleth.aa.attrresolv.AttributeResolver.java
- Methode: public void resolveAttributes(...)
- Beschreibung: löst Attribute auf

Die Methode public void resolveAttributes(Principal principal, String requester, String responder, ResolverAttributeSet attributes) sucht in LDAP alle Attribute des AttributeSets und löst diese auf.

public void filterAttributes(...)

- Klasse: edu.internet2.middleware.shibboleth.aa.arp.ArpEngine.java (siehe 7.2.2)
- Methode: public void filterAttributes(...)
- Beschreibung: wendet alle relevanten ARPs auf die Attribute an

Die Methode public void filterAttributes(ArpAttributeSet attributes, Principal principal, String requester, URL resource, String roleName, ArpAttributeSet conditionAttributes) bestimmt zunächst erneut die zu verwendenden LDAP-Verzeichnisse, in denen die Policies liegen und initialisiert mit dieser Information, sowie mit dem Benutzernamen und dessen Rolle, eine neue PDP Instanz, welche auch eine neue Instanz eines LDAPPolicyFinderModule beinhaltet. Für jedes Attribut wird nun die Methode public Result filter(String subject, String resource, String action, SimplePDP PDP, ArpAttributeSet conditionAttributes) aufgerufen, welche über die Freigabe jedes Attributs entscheidet. Pro Attribut wird eine neue Anfrage erstellt, und diese dem PDP in der Methode public ResponseCtx evaluate(RequestCtx request) übergeben. Der PDP erstellt aus einer Anfrage einen Kontext, findet für jeden Kontext die relevante Policy Menge und wertet den Kontext gegen die erstellte Policy Menge mit Hilfe der Methode private Result evaluateContext(EvaluationCtx context) aus.

public SimplePDP(...)

- Klasse: edu.internet2.middleware.shibboleth.xacml.com.sun.xacml.arp.SimplePDP.java (siehe 7.2.5)
- Konstruktor: public SimplePDP(...)
- Beschreibung: Initialisierung eines neuen PDP

Der Konstruktor public SimplePDP(String [] LDAPuris, Principal principal, Logger log, String roleName) erstellt eine neue PDP Instanz. Es werden Module zum Auffinden von Policies und zum Auffinden von Attributen (wie Systemzeit) initialisiert.

public Result filter(...)

- Klasse: edu.internet2.middleware.shibboleth.aa.arp.ArpEngine.java (siehe 7.2.2)
- Methode: public Result filter(...)

- Beschreibung: erzeugt pro Attribut eine Anfrage und liefert eine Antwort zurück

Die Methode public Result filter(String subject, String resource, String action, SimplePDP PDP, ArpAttributeSet conditionAttributes) erzeugt pro Attribut einen Request mit Hilfe der Konstruktoren der Klasse edu.internet2.middleware.shibboleth.xacml.com.sun.xacml.ctx.RequestCtx.java. Anschließend wird dem PDP die erstellte Anfrage zur Auswertung durch die Methode public ResponseCtx evaluate(RequestCtx request) übergeben und ein entsprechendes Ergebnis zurückgeliefert.

public RequestCtx(...)

- Klasse: edu.internet2.middleware.shibboleth.xacml.com.sun.xacml.ctx.RequestCtx.java
- Konstruktor: public RequestCtx(...)
- Beschreibung: Initialisierung von Anfrageinstanzen

Zur Generierung einer Anfrageinstanz stehen mehrere Konstruktoren zur Verfügung. Will man eine Anfrage ohne ResourceContent erstellen, so wird der Konstruktor public RequestCtx(Set subjects, Set resource, Set action, Set environment) verwendet. Mit ResourceContent wird der Konstruktor public RequestCtx(Set subjects, Set resource, Set action, Set environment, String resourceContent) verwendet. Anfragen können also mit dem Benutzer, der Aktion, der Resource (Attribut) dem Content und ggf. mit UmgebungsAttributen initialisiert werden. Um eine Anfrage mit Werten zu initialisieren gibt es sog. Setter-Methoden, wie z.B.: public Set setupSubjects(String sub), public Set setupResource(String res), public Set setupAction() und public String setResourceContent(ArpAttributeSet conditionAttributes).

public ResponseCtx evaluate(...)

- Klasse: edu.internet2.middleware.shibboleth.xacml.com.sun.xacml.PDP.java
- Methode: public ResponseCtx evaluate(...)
- Beschreibung: Auswertung einer Anfrage

Die Methode public ResponseCtx evaluate(RequestCtx request) erzeugt zunächst aus einer Anfrage einen für den PDP verständlichen Kontext. Danach wird die gleichnamige Methode public ResponseCtx evaluate(EvaluationCtx context) aufgerufen, welche bezüglich des erstellten Kontexts eine Entscheidung zurückliefert. Zur Entscheidungsfindung wird sich der Methode private Result evaluateContext(EvaluationCtx context) bedient, welche mit Hilfe des Moduls LDAPPolicyFinderModule alle zur Anfrage passenden Policies findet und daraus ggf. eine Policy Menge erstellt. Auf die so gefundene Policy bzw. Policy Menge wird nun der Kontext mit durch die Methode public Result evaluate(EvaluationCtx context) angewandt, welche schließlich ein Ergebnis unter Verwendung von Konbinierungsalgorithmen zurückliefert.

private Result evaluateContext(...)

- Klasse: edu.internet2.middleware.shibboleth.xacml.com.sun.xacml.PDP.java
- Methode: private Result evaluateContext(...)

- Beschreibung: Erstellung einer Policy bzw. Policy Menge mit anschließender Auswertung des Kontextes

Die Methode private Result evaluateContext(EvaluationCtx context) bedient sich zu Beginn dem LDAPPolicyFinderModule um mit dessen Methode public PolicyFinderResult findPolicy(EvaluationCtx context) alle relevanten Policies für den Kontext zu finden. Die gefundene Policy bzw. Policy Menge wird zur Auswertung weitergegeben.

public PolicyFinderResult findPolicy(...)

- Klasse: edu.internet2.middleware.shibboleth.xacml.com.sun.xacml.arp.LDAPPolicyFinderModule.java (siehe 7.2.4)
- Methode: public PolicyFinderResult findPolicy(...)
- Beschreibung: findet kontext-relevante Policies und bildet ggf. daraus eine Policy Menge

Die Methode public PolicyFinderResult findPolicy(URI idReference, int type) ähnelt der bereits beschriebenen Methode public ArrayList collectAllPolicies(), mit dem Unterschied, dass gefundene Policies zum Kontext passen müssen, und dass ggf. eine Policy Menge erstellt wird, sollten mehrere relevante Policies gefunden werden. Die erzeugte Policy Menge wird mit dem Prioritäts-Kombinierungsalgorithmus für Policies (`edu.internet2.middleware.shibboleth.xacml.com.sun.xacml.combine.PriorityPolicyAlg.java`) versehen.

public Result evaluate(...)

- Klasse: edu.internet2.middleware.shibboleth.xacml.com.sun.xacml.AbstractPolicy.java
- Methode: public Result evaluate(...)
- Beschreibung: erstellt ein Auswertungsergebnis

Die Methode public Result evaluate(EvaluationCtx context) ruft entsprechende Kombinierungsalgorithmen auf, welche mit Hilfe der Methode public Result combine(...) ein Auswertungsergebnis bzgl. des Kontextes erstellen. Sollten Obligationen vorhanden sein, werden diese dem Ergebnis hinzugefügt. Als Policy-Kombinierungsalgorithmus wird ein neu entwickelter Prioritätsalgorithmus (siehe 7.2.6) verwendet, welcher im beigefügten Java-Quelltext eingesehen werden kann.

6.4 Zusammenfassung

Im Laufe dieses Kapitels wurde die neue policy-basierte Privacy Management Architektur für Shibboleth schrittweise eingeführt. Zu Beginn wurde dargestellt, wie die einzelnen Anforderungen für die neue Architektur im FIM Umfeld mit Hilfe von XACML umgesetzt werden können. Dabei ergab sich u.a., dass das Rollenmodell aus XACML ausgelagert wird, um so eine Vereinfachung und bessere Performanz zu erzielen. Komponenten der neuen Architektur sind der PAP, der XACML-PEP und der XACML-PDP. Im PAP werden Policies mit zusätzlichen Informationen, wie Rolle und Benutzer, in einer übersichtlichen LDAP-Baumstruktur abgelegt. Die Architektur ist nicht an einen einzigen

LDAP Server gebunden, so dass eine verteilte Policy Administration durch ein ganzes Netz von LDAP Servern möglich wird. Der XACML-PEP fungiert als Schnittstelle zwischen Shibboleth und dem XACML-PDP. Der XACML-PEP generiert geeignete Anfragen, leitet diese an den XACML-PDP weiter und gibt dessen Auswertungsergebnisse an Shibboleth zurück. Für den XACML-PDP wird die Open Source Implementierung von Sun verwendet, welche übersichtliche Schnittstellen bietet und flexibel im Einsatz ist. So konnte z.B. ein neuer Policy-Kombinierungsalgorithmus in den XACML-PDP eingebunden wurde. Die genaue Implementierung der neuen Architektur zeigt das nächste Kapitel.

7 Implementierung der Architektur

Dieses Kapitel zeigt die neu eingeführten LDAP-Schemata und die wichtigsten geänderten bzw. neu implementierten Java-Klassen und -Methoden welche die neu entwickelte Architektur (siehe Abschnitt 6.3) umsetzen.

7.1 Implementierung des PAP

Um in LDAP Benutzergruppen und Policies entsprechend der neuen Architektur ablegen zu können, werden folgende neue Schemata benötigt. Die beiden LDAP Schemata müssen im entsprechenden Schema-Ordner des openLDAP Servers abgelegt und anschließend in LDAP importiert werden. Der zugrunde liegende openLDAP Server hat die Version 2.2.27.

7.1.1 Schema für Gruppen

Für eine genaue Beschreibung zur Erweiterung von LDAP-Schemata siehe [openLDAP].

```
1  attributetype (1.1.1.1.1
2   NAME 'XACMLgroupName'
3   DESC 'name of the group'
4   EQUALITY caseIgnoreMatch
5   SUBSTR caseIgnoreSubstringsMatch
6   SYNTAX '1.3.6.1.4.1.1466.115.121.1.15' SINGLE-VALUE)
7  attributetype (1.1.1.1.2
8   NAME 'XACMLmembers'
9   DESC 'multi-valued: members of the group'
10  EQUALITY caseIgnoreMatch
11  SUBSTR caseIgnoreSubstringsMatch
12  SYNTAX '1.3.6.1.4.1.1466.115.121.1.15')
13 objectclass (1.1.1.2.1
14  NAME 'XACMLgroup'
15  STRUCTURAL
16  MUST (XACMLgroupName \$ XACMLmembers))
```

7.1.2 Schema für Policies

```
1  attributetype (2.1.1.1.0
2   NAME 'XACMLpolicyId'
3   DESC 'name of policy'
4   EQUALITY caseIgnoreMatch
5   SUBSTR caseIgnoreSubstringsMatch
6   SYNTAX '1.3.6.1.4.1.1466.115.121.1.15' SINGLE-VALUE)
7  attributetype (2.1.1.1.1
8   NAME 'XACMLpolicy'
```

```
 DESC 'contains the policy'
 EQUALITY caseIgnoreMatch
 SUBSTR caseIgnoreSubstringsMatch
 SYNTAX '1.3.6.1.4.1.1466.115.121.1.15')
attributetype (2.1.1.1.2
 NAME 'XACMLusers'
 DESC 'multi-valued: for whom policies are applied'
 EQUALITY caseIgnoreMatch
 SUBSTR caseIgnoreSubstringsMatch
 SYNTAX '1.3.6.1.4.1.1466.115.121.1.15')
attributetype (2.1.1.1.3
 NAME 'XACMLgroupNames'
 DESC 'multi-valued: groups for which users policies are applied'
 EQUALITY caseIgnoreMatch
 SUBSTR caseIgnoreSubstringsMatch
 SYNTAX '1.3.6.1.4.1.1466.115.121.1.15')
attributetype (2.1.1.1.4
 NAME 'XACMLtargets'
 DESC 'multi-valued: targets of policy'
 EQUALITY caseIgnoreMatch
 SUBSTR caseIgnoreSubstringsMatch
 SYNTAX '1.3.6.1.4.1.1466.115.121.1.15')
attributetype (2.1.1.1.5
 NAME 'XACMLroles'
 DESC 'multi-valued: roles for which the policy is'
 EQUALITY caseIgnoreMatch
 SUBSTR caseIgnoreSubstringsMatch
 SYNTAX '1.3.6.1.4.1.1466.115.121.1.15')
objectclass (2.1.1.2.1
 NAME 'XACMLpolicy'
 STRUCTURAL
 MUST (XACMLpolicyId \$ XACMLpolicy)
 MAY (XACMLusers \$ XACMLgroupNames \$ XACMLtargets \$ XACMLroles))
```

7.2 Neuer bzw. geänderter Java Quelltext

In diesem Abschnitt befinden sich die wichtigsten geänderten bzw. neu erstellten Java-Klassen und -Methoden der neu entwickelten Architektur. Es sei hier explizit erwähnt, dass die angegebenen Java-Klassen nicht vollständig sind, sondern nur die zum Verstehen der Umsetzung der Architektur notwendigen Konstrukte enthalten. Für die neue Architektur dient der Java-Quelltext der Shibboleth-Version 1.3c und der Sun-XACML-PDP-Version 1.2 als Grundlage. Der Sun-XACML-PDP wurde dabei zu Shibboleth im Paket `edu.internet2.middleware.shibboleth.xacml` hinzugefügt. Erklärungen zur Funktionalität der einzelnen Methoden befinden sich im Abschnitt 6.3.3.Durch folgende Klassen und Methoden wurde Shibboleth mit dem Sun-XACML-PDP verknüpft und die beschriebene Architektur umgesetzt.

7.2.1 IdPProtocolSupport.java

Listing 7.1: Klasse IdPProtocolSupport

```
1  package edu.internet2.middleware.shibboleth.idp;
2
3  public class IdPProtocolSupport implements Metadata {
4
5  private String roleName = null;
6
7  /**
8  *
9  * @param principal
10 * @param relyingParty
11 * @param requester
12 * @param resource
13 *
14 * @return attributes to be released
15 *
16 * @throws AAException
17 */
18 public SAMLAttribute[] getReleaseAttributes(Principal principal,
19                                              RelyingParty relyingParty,
20                                              String requester,
21                                              URL resource)
22                                              throws AAException
23 {
24  try
25  {
26   //getting Role of principal. Role is stored in attribute
27   //urn:mace:dir:attribute-def:eduPersonEntitlement (
28   //for testing, can be changed here).
29   //Role is needed to find policies made for this role and to filter
30   //possible attributes from them.
31   try
32   {
33    AAAttribute att =
34    new AAAttribute("urn:mace:dir:attribute-def:eduPersonEntitlement");
35
36    AAAttributeSet role = new AAAttributeSet();
37    role.add(att);
38
39    resolver.resolveAttributes(principal, requester,
40    relyingParty.getIdentityProvider().getProviderId(), role);
41
42    for (Iterator iter =
43    role.getByName("urn:mace:dir:attribute-def:eduPersonEntitlement").getValues();
44    iter.hasNext();)
45    {
46     roleName = (String) iter.next();
47    }
48   }catch(Exception e){}
49   //getting all possible attributes, that could be released
50   //based on role and principal
51   URI[] potentialAttributes =
52   arpEngine.listPossibleReleaseAttributes(principal, requester, resource, roleName);
53
```

```
54    return getReleaseAttributes(principal, relyingParty,
55                                requester, resource, potentialAttributes);
56   }
57   catch (ArpProcessingException e) {
58   log.error("An error occurred while processing the ARPs for principal
59  (" + principal.getName() + "): "
60   + e.getMessage());
61   throw new AAException("Error retrieving data for principal.");
62   }
63  }
64
65
66  public SAMLAttribute[] getReleaseAttributes(Principal principal,
67                                              RelyingParty relyingParty,
68                                              String requester,
69                                              URL resource,
70                                              URI[] attributeNames)
71                                              throws AAException {
72   try
73   {
74    AAAttributeSet attributeSet = new AAAttributeSet();
75
76    for (int i = 0; i < attributeNames.length; i++)
77    {
78     AAAttribute attribute = null;
79     if (relyingParty.wantsSchemaHack())
80     {
81      attribute = new AAAttribute(attributeNames[i].toString(), true);
82     } else
83     {
84      attribute = new AAAttribute(attributeNames[i].toString(), false);
85     }
86     attributeSet.add(attribute);
87    }
88    return resolveAttributes(principal, requester,
89    relyingParty.getIdentityProvider().getProviderId(),
90    resource, attributeSet);
91
92   }
93   catch (SAMLException e)
94   {
95    log.error("An error occurred while creating attributes for
96  principal (" + principal.getName() + "): "
97    + e.getMessage());
98    throw new AAException("Error retrieving data for principal.");
99   }
100  catch (ArpProcessingException e)
101  {
102   log.error("An error occurred while processing the ARPs for
103  principal (" + principal.getName() + "): "
104   + e.getMessage());
105   throw new AAException("Error retrieving data for principal.");
106  }
107 }
108
109 /**
```

```
110 * resolve attributes for ResourceContent and for Requests
111 *
112 * @param principal
113 * @param requester
114 * @param responder
115 * @param resource
116 * @param attributeSet
117 * @return attributes to be returned to SP
118 * @throws ArpProcessingException
119 */
120 public SAMLAttribute[] resolveAttributes(Principal principal,
121                                          String requester,
122                                          String responder,
123                                          URL resource,
124                                          AAAttributeSet attributeSet)
125                                          throws ArpProcessingException {
126
127  //resolves all possible attributes that could be released
128  resolver.resolveAttributes(principal, requester, responder, attributeSet);
129
130  //resolves attributes for resourceContent
131  //(all attributes defined in resolver.ldap.xml are tried to be resolved),
132  //first get all attribute-Names defined in RESOLVER
133  String[] conditionAttributes =
134  resolver.listRegisteredAttributeDefinitionPlugIns();
135
136  //build Attributes
137  AAAttributeSet conditionSet = new AAAttributeSet();
138  for (int i = 0; i < conditionAttributes.length; i++) {
139   AAAttribute attribute = null;
140
141   try
142   {
143    attribute = new AAAttribute(conditionAttributes[i].toString(), false);
144   }
145   catch (SAMLException e)
146   {
147    log.error("FOR CONDITIONS: could not create attributes for
148 principal (" + principal.getName() + ") :"
149    + e.getMessage());
150   }
151   conditionSet.add(attribute);
152  }
153
154  //resolve attributes for conditions
155  resolver.resolveAttributes(principal, requester, responder, conditionSet);
156  //filters attributes by aplying policies
157  arpEngine.filterAttributes(attributeSet, principal, requester,
158  resource, roleName, conditionSet);
159
160  return attributeSet.getAttributes();
161 }
162 }
```

7.2.2 ArpEngine.java

Listing 7.2: Klasse ArpEngine

```
1  package edu.internet2.middleware.shibboleth.aa.arp;
2
3  /**
4  * Defines a processing engine for Attribute Release Policies.
5  */
6  public class ArpEngine {
7
8   private Element config = null;
9   private static Logger log = Logger.getLogger(ArpEngine.class.getName());
10  private ArpRepository repository;
11
12  /**
13  * Loads Arp Engine
14  *
15  * for XACML, the Repository has to be
16  * edu.internet2.middleware.shibboleth.aa.arp.provider.LdapArpRepository
17  *
18  * it has to be set in idp.xml
19  * with several ldap-uris, that are used to store the arps
20  *
21  * @throws ArpException
22  * if engine cannot be loaded
23  */
24  public ArpEngine(Element config) throws ArpException
25  {
26   if (!config.getLocalName().equals("ReleasePolicyEngine"))
27   { throw new IllegalArgumentException(); }
28
29   NodeList itemElements =
30   config.getElementsByTagNameNS(IdPConfig.configNameSpace, "ArpRepository");
31
32   if (itemElements.getLength() > 1)
33   {
34    log.warn("Encountered multiple <ArpRepository> configuration elements.
35 Arp Engine currently only supports one. Using first ...");
36   }
37
38   if (itemElements.getLength() == 0)
39   {
40    log.error("No <ArpRepsitory/> specified for this Arp Engine.");
41    throw new ArpException("Could not start Arp Engine.");
42   }
43
44   try
45   {
46    repository = ArpRepositoryFactory.getInstance((Element) itemElements.item(0));
47    //getting the Element config available
48    this.config = config;
49   }
50   catch (ArpRepositoryException e)
51   {
52    log.error("Could not start Arp Engine: " + e);
53    throw new ArpException("Could not start Arp Engine.");
```

```
54   }
55  }
56
57  /**
58  * Determines which attributes MIGHT be releasable for a given request.
59  * This function may be used to determine which
60  * attributes to resolve when a request for all attributes is made.
61  * This is done for performance reasons only. ie:
62  * The resulting attributes must still be filtered before release.
63  *
64  * @return an array of <code>URI</code> objects that name the possible attributes
65  */
66  public URI[] listPossibleReleaseAttributes(Principal principal,
67                                             String requester,
68                                             URL resource,
69                                             String roleName)
70                                             throws ArpProcessingException
71  {
72   //here attributes will be stored
73   Set possibleReleaseSet = new HashSet();
74
75   try
76   {
77    //returns where policies are stored
78    //String [] policyFiles = collectAllArps();
79    String[] ldapUris = repository.getLdapUris();
80
81    //is needed, that abstract-policies are created in the right way
82    SimplePDP simplePDP = null;
83    simplePDP = new SimplePDP(ldapUris, principal, log, roleName);
84
85    //create FinderModule
86    LDAPPolicyFinderModule LDAPPolicyModule =
87    new LDAPPolicyFinderModule(ldapUris, principal, log, roleName);
88
89    //colect all Policies in respect to principal and its role
90    ArrayList policies = LDAPPolicyModule.collectAllPolicies();
91
92    //out from these Policies filter possible resource attributes
93    possibleReleaseSet = LDAPPolicyModule.getPossibleReleaseSet(policies);
94   }
95   catch(Exception e)
96   {
97    log.debug("ERROR while creating possible attribute release set!!");
98   }
99
100  if (log.isDebugEnabled())
101  {
102   log.debug("##########Computed possible attribute release set.");
103   Iterator iterator = possibleReleaseSet.iterator();
104   while (iterator.hasNext())
105   {
106    log.debug("Possible attribute: " + iterator.next().toString());
107   }
108   log.debug("##########");
109  }
```

```
110   return (URI[]) possibleReleaseSet.toArray(new URI[0]);
111  }
112
113  /**
114  * Applies all applicable ARPs to a set of attributes.
115  *
116  * Possible results:
117  * //DECISION_PERMIT = 0;
118  * //DECISION_DENY = 1;
119  * //DECISION_INDETERMINATE = 2;
120  * //DECISION_NOT_APPLICABLE = 3;
121  *
122  * @return the attributes to be released
123  */
124  public void filterAttributes(ArpAttributeSet attributes,
125                               Principal principal,
126                               String requester,
127                               URL resource,
128                               String roleName,
129                               ArpAttributeSet conditionAttributes)
130                               throws ArpProcessingException
131  {
132
133   log.debug("#######XACML-PDP is filtering attributes...#########");
134   log.debug("Attribute requester is: " + requester);
135   log.debug("Principal is: " + principal.getName());
136   log.debug("User Role is: " + roleName);
137   try
138   {
139    SimplePDP simplePDP = null;
140
141    //returns where policies are stored
142    //String [] policyFiles = collectAllArps();
143    String[] ldapUris = repository.getLdapUris();
144
145    //create PDP with LDAP uris as input
146
147    simplePDP = new SimplePDP(ldapUris, principal, log, roleName);
148
149    //if there are no attributes for making a request
150    ArpAttributeIterator arpIterator = attributes.arpAttributeIterator();
151    if (!arpIterator.hasNext())
152    {
153     log.debug("ARP Engine was asked to apply filter to empty attribute set.");
154     return;
155    }
156
157    //############################
158    // get attributes and run PDP
159    for (ArpAttributeIterator attrIterator = attributes.arpAttributeIterator();
160    attrIterator.hasNext();)
161    {
162
163     String attrName = attrIterator.nextArpAttribute().getName();
164     log.debug("#### Creating decision for attribute " + attrName);
165     Integer decision = new Integer(3);
```

```
166     //##########################
167     //run PDP for each attribute
168     //filter creates request and returns response from PDP
169     //until now, action and purpose is fixed to read and authorization
170
171     Result res =
172     filter(requester,attrName,"",simplePDP, conditionAttributes);
173
174     decision = new Integer(res.getDecision());
175
176     //log decisions and handle attribute release
177     if(decision.intValue() == 0)
178     {
179      log.debug("DECISION_PERMIT: " + attrName);
180     }
181     if(decision.intValue() == 1)
182     {
183      log.debug("DECISION_DENY: " + attrName);
184      attrIterator.remove();
185     }
186     if(decision.intValue() == 2)
187     {
188      log.debug("DECISION_INDETERMINATE: " + attrName);
189      attrIterator.remove();
190     }
191     if(decision.intValue() == 3)
192     {
193      log.debug("NOT_APPLICABLE: " + attrName);
194      attrIterator.remove();
195     }
196
197     //print out in log the OBLIGATIONS for the attribute
198     if(!res.getObligations().isEmpty())
199     {
200      log.debug("Obligation to be fulfilled: ");
201
202      Set ob = res.getObligations();
203      Iterator iterator = ob.iterator();
204
205      while (iterator.hasNext())
206      {
207       Obligation o = (Obligation)iterator.next();
208
209       log.debug("<Obligation ObligationId=\"" + o.getId().toString() +
210       "\" FulfillOn=\"" + Result.DECISIONS[o.getFulfillOn()] + "\">");
211
212       Iterator it = o.getAssignments().iterator();
213       while (it.hasNext())
214       {
215        Attribute attr = (Attribute)(it.next());
216        log.debug("<AttributeAssignment AttributeId=\"" +
217        attr.getId().toString() + "\" DataType=\"" +
218        attr.getType().toString() + "\">" +
219        attr.getValue().encode() +
220        "</AttributeAssignment>");
221       }
```

```
222         log.debug("</Obligation>");
223        }
224       }
225      }
226      log.debug("################XACML-PDP is done.################");
227     }catch(Exception e){log.debug("ERROR WITH PDP");}
228     //-----------------------------------------
229     }
230
231     /**
232     * XACML FILTER Method
233     * erzeugt request und liefert response
234     *
235     * @return Result
236     */
237     public Result filter(String subject,
238                          String resource,
239                          String action,
240                          SimplePDP PDP,
241                          ArpAttributeSet conditionAttributes)
242     {
243      //at the beginning: not applicable (3)
244      Result res = new Result(3);
245      RequestCtx request;
246      try
247      {
248
249       //if there are no attributes for creating a request with resourceContent
250       ArpAttributeIterator arpIterator =
251       conditionAttributes.arpAttributeIterator();
252       if (!arpIterator.hasNext())
253       {
254        log.debug("No attributes found for creating a resourceContent");
255        request =
256        new RequestCtx(setupSubjects(subject), setupResource(resource),
257        setupAction(), new HashSet());
258       }
259       else
260       {
261        //request plus resourceContent
262        request =
263        new RequestCtx(setupSubjects(subject), setupResource(resource),
264        setupAction(), new HashSet(), setResourceContent(conditionAttributes));
265       }
266
267       // evaluate the request####################
268       //CALLS THE PDP
269       //########################################
270
271       ResponseCtx response = PDP.evaluate(request);
272
273       //print out the response
274       //response.encode(System.out, new Indenter());
275
276       //get Decision
277       Set i = response.getResults();
```

```
278     Iterator iterator = i.iterator();
279     while (iterator.hasNext())
280     {
281      res = (Result)iterator.next();
282
283      //zum spicken:
284      //DECISION_PERMIT = 0;
285      //DECISION_DENY = 1;
286      //DECISION_INDETERMINATE = 2;
287      //DECISION_NOT_APPLICABLE = 3;
288     }
289
290    }catch(Exception e){}
291    return res;
292   }
293
294   /**
295   * Sets up the Subject section of the request. This Request only has
296   * one Subject section, and it uses the default category. To create a
297   * Subject with a different category, you simply specify the category
298   * when you construct the Subject object.
299   *
300   * @return a Set of Subject instances for inclusion in a Request
301   *
302   * @throws URISyntaxException if there is a problem with a URI
303   */
304   public Set setupSubjects(String sub) throws URISyntaxException
305   {
306
307    HashSet attributes = new HashSet();
308
309    //setup the id and value for the requesting subject
310    URI subjectId =
311    new URI("urn:oasis:names:tc:xacml:1.0:subject:service-provider");
312
313
314    //create the subject section with the subject's identity...
315    attributes.add(new Attribute(subjectId, null, null, new StringAttribute(sub)));
316
317    //bundle the attribute in a Subject with the default category
318    HashSet subjects = new HashSet();
319    subjects.add(new Subject(attributes));
320
321    return subjects;
322
323   }
324
325   /**
326   *Creates a Resource specifying the resource-id, a required attribute.
327   *
328   *@return a Set of Attributes for inclusion in a Request
329   *
330   *@throws URISyntaxException if there is a problem with a URI
331   */
332   public Set setupResource(String res) throws URISyntaxException
333   {
```

```
334   HashSet resource = new HashSet();
335
336   AnyURIAttribute value =
337   new AnyURIAttribute(new URI(res));
338
339   //create the resource using a standard, required identifier for
340   //the resource being requested
341   resource.add(new Attribute(new URI(EvaluationCtx.RESOURCE_ID),
342                null, null, value));
343
344
345   return resource;
346  }
347
348  /**
349  * Creates an Action specifying the action-id, purpose-id, ACTION, PURPOSE
350  *
351  * @return a Set of Attributes for inclusion in a Request
352  *
353  * @throws URISyntaxException if there is a problem with a URI
354  */
355  public Set setupAction() throws URISyntaxException
356  {
357
358  //TODO: Until now, ACTION and PURPOSE have fixed values.
359  //Mechanism has to be set up, that these values
360  //are set in respect to request from SP
361  //
362  //Optionally - for testing - action and purpose
363  //can be set in a xml-file in the shib-folder
364  //(e.g. /opt/shibboleth/etc/XACMLaction.xml).
365  //Therefore create an xml-file (XACMLaction.xml)
366  //with content like:
367  //
368  //<test>
369  //<action>...</action>
370  //<purpose>...</purpose>
371  //</test>
372
373   HashSet action = new HashSet();
374   //default-values for action and purpose
375   String actionValue = "read";
376   String purposeValue = "authorization";
377
378   //tries to get ACTION/PURPOSE from XACMLaction.xml
379   try
380   {
381    String base = config.getBaseURI();
382    base = base.replaceAll("idp.xml","XACMLaction.xml");
383    Document testFile = Parser.loadDom(base, false);
384    actionValue = testFile.getElementsByTagName("action").item(0).
385    getFirstChild().getNodeValue();
386    purposeValue = testFile.getElementsByTagName("purpose").item(0).
387    getFirstChild().getNodeValue();
388   }
389   catch(Exception e)
```

```
390   {
391    log.debug("=> Setting default-values for action/purpose");
392   }
393   //create ACTION
394   log.debug("ACTION: " + actionValue + " | PURPOSE: " + purposeValue);
395   URI actionId =
396   new URI("urn:oasis:names:tc:xacml:1.0:action:action-id");
397
398   action.add(new Attribute(actionId, null, null,
399   new StringAttribute(actionValue)));
400   //create PURPOSE
401   URI purpose =
402   new URI("urn:oasis:names:tc:xacml:1.0:action:purpose");
403
404   action.add(new Attribute(purpose, null, null,
405   new StringAttribute(purposeValue)));
406
407   return action;
408  }
409
410  /**
411  *
412  * Method to create resource Content out of resolved attributes
413  * that are listed in resolver
414  *
415  * @param conditionAttributes
416  * @return String of Content
417  */
418  public String setResourceContent(ArpAttributeSet conditionAttributes)
419  {
420   String content = "";
421   try
422   {
423    for (ArpAttributeIterator attrIterator =
424    conditionAttributes.arpAttributeIterator(); attrIterator.hasNext();)
425    {
426     ArpAttribute attr = attrIterator.nextArpAttribute();
427     //getting attribute name
428     String attrName = attr.getName();
429     //deleting attr definition, like "urn:mace:dir:attribute-def:..."
430     int colon = attrName.lastIndexOf(":");
431     if(colon!=-1)
432     {
433      attrName = attrName.substring(colon + 1);
434     }
435
436     for (Iterator it = attr.getValues(); it.hasNext();)
437     {
438      //getting attribute value
439      String attrValue = (String)it.next();
440
441      content = content + "<condition:" + attrName + ">" +
442      attrValue + "</condition:" + attrName + ">";
443     }
444    }
445   }catch(Exception e){log.debug("ERROR while creating RESOURCE-CONTENT");}
```

```
446
447   return content;
448  }
449
450 }
```

7.2.3 LdapArpRepository.java

Listing 7.3: Klasse LdapArpRepository

```
1  package edu.internet2.middleware.shibboleth.aa.arp.provider;
2
3  /**
4  *
5  * Class for retrieving location of LDAP Servers
6  *
7  * Policy-Finding is done by PDP (LDAPPolicyFinderModule) not here
8  *
9  * Supported method: getLdapUris
10 *
11 * @author Matthias Ebert
12 *
13 */
14 public class LdapArpRepository implements ArpRepository{
15
16  private static Logger log =
17  Logger.getLogger(FileSystemArpRepository.class.getName());
18
19  private String [] LdapUris;
20
21  public LdapArpRepository(Element config) throws ArpRepositoryException
22  {
23
24   NodeList itemElements =
25   config.getElementsByTagNameNS(IdPConfig.configNameSpace, "Path");
26
27   //retrieving all LDAPuris defined in <ReleasePolicyEngine> in idp.xml
28   LdapUris = new String[itemElements.getLength()];
29
30   for(int i = 0; i < itemElements.getLength(); i++)
31   {
32
33    Node tnode = itemElements.item(i).getFirstChild();
34    String path = null;
35    if (tnode != null && tnode.getNodeType() == Node.TEXT_NODE)
36    {
37     path = tnode.getNodeValue();
38    }
39    if (path == null || path.equals(""))
40    {
41     log.error("LDAP ARP repository path not specified.");
42     throw new ArpRepositoryException(
43     "Cannot initialize LdapArpRepository: <ArpRepository>
44 element must contain a valid <Path> element.");
45    }
```

```
46      path = path.replaceAll("file:","");
47      if (!path.endsWith("/"))
48      {
49       path += "/";
50      }
51      log.debug("LDAP uri is: " + path);
52
53      LdapUris[i] = path;
54     }
55
56   }
57
58   public String[] getLdapUris()
59   {
60    return LdapUris;
61   }
62 }
```

7.2.4 LDAPPolicyFinderModule.java

Listing 7.4: Klasse LDAPPolicyFinderModule

```
1  package edu.internet2.middleware.shibboleth.xacml.com.sun.xacml.arp;
2
3  /**
4  *finds policies stored in ldap
5  */
6  public class LDAPPolicyFinderModule extends PolicyFinderModule
7  {
8   /**
9   *The standard value that JNDI uses to do LDAP lookups
10  */
11  public static final String jndiCtx = "com.sun.jndi.ldap.LdapCtxFactory";
12
13  /**
14  * This is the LDAP server and root that we're using
15  */
16  public String serverRoot = null;
17  //example: "ldap://lxsidp02.lrz-muenchen.de:389/
18  //dc=lxsidp02,dc=lrz-muenchen,dc=de";
19
20  //the finder that is using this module
21  private PolicyFinder finder;
22
23  //the root context used for LDAP queries
24  private DirContext directory;
25
26  //the logger used in shibboleth
27  private Logger log;
28
29  private String[] LDAPuris;
30
31  private Principal principal;
32
33  private String roleName;
```

```
34
35  /**
36  *Basic constructor where you can initialize private module data.
37  */
38  public LDAPPolicyFinderModule(String [] LDAPuris,
39                                Principal principal,
40                                Logger log,
41                                String roleName) throws NamingException
42  {
43   this.log = log;
44   this.principal = principal;
45   this.LDAPuris = LDAPuris;
46   this.roleName = roleName;
47  }
48
49  public void init(PolicyFinder finder)
50  {
51   this.finder = finder;
52  }
53
54  /**
55  * - first finds all policies that apply to a principal + role (
56  *   delegated method)
57  * - then finds from these policies all matching policies
58  *   based on request (Target match)
59  * - then builds a policySet if necessary
60  *
61  * @param context representation of the request
62  *
63  * @return the result of looking for a matching policy (return PolicySet)
64  */
65  public PolicyFinderResult findPolicy(EvaluationCtx context)
66  {
67
68   AbstractPolicy abPolicy = null;
69
70   AbstractPolicy policySet = null;
71
72   String policy = null;
73   //collecting matching policies
74   ArrayList matchingPolicies = new ArrayList();
75
76   try
77   {
78    //#####################################################################
79    //collect all policies, that can be found based on PRINCIPAL and ROLE
80    ArrayList allPolicies = collectAllPolicies();
81
82    //create the factory for documentBuilder
83    DocumentBuilderFactory factory =
84    DocumentBuilderFactory.newInstance();
85    factory.setIgnoringComments(true);
86
87    DocumentBuilder db = null;
88
89    // as of 1.2, we always are namespace aware
```

```
90    factory.setNamespaceAware(true);
91
92    //we're not doing any validation
93    factory.setValidating(false);
94
95    db = factory.newDocumentBuilder();
96
97    //for each policy do:
98    for (Iterator iter = allPolicies.iterator(); iter.hasNext();)
99    {
100    policy = (String) iter.next();
101
102    //try to load the policy file
103    Document doc = db.parse(new InputSource(new StringReader(policy)));
104
105    //handle the policy, if it's a known type
106    Element Root = doc.getDocumentElement();
107    String name = Root.getTagName();
108
109    if (name.equals("Policy"))
110    {
111     abPolicy = Policy.getInstance(Root);
112     //########################################
113     //see if we match (SUBJECT/RESOURCE/ACTION)
114     MatchResult match = abPolicy.match(context);
115     int result = match.getResult();
116     if (result == MatchResult.MATCH)
117     {
118      //if it matches, add it to matchingPolicies
119      matchingPolicies.add(abPolicy);
120     }
121    }
122    else
123    {
124     // this isn't a root type that we know how to handle
125    }
126   }
127
128   //###################################
129   //now build a POLICY-SET
130   List policiesInList = matchingPolicies;
131
132   if(!matchingPolicies.isEmpty())
133   {
134
135    //specifying AnyTarget
136    Target target = new Target(null,null,null);
137
138    //COMBINING ALGORITHM: PRIORITY
139    //an policy combining algorithm that takes priorities from
140    //patameters of policies and then applies the policy with
141    //the highest priority.
142    //At euqal priorities there is first applicable.
143    PolicyCombiningAlgorithm combiningAlg = new PriorityPolicyAlg();
144
145    policySet = new PolicySet(new URI("policySet"),
```

```
146                                 combiningAlg,
147                                 target,
148                                 policiesInList);
149     }
150   }catch (Exception e)
151   {
152     // some error occured while loading the policy, so we need to
153     // return the error for the PolicyFinder to handle the problem
154     ArrayList code = new ArrayList();
155     String message = "error parsing a possibly applicable policy: " +
156     e.getMessage();
157
158     code.add(Status.STATUS_SYNTAX_ERROR);
159
160     return new PolicyFinderResult(new Status(code, message));
161   }
162
163   //if we successfully created a POLICY-SET
164   if(policySet != null)
165   {
166     return new PolicyFinderResult(policySet);
167   }
168   else
169   {
170     log.debug("unable to create PolicySet in LDAPPolicyFinderModule");
171     return new PolicyFinderResult();
172   }
173 }
174
175 /**
176 *querying all policies (with matching PRINCIPAL (looking in user and group)
177 *and ROLE)
178 *that are stored in ldap
179 */
180 public ArrayList collectAllPolicies() throws Exception
181 {
182
183   //return, including all found policies
184   ArrayList allPolicies = new ArrayList();
185
186   //in that String the found Policy will be loaded
187   String policy = null;
188
189   //roles, a policy is made for
190   ArrayList roles = new ArrayList();
191   String role = null;
192
193   //roots of site- and userarps
194   String site_ou = "ou=sitearps,ou=policies";
195   String user_ou = "ou=userarps,ou=policies";
196   ArrayList location = new ArrayList();
197   location.add(site_ou);
198   location.add(user_ou);
199
200   //getting all groups the principal is affiliated with
201   ArrayList affiliatedGroups = getAffiliatedGroups();
```

```
202
203     if(roleName == null)
204     {
205      log.debug("no ROLE could be found in LDAPPolicyFinderModule!!");
206     }
207
208     //for each LDAPuri find all policies
209     for(int i = 0; i < LDAPuris.length; i++)
210     {
211      try
212      {
213       serverRoot = LDAPuris[i];
214       //remove the / at the end of the uri
215       serverRoot = serverRoot.substring(0,serverRoot.length()-1);
216
217       Hashtable env = new Hashtable();
218       env.put(Context.INITIAL_CONTEXT_FACTORY, jndiCtx);
219       env.put(Context.PROVIDER_URL, serverRoot);
220       //Specify timeout to be 5 seconds
221       env.put("com.sun.jndi.ldap.connect.timeout", "5000");
222
223       //build LDAP connection
224       directory = new InitialDirContext(env);
225
226       //first look in siteARPS, then look in userARPS:
227       for (Iterator iter = location.iterator(); iter.hasNext();)
228       {
229        String loc = (String) iter.next();
230
231        //looking for policy entries, based on PRINCIPAL
232        //################################################
233
234        //looking in Attribute XACMLusers for PRINCIPAL
235        Attributes attrUsers = new BasicAttributes("XACMLusers",
236                                                   principal.getName(),
237                                                   true);
238        NamingEnumeration enu = directory.search(loc,attrUsers);
239
240        if (enu.hasMore())
241        {
242         // you can now iterate through the entries (Policies)
243         while(enu.hasMoreElements())
244         {
245          SearchResult result = (SearchResult)(enu.next());
246          policy = null;
247          role = null;
248          roles.clear();
249
250
251          //getting all attributes of the entry
252          Attributes attrs = result.getAttributes();
253          NamingEnumeration ids = attrs.getIDs();
254
255          //going through attributes of entry
256          while(ids.hasMore())
257          {
```

```
258
259          //AttributeId
260          String id = (String)ids.next();
261
262          //########################
263          //find the Policy Element
264          if(id.equals("XACMLpolicy"))
265          {
266           policy = attrs.get(id).toString();
267           policy = policy.replaceFirst("XACMLpolicy: ","");
268          }
269
270          //########################
271          //find all the roles for which the policy is made
272          //looking in Attribute XACMLroles
273          if(id.equals("XACMLroles"))
274          {
275           //fot each role
276           for(int a = 0; a<attrs.get(id).size(); a++)
277           {
278            role = attrs.get(id).get(a).toString();
279            role = role.replaceFirst("XACMLroles: ","");
280
281            //if name is not already stored
282            if(!roles.contains(role))
283            {
284             roles.add(role);
285            }
286           }
287          }
288
289         }
290         //if the Policy applies to the Role:
291         //add it
292         if(roles.contains(roleName) && !allPolicies.contains(policy))
293         {
294
295          //add the Policy
296          allPolicies.add(policy);
297         }
298        }
299       }
300
301       //looking in Attribute XACMLgroupNames for Principal
302       for(Iterator it = affiliatedGroups.iterator(); it.hasNext();)
303       {
304        String group = (String) it.next();
305        Attributes attrGroupName = new BasicAttributes("XACMLgroupNames",
306                                                       group,
307                                                       true);
308        NamingEnumeration en = directory.search(loc,attrGroupName);
309
310        if (en.hasMore())
311        {
312         //you can now iterate through the entries (Policies)
313         while(en.hasMoreElements())
```

```
314         {
315          SearchResult result = (SearchResult)(en.next());
316          policy = null;
317          role = null;
318          roles.clear();
319
320          //getting all attributes of the entry
321          Attributes attrs = result.getAttributes();
322          NamingEnumeration ids = attrs.getIDs();
323
324          //going through attributes of entry
325          while(ids.hasMore())
326          {
327           //AttributeId
328           String id = (String)ids.next();
329
330           //########################
331           //find the Policy Element
332           if(id.equals("XACMLpolicy"))
333           {
334            policy = attrs.get(id).toString();
335            policy = policy.replaceFirst("XACMLpolicy: ","");
336           }
337
338           //########################
339           //find all the roles for which the policy is made
340           //looking in Attribute XACMLroles
341           if(id.equals("XACMLroles"))
342           {
343            //fot each role
344            for(int a = 0; a<attrs.get(id).size(); a++)
345            {
346             role = attrs.get(id).get(a).toString();
347             role = role.replaceFirst("XACMLroles: ","");
348
349             //if name is not already stored
350             if(!roles.contains(role))
351             {
352              roles.add(role);
353             }
354            }
355           }
356
357          }
358          //if the Policy applies to the Role:
359          //add it
360          if(roles.contains(roleName) && !allPolicies.contains(policy))
361          {
362
363           //add the Policy
364           allPolicies.add(policy);
365          }
366         }
367        }
368       }
369
```

```
370       }
371      }catch(Exception e)
372      {
373       log.debug("####LDAP-Server " + serverRoot + " is not reachable!!!");
374      }
375     }
376
377     if(allPolicies.isEmpty())
378     {
379      log.debug("No Policies could be found in LDAPPolicyFinderModule");
380     }
381
382     return allPolicies;
383    }
384
385    /**
386    *@return all groups, the principal belongs to
387    */
388    private ArrayList getAffiliatedGroups() throws Exception
389    {
390
391     String groupName = null;
392     ArrayList groupNames = new ArrayList();
393
394     //for each LDAPuri find all policies
395     for(int i = 0; i < LDAPuris.length; i++)
396     {
397      try
398      {
399       serverRoot = LDAPuris[i];
400       //remove the / at the end of the uri
401       serverRoot = serverRoot.substring(0,serverRoot.length()-1);
402
403       Hashtable env = new Hashtable();
404       env.put(Context.INITIAL_CONTEXT_FACTORY, jndiCtx);
405       env.put(Context.PROVIDER_URL, serverRoot);
406       //Specify timeout to be 5 seconds
407       env.put("com.sun.jndi.ldap.connect.timeout", "5000");
408
409       //build LDAP connection
410       directory = new InitialDirContext(env);
411
412       Attributes attrMember = new BasicAttributes("XACMLmembers",
413                                                   principal.getName(),
414                                                   true);
415       NamingEnumeration enu = directory.search("ou=groups",attrMember);
416
417       if (enu.hasMore())
418       {
419        // you can now iterate through the entries (Groups)
420        while(enu.hasMoreElements())
421        {
422         SearchResult result = (SearchResult)(enu.next());
423         //getting all attributes of the entry
424         Attributes attrs = result.getAttributes();
425         NamingEnumeration ids = attrs.getIDs();
```

```
426
427         //going through attributes of entry
428         while(ids.hasMore())
429         {
430
431          //AttributeId
432          String id = (String)ids.next();
433
434          //########################
435          //find the GroupName
436          if(id.equals("XACMLgroupName"))
437          {
438           groupName = attrs.get(id).toString();
439           groupName = groupName.replaceFirst("XACMLgroupName: ","");
440
441           if(!groupNames.contains(groupName))
442           {
443            groupNames.add(groupName);
444           }
445
446          }
447         }
448        }
449       }
450      }catch(Exception e){}
451     }
452     return groupNames;
453    }
454
455    /**
456    *method for retrieving all possible release attributes
457    *
458    *@param All policies applying to a principal and its role
459    *
460    *@return possible Attributes
461    *
462    */
463    public HashSet getPossibleReleaseSet(ArrayList pol) throws Exception
464    {
465
466     HashSet possibleAttrs = new HashSet();
467     String policy = null;
468     AbstractPolicy abPolicy = null;
469
470     //create the factory for documentBuilder
471     DocumentBuilderFactory factory =
472     DocumentBuilderFactory.newInstance();
473     factory.setIgnoringComments(true);
474
475     DocumentBuilder db = null;
476
477     //as of 1.2, we always are namespace aware
478     factory.setNamespaceAware(true);
479
480     // we're not doing any validation
481     factory.setValidating(false);
```

```
482
483   db = factory.newDocumentBuilder();
484
485   //for each policy do:
486   for (Iterator iter = pol.iterator(); iter.hasNext();)
487   {
488    policy = (String) iter.next();
489
490    //try to load the policy file
491    Document doc = db.parse(new InputSource(new StringReader(policy)));
492
493    //handle the policy, if it's a known type
494    Element Root = doc.getDocumentElement();
495
496    String name = Root.getTagName();
497
498    //if we found a policy
499    if(name.equals("Policy"))
500    {
501     //get Resources
502     abPolicy = Policy.getInstance(Root);
503
504     Target tar = abPolicy.getTarget();
505
506     List res = tar.getResources();
507
508     Iterator it = res.iterator();
509     while (it.hasNext())
510     {
511      List list = (List)(it.next());
512      Iterator it2 = list.iterator();
513      while (it2.hasNext())
514      {
515       //get VALUE
516       TargetMatch tm = (TargetMatch)(it2.next());
517       String val = tm.getMatchValue().toString();
518       val = val.replaceFirst("AnyURIAttribute: ","");
519       val = val.replaceAll("\"","");
520
521       URI uri = new URI(val);
522
523       //add VALUE
524       if(!possibleAttrs.contains(uri))
525       {
526        possibleAttrs.add(uri);
527       }
528      }
529     }
530    }
531   }
532   return possibleAttrs;
533  }
534 }
```

7.2.5 SimplePDP.java

Listing 7.5: Klasse SimplePDP

```
1  package edu.internet2.middleware.shibboleth.xacml.com.sun.xacml.arp;
2
3  public class SimplePDP
4  {
5
6   //this is the actual PDP object we'll use for evaluation
7   private PDP pdp = null;
8   private Logger log;
9   private Principal principal;
10
11  /**
12  *Constructor that takes an array of filenames, each of which
13  *contains an XACML policy, and sets up a <code>PDP</code> with access
14  *to these policies only. The <code>PDP</code> is configured
15  *programatically to have only a few specific modules.
16  *
17  *@param policyFiles an arry of filenames that specify policies
18  */
19  public SimplePDP(String [] LDAPuris,
20                   Principal principal,
21                   Logger log,
22                   String roleName)
23                   throws Exception
24  {
25
26    this.log=log;
27    this.principal = principal;
28
29    //using LDAP
30    LDAPPolicyFinderModule LDAPPolicyModule =
31    new LDAPPolicyFinderModule(LDAPuris, principal, log, roleName);
32
33    //next, setup the PolicyFinder that this PDP will use
34    PolicyFinder policyFinder = new PolicyFinder(log);
35    Set policyModules = new HashSet();
36    policyModules.add(LDAPPolicyModule);
37    policyFinder.setModules(policyModules);
38
39    //now setup attribute finder modules for the current date/time and
40    //AttributeSelectors (selectors are optional, but this project does
41    //support a basic implementation)
42    CurrentEnvModule envAttributeModule = new CurrentEnvModule();
43    SelectorModule selectorAttributeModule = new SelectorModule();
44
45    //Setup the AttributeFinder just like we setup the PolicyFinder. Note
46    //that unlike with the policy finder, the order matters here. See the
47    //the javadocs for more details.
48    AttributeFinder attributeFinder = new AttributeFinder();
49    List attributeModules = new ArrayList();
50    attributeModules.add(envAttributeModule);
51    attributeModules.add(selectorAttributeModule);
52    attributeFinder.setModules(attributeModules);
53
```

```
54    //Try to load the time-in-range function, which is used by several
55    //of the examples...see the documentation for this function to
56    //understand why it's provided here instead of in the standard
57    //code base.
58    FunctionFactoryProxy proxy =
59    StandardFunctionFactory.getNewFactoryProxy();
60    FunctionFactory factory = proxy.getConditionFactory();
61    factory.addFunction(new TimeInRangeFunction());
62    FunctionFactory.setDefaultFactory(proxy);
63
64    //finally, initialize our pdp
65    pdp = new PDP(new PDPConfig(attributeFinder, policyFinder, null), log);
66
67  }
68
69  /**
70  *Evaluates the given request and returns the Response that the PDP
71  *will hand back to the PEP.
72  *
73  *@param requestFile the name of a file that contains a Request
74  *
75  *@return the result of the evaluation
76  *
77  *@throws IOException if there is a problem accessing the file
78  *@throws ParsingException if the Request is invalid
79  */
80  public ResponseCtx evaluate(RequestCtx request)
81  throws IOException, ParsingException
82  {
83    return pdp.evaluate(request);
84  }
85 }
```

7.2.6 PriorityPolicyAlg.java

Listing 7.6: Klasse PriorityPolicyAlg

```
1  package edu.internet2.middleware.shibboleth.xacml.com.sun.xacml.combine;
2
3  /**
4  *PolicyCombining Algorithm based on Priority of Policies
5  *Policy of highest priority is applied.
6  *If there are Policies with same priority => first applicable
7  */
8  public class PriorityPolicyAlg extends PolicyCombiningAlgorithm{
9
10   /**
11   *The standard URN used to identify this algorithm
12   */
13   public static final String algId =
14   "urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:" +
15   "priority";
16
17   //a URI form of the identifier
18   private static URI identifierURI;
```

```
19
20  //value of combining-parameter
21  private int par = -1;
22
23  //exception if the URI was invalid, which should never be a problem
24  private static RuntimeException earlyException;
25
26  static {
27   try {
28    identifierURI = new URI(algId);
29   } catch (URISyntaxException se) {
30    earlyException = new IllegalArgumentException();
31    earlyException.initCause(se);
32   }
33  }
34
35  /**
36  *Standard constructor.
37  */
38  public PriorityPolicyAlg()
39  {
40   super(identifierURI);
41
42   if (earlyException != null)
43   throw earlyException;
44  }
45
46  /**
47  *Applies the combining rule to the set of policies based on the
48  *evaluation context.
49  *
50  *@param context the context from the request
51  *@param policies the policies to combine
52  *
53  *@return the result of running the combining algorithm
54  */
55  public Result combine(EvaluationCtx context, List policies)
56  {
57   Iterator it = policies.iterator();
58   Result res = null;
59
60   while (it.hasNext())
61   {
62    AbstractPolicy policy = (AbstractPolicy)(it.next());
63
64    //make sure that the policy matches the context
65    MatchResult match = policy.match(context);
66
67    if (match.getResult() == MatchResult.INDETERMINATE)
68    return new Result(Result.DECISION_DENY,
69    context.getResourceId().encode());
70
71    if (match.getResult() == MatchResult.MATCH)
72    {
73     int actualPar = policy.parameter.intValue();
74
```

```
75      //check if we have a higher priority
76      if(actualPar > par)
77      {
78       par = actualPar;
79
80       //evaluate the policy
81       Result result = policy.evaluate(context);
82       int effect = result.getDecision();
83
84       //unlike in the RuleCombining version of this alg, we always
85       //return DENY if any Policy returns DENY or INDETERMINATE
86       if ((effect == Result.DECISION_DENY) ||
87       (effect == Result.DECISION_INDETERMINATE))
88       res = new Result(Result.DECISION_DENY,
89       context.getResourceId().encode(),
90       result.getObligations());
91
92       //remember if Policy said PERMIT
93       if (effect == Result.DECISION_PERMIT)
94       {
95        res = new Result(Result.DECISION_PERMIT,
96        context.getResourceId().encode(),
97        result.getObligations());
98       }
99      }
100    }
101   }
102   return res;
103  }
104 }
```

7.3 Zusammenfassung

Nachdem die neuen LDAP-Schemata für eine geeignete Policy Speicherung angegeben wurden, zeigen die wichtigsten neuen Java-Klassen und -Methoden, wie diese die neue Architektur umsetzen. Die Klasse `IdPProtocolSupport.java` dient der Architektur als Ausgangspunkt und stößt den XACML-PEP, dessen Hauptklasse `ArpEngine.java` ist, an. Die Klasse `SimplePDP.java` initialisiert den XACML-PDP, u.a. mit einem geeigneten Modul zur Policy-Auffindung (Klasse `LDAP-PolicyFinderModule.java`). Die Klasse `LdapArpRepository.java` liefert dabei die in der Konfiguration angegebenen LDAP-Server zurück. Wie die Auswertungsergebnisse miteinander kombiniert werden, regelt die Klasse `PriorityPolicyAlg.java`, welche einen Policy-Kombinierungsalgorithmus spezifiziert, der das Ergebnis der Policy mit höchster Priorität akzeptiert. Das nächste Kapitel erklärt alle notwendigen Schritte, um die neu entwickelte Architektur einsatzfähig zu machen.

8 Konfiguration und Anwendung

In diesem Kapitel werden die notwendigen Konfigurationsschritte für die neue Architektur erklärt. Danach werden Beispiele für Gruppen- und Policy-Einträge angegeben, welche Bezug auf die anfangs beschriebenen Szenarien nehmen. Des Weiteren wird die am LRZ erstellte Shibboleth-Testumgebung beschrieben.

8.1 Konfiguration der neuen Architektur

Nachdem Shibboleth 1.3c orndungsgemäß installiert und konfiguriert wurde, müssen die alten Shibboleth Java-Klassen durch die neu entwickelten Klassen ersetzt werden. Dazu werden alle Klassen, welche sich bei der Identity Provider Installation in Tomcat unter dem Shibboleth-Ordner ../WEB-INF/classes/edu befinden (Beispiel für den kompletten Pfad könnte sein: /srv/www/tomcat5/webapps-/shibboleth-idp/WEB-INF/classes/edu), durch die neuen Klassen ersetzt, welche sich im Umfang dieser Arbeit im Ordner ebertm06/Sourcen/Shibboleth-Klassen befinden.

Will man die neuen Java Klassen selbst aus dem ursprünglichen Shibboleth Quellcode erstellen, so kann wie folgt vorgegangen werden. Im Ordner ebertm06/Sourcen/Shibboleth-Patch befindet sich die Patch-Datei „shibPatch“, welche die ursprünglichen Java Sourcen durch die neuen Sourcen ersetzt. Dazu wird der Patch in das Stamm-Verzeichnis der Quelldaten von Shibboleth (.../src) kopiert und von dort mit dem Linux-Befehl `patch -p1 < shibPatch` aufgerufen. Nachdem die Sourcen erfolgreich aktualisiert wurden, können diese neu kompiliert werden. Dieses Vorgehen hat den Vorteil, dass der Patch auch auf von Shibboleth 1.3c geringfügig abweichende Shibboleth-Versionen angewendet werden kann, da im Patch nur der geänderte Quellcode enthalten ist.

Wurden die Shibboleth Klassen erfolgreich ersetzt, müssen folgende Schritte getätigt werden, um die neue Architektur funktionsfähig zu machen.

8.1.1 Konfiguration des PAP

Zu Beginn müssen die beiden LDAP-Schemata für Gruppen und Policies, welche im Abschnitt 7.1 definiert wurden, in LDAP eingebunden werden (durch Kopieren der neuen Schemata in das LDAP-Schema-Verzeichnis und anschließende Referenzierung in der LDAP-Datei slapd.conf). Danach können in LDAP geeignete Policies und Gruppen erstellt werden (siehe dazu 8.2 und 8.3). In der idp.xml des Identity Providers muss im Element `ReleasePolicyEnginge` das neu implementierte Arp Repository angegeben werden, gefolgt von einer oder mehreren LDAP Adressen (`<path>`), welche auf die Speicherungsorte der Policies verweisen. Ein Beispiel könnte so aussehen (wobei `dc=lxsidp02, dc=lrz-muenchen, dc=de` oder `dc=my-domain, dc=com` eine beliebige LDAP Wurzel darstellt, unter der dann die Policies, wie im Informationsmodell des PAP beschrieben, liegen müssen):

```
<ReleasePolicyEngine>
 <ArpRepository implementation="edu.internet2.middleware.
␣␣shibboleth.aa.arp.provider.LdapArpRepository">
      <Path>ldap://lxsidp02.lrz-muenchen.de:389/
      dc=lxsidp02,dc=lrz-muenchen,dc=de
      </Path>
      <Path>
      ldap://10.156.15.6:389/dc=my-domain,dc=com
      </Path>
 </ArpRepository>
</ReleasePolicyEngine>
```

8.1.2 Konfiguration des Resolvers

In der Datei `resolver.ldap.xml` müssen neben allen möglichen freizugebenden Attributen, auch solche Attribute eingetragen werden, welche in den Bedingungen der Policies vorkommen könnten, da auch diese von Shibboleth aufgelöst werden müssen. Ein Beispieleintrag könnte so aussehen:

```
<SimpleAttributeDefinition
 id="urn:mace:dir:attribute-def:eduPersonAffiliation">
 <DataConnectorDependency requires="directory"/>
</SimpleAttributeDefinition>
```

8.1.3 Konfiguration von Action/Purpose

Da die Seite des Service Providers noch nicht auf die neue Architektur angepasst wurde, sendet dieser in seinen Attributanfragen auch noch keine Informationen über den Verwendungszweck oder über die Handlung. Für Testzwecke kann man daher optional eine Datei mit dem Namen `XACMLaction.xml` im etc/-Verzeichnis vom Shibboleth IdP anlegen, in der die gewünschte Handlung und der Zweck des Service Providers simuliert werden können. Die Datei kann aus den Elementen `action` und/oder `purpose` bestehen, wie folgendes Beispiel zeigt:

```
<test>
   <action>read</action>
   <purpose>statistics</purpose>
</test>
```

Wird diese Datei nicht angelegt, so werden für die Handlung und für den Zweck die Standardwerte `read` und `authorization` vorausgesetzt.

8.2 LDAP Beispieleinträge

Um in LDAP neue Einträge hinzufügen zu können, werden diese zuerst in *.ldif Dateien gespeichert, welche dann in LDAP importiert werden können.

Die drei nächsten Befehle zeigen Beispiele, wie man Einträge in LDAP hinzufügen, modifizieren und entfernen kann. In Anführungszeichen steht jeweils der `rootdn`, welcher den Datenbank-Benutzer (`cn`) und die Wurzel der Datenbank (`dc`) angibt. Danach folgt jeweils der Pfad zur LDIF-Datei.

- ldapadd -x -D „cn=Manager,dc=example,dc=com“ -W -f .../exampleArp.ldif
- ldapmodify -x -D „cn=Manager,dc=example,dc=com“ -W -f .../exampleArp.ldif
- ldapdelete -x -D „cn=Manager,dc=example,dc=com“ -W -f .../exampleArp.ldif

Es folgen LDIF Beispiele für einen Policyeintrag und für einen Gruppeneintrag, wobei die Policy `exampleArp` für die Gruppe `testGroup`, also für `ebertm` und `wh`, mit der Rolle `defaultrole` gilt. Die Wurzel, unter der der Beispiel-Policyeintrag abgelegt wird, ist `sitearps.policies.example.com`.

8.2.1 LDAP-Policy

```
1  dn:                          XACMLpolicyId=exampleArp,
2                               ou=sitearps,
3                               ou=policies,
4                               dc=example,
5                               dc=com
6  objectclass:                 XACMLpolicy
7  XACMLpolicyId:               exampleArp
8  XACMLpolicy:                 <Policy>
9    ...
10   </Policy>
11 XACMLgroupNames:             testGroup
12 XACMLtargets:                https://example-target.com
13 XACMLroles:                  defaultrole
```

8.2.2 LDAP-Gruppe

```
1  dn:                          XACMLgroupName=testGroup,
2                               ou=groups,
3                               dc=example,
4                               dc=com
5  objectclass:                 XACMLgroup
6  XACMLgroupName:              testGroup
7  XACMLmembers:                ebertm
8  XACMLmembers:                wh
```

8.3 Beispiel ARPs

Die folgenden Abschnitte zeigen ARP-Beispiele, mit deren Funktionalität es möglich ist, die obigen Szenarien umzusetzen. Die Beispiele sind jedoch keine Eins-Zu-Eins-Umsetzung der Szenarien, da dies den Umfang der Arbeit unnötig vergrößern würde. Vielmehr sollen die Beispiele Lösungswege aufzeigen, wie die Szenarien umgesetzt werden können.

8.3.1 Beispiel für Szenario 3

In Szenario 3 sollen alle Attribute ohne Einschränkungen freigegeben werden. Dafür legt z.B. ein Administrator für jedes mögliche Attribut eine Policy mit einer immer geltenden Regel (siehe Zeile

48) an. Des Weiteren wird festgelegt, dass die Attribute an beliebige Service Provider (Zeilen 28 - 30) freigegeben werden. Durch das LDAP-Attribut XACMLroles (Zeilen 51 - 53) können alle gewünschten Rollen, für welche die ARP gelten soll, angegeben werden. Soll die ARP als Standard-ARP für alle Benutzer gelten, so kann dies durch die Angabe einer Gruppe geschehen (Zeile 50), in der alle Benutzer erfasst sind. Für jedes freizugebende Attribut muss eine eigene ARP angegeben werden. Aus Gründen der Implementierung (Filterung möglicher Attribute) ist es nicht möglich, mit Hilfe eines Tags wie `<AnyResource/>` (auf Policy-Ebene) eine einzige ARP zu schreiben, die für alle freizugebenden Attribute gelten soll. Wie beschrieben könnte man so vorgehen, dass man eine Attributgruppe erstellt, welche alle möglichen freizugebenden Attribute zusammenfasst. Somit müsste man nur eine Policy angeben, welche sich dann auf die Attribute bezieht. Leider ist die Gruppierung von Attributen des Typs `anyURI` (im Element Resource ist der Datentyp anyURI nötig) jedoch noch nicht im Sun XACML PDP implementiert. Die Gruppierung von Attributen des Typs String (wie z.B. Subject/Action/Purpose) sind bereits im Sun XACML PDP implementiert.

```
1  dn:                                XACMLpolicyId=arp,ou=sitearps,ou=policies,
2   dc=lxsidp02,dc=lrz-muenchen,dc=de
3  objectclass:                       XACMLpolicy
4  XACMLpolicyId:                     arp
5  XACMLpolicy:                       <Policy
6   xmlns="urn:oasis:names:tc:xacml:1.0:policy"
7   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
8   PolicyId="ARP"
9   RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0
10 ␣:rule-combining-algorithm:ordered-permit-overrides">
11  <Description>
12  ATTRIBUT FREIGEBEN
13   Attribut: EduPersonAffiliation
14   Bedingung:-
15   Zweck: -
16   Handlung: -
17   Obligation:-
18   Nutzer: https://lxssp02.lrz-muenchen.de
19   Kombination:-
20   Nicht-Benutzer-Attribute:-
21   </Description>
22  <CombinerParameters>
23  <CombinerParameter ParameterName="ARPpriority">
24   10
25   </CombinerParameter>
26   </CombinerParameters>
27  <Target>
28  <Subjects>
29  <AnySubject/>
30   </Subjects>
31  <Resources>
32  <Resource>
33  <ResourceMatch
34   MatchId="urn:oasis:names:tc:xacml:1.0:function:anyURI-equal">
35  <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#anyURI">
36   urn:mace:dir:attribute-def:eduPersonAffiliation
37   </AttributeValue>
38  <ResourceAttributeDesignator
39   DataType="http://www.w3.org/2001/XMLSchema#anyURI"
40   AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"/>
41   </ResourceMatch>
```

```
42  </Resource>
43  </Resources>
44  <Actions>
45  <AnyAction/>
46  </Actions>
47  </Target>
48  <Rule RuleId="Rule1" Effect="Permit"/>
49  </Policy>
50 XACMLgroupNames:                    gruppeMitGewuenschtenNutzern
51 XACMLroles:                         defaultrole
52 XACMLroles:                         role1
53 XACMLroles:                         ...
```

8.3.2 Beispiel für Szenario 4

In Szenario 4 soll ein Attributwert eingeschränkt werden. Das Beispiel zeigt, wie die Freigabe des Attributs eduPersonNickName eingeschränkt werden kann. Freizugebende Werte sind nur „Matze“ und „Hias“.

Wird in einer Policy eine Bedingung verwendet, so muss im Policy-Element der Namespace `xmlns:condition='urn:mace:dir:attribute-def'` (Zeile 9) angegeben werden. Weiterhin muss die verwendete Version von XPath wie in Zeilen 26 - 28 angegeben werden. Ein XPath Ausdruck hat z.B. die Form RequestContextPath='//context:ResourceContent/condition:eduPersonNickname/-text()' (Zeilen 84 - 85). Dabei ist darauf zu achten, dass das Attribut nur durch seinen Namen selbst und durch den condition-Namespace angegeben wird, ohne z.B. urn:mace:dir:attribute:def.

```
1  dn:                                 XACMLpolicyId=arp,ou=userarps,
2   ou=policies,dc=lxsidp02,dc=lrz-muenchen,dc=de
3  objectclass:                        XACMLpolicy
4  XACMLpolicyId:                      arp
5  XACMLpolicy:                        <Policy
6   xmlns="urn:oasis:names:tc:xacml:1.0:policy"
7   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
8   xmlns:context="urn:oasis:names:tc:xacml:1.0:context"
9   xmlns:condition="urn:mace:dir:attribute-def"
10  PolicyId="ARP"
11  RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:
12 ␣rule-combining-algorithm:ordered-permit-overrides">
13  <Description>
14  EINFACHE BEDINGUNG
15   Attribut: EduPersonNickname
16   Bedingung: Attribut nur freigeben,
17   wenn eduPersonNickName den Wert Matze oder Hias hat
18   Zweck: -
19   Handlung: -
20   Obligation:-
21   Nutzer: -
22   Kombination:-
23   Nicht-Benutzer-Attribute:-
24   </Description>
25  <PolicyDefaults>
26  <XPathVersion>
27   http://www.w3.org/TR/1999/Rec-xpath-19991116
28   </XPathVersion>
```

```
29  </PolicyDefaults>
30  <CombinerParameters>
31  <CombinerParameter ParameterName="ARPpriority">
32  100
33  </CombinerParameter>
34  </CombinerParameters>
35  <Target>
36  <Subjects>
37  <AnySubject/>
38  </Subjects>
39  <Resources>
40  <Resource>
41  <ResourceMatch
42  MatchId="urn:oasis:names:tc:xacml:1.0:function:anyURI-equal">
43  <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#anyURI">
44  urn:mace:dir:attribute-def:eduPersonNickname
45  </AttributeValue>
46  <ResourceAttributeDesignator
47  DataType="http://www.w3.org/2001/XMLSchema#anyURI"
48  AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"/>
49  </ResourceMatch>
50  </Resource>
51  </Resources>
52  <Actions>
53  <AnyAction/>
54  </Actions>
55  </Target>
56  <Rule RuleId="Rule1" Effect="Permit">
57  <Target>
58  <Subjects>
59  <AnySubject/>
60  </Subjects>
61  <Resources>
62  <AnyResource/>
63  </Resources>
64  <Actions>
65  <AnyAction/>
66  </Actions>
67  </Target>
68  <Condition
69  FunctionId="urn:oasis:names:tc:xacml:1.0:function:any-of-any">
70  <Function
71  FunctionId="urn:oasis:names:tc:xacml:1.0:function:regexp-string-match"/>
72  <Apply
73  FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-bag">
74  <AttributeValue
75  DataType="http://www.w3.org/2001/XMLSchema#string">
76  Hias
77  </AttributeValue>
78  <AttributeValue
79  DataType="http://www.w3.org/2001/XMLSchema#string">
80  Matze
81  </AttributeValue>
82  </Apply>
83  <AttributeSelector
84  RequestContextPath="//context:ResourceContent/
```

```
85 ␣condition:eduPersonNickname/text()"
86  DataType="http://www.w3.org/2001/XMLSchema#string"/>
87  </Condition>
88  </Rule>
89  <!-- ... weitere Freigaben ... -->
90  <Rule RuleId="sonstNichtsFreigeben" Effect="Deny"/>
91  </Policy>
92 XACMLusers:                          Matthias
93 XACMLroles:                          defaultrole
```

8.3.3 Beispiel für Szenario 5

In Szenario 5 soll der Service Provider und der Verwendungszweck eingeschränkt werden. Das Beispiel zeigt alle dafür notwendigen Konstrukte (Service Provider: Zeilen 30 - 42 und Zweck: Zeilen 68 - 80).

```
1  dn:                                  XACMLpolicyId=
2   arp,ou=userarps,ou=policies,dc=lxsidp02,dc=lrz-muenchen,dc=de
3  objectclass:                         XACMLpolicy
4  XACMLpolicyId:                       arp
5  XACMLpolicy:                         <Policy
6   xmlns="urn:oasis:names:tc:xacml:1.0:policy"
7   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
8   xmlns:context="urn:oasis:names:tc:xacml:1.0:context"
9   PolicyId="ARP"
10  RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:
11 ␣rule-combining-algorithm:ordered-permit-overrides">
12  <Description>
13   Attribut: EduPersonNickname
14   Bedingung: -
15   Zweck: authorization
16   Handlung: -
17   Obligation:-
18   Nutzer: LMU
19   Kombination:-
20   Nicht-Benutzer-Attribute:-
21   Attribut nur freigeben, der Service Provider
22   die LMU ist und der Zweck der Autorisierung dient.
23   </Description>
24  <CombinerParameters>
25  <CombinerParameter ParameterName="ARPpriority">
26   100
27   </CombinerParameter>
28   </CombinerParameters>
29  <Target>
30  <Subjects>
31  <Subject>
32  <SubjectMatch
33   MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
34  <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
35  LMU
36   </AttributeValue>
37  <SubjectAttributeDesignator
38   DataType="http://www.w3.org/2001/XMLSchema#string"
```

```
39  AttributeId="urn:oasis:names:tc:xacml:1.0:subject:service-provider"/>
40  </SubjectMatch>
41  </Subject>
42  </Subjects>
43  <Resources>
44  <Resource>
45  <ResourceMatch
46  MatchId="urn:oasis:names:tc:xacml:2.0:function:anyURI-equal">
47  <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#anyURI">
48  urn:mace:dir:attribute-def:eduPersonNickname
49  </AttributeValue>
50  <ResourceAttributeDesignator
51  DataType="http://www.w3.org/2001/XMLSchema#anyURI"
52  AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"/>
53  </ResourceMatch>
54  </Resource>
55  </Resources>
56  <Actions>
57  <AnyAction/>
58  </Actions>
59  </Target>
60  <Rule RuleId="Rule1" Effect="Permit">
61  <Target>
62  <Subjects>
63  <AnySubject/>
64  </Subjects>
65  <Resources>
66  <AnyResource/>
67  </Resources>
68  <Actions>
69  <Action>
70  <ActionMatch
71  MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
72  <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
73  authorization
74  </AttributeValue>
75  <ActionAttributeDesignator
76  DataType="http://www.w3.org/2001/XMLSchema#string"
77  AttributeId="urn:oasis:names:tc:xacml:1.0:action:purpose"/>
78  </ActionMatch>
79  </Action>
80  </Actions>
81  </Target>
82  </Rule>
83  <!-- ... weitere Freigaben ... -->
84  <Rule RuleId="sonstNichtsFreigeben" Effect="Deny"/>
85  </Policy>
86 XACMLgroupNames:                    testGruppe
87 XACMLtargets:                       LMU
88 XACMLroles:                         defaultrole
```

8.3.4 Beispiel für Szenario 6

Das Beispiel für Szenario 6 zeigt eine Möglichkeit für die Verwendung von attributübergreifenden Bedingungen (Zeilen 81 - 96), welche analog zu normalen Bedingungen zu formulieren sind, und Obligationen (Zeilen 100 - 109). Auch werden Gruppen von Subject-Attributen definiert, um dessen Gruppierung zu verdeutlichen. Die Abhängigkeiten zu anderen Benutzern konnte im Umfang dieser Arbeit nicht umgestetzt werden. Auch die Gruppierung von Resource-Attributen wird nicht unterstützt, da der Sun PDP diese Funktionalität (Reguläre Ausdrücke auf den Typ anyURI), wie bereits erwähnt, nicht unterstützt.

```
1  dn:                                    XACMLpolicyId=
2   arp,ou=userarps,ou=policies,dc=lxsidp02,dc=lrz-muenchen,dc=de
3  objectclass:                           XACMLpolicy
4  XACMLpolicyId:                         arp
5  XACMLpolicy:                           <Policy
6   xmlns="urn:oasis:names:tc:xacml:1.0:policy"
7   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
8   xmlns:context="urn:oasis:names:tc:xacml:1.0:context"
9   xmlns:condition="urn:mace:dir:attribute-def"
10  PolicyId="ARP"
11  RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:
12 ␣rule-combining-algorithm:ordered-permit-overrides">
13  <Description>
14  ATTRIBUTUEBERGREIFENDE BEDINGUNG
15   Attribut: EduPersonNickname
16   Bedingung: Attribut nur freigeben, wenn
17   das attributuebergreifende Attribut
18   eduPersonPrincipalName den Wert foo.bar hat
19   Zweck: -
20   Handlung: -
21   Obligation: Daten nach Kursende löschen
22   Nutzer: SP1 oder SP2 oder SP3
23   Kombination:-
24   Nicht-Benutzer-Attribute:-
25   Attribut EduPersonNickname nur, wenn der SP SP1, SP2 oder SP3
26   ist, die Bedingung gilt und die Obligation umgesetzt wurde.
27   </Description>
28  <PolicyDefaults>
29  <XPathVersion>
30   http://www.w3.org/TR/1999/Rec-xpath-19991116
31   </XPathVersion>
32   </PolicyDefaults>
33  <CombinerParameters>
34  <CombinerParameter ParameterName="ARPpriority">
35   100
36   </CombinerParameter>
37   </CombinerParameters>
38  <Target>
39  <Subjects>
40  <Subject>
41  <SubjectMatch
42   MatchId="urn:oasis:names:tc:xacml:1.0:function:regexp-string-match">
43  <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
44   SP1|SP2|SP3
45   </AttributeValue>
46  <SubjectAttributeDesignator
```

```
47  DataType="http://www.w3.org/2001/XMLSchema#string"
48  AttributeId="urn:oasis:names:tc:xacml:1.0:subject:service-provider"/>
49  </SubjectMatch>
50  </Subject>
51  </Subjects>
52  <Resources>
53  <Resource>
54  <ResourceMatch
55  MatchId="urn:oasis:names:tc:xacml:2.0:function:anyURI-equal">
56  <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#anyURI">
57  urn:mace:dir:attribute-def:eduPersonNickname
58  </AttributeValue>
59  <ResourceAttributeDesignator
60  DataType="http://www.w3.org/2001/XMLSchema#anyURI"
61  AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"/>
62  </ResourceMatch>
63  </Resource>
64  </Resources>
65  <Actions>
66  <AnyAction/>
67  </Actions>
68  </Target>
69  <Rule RuleId="Rule1" Effect="Permit">
70  <Target>
71  <Subjects>
72  <AnySubject/>
73  </Subjects>
74  <Resources>
75  <AnyResource/>
76  </Resources>
77  <Actions>
78  </AnyAction/>
79  </Actions>
80  </Target>
81  <Condition
82  FunctionId="urn:oasis:names:tc:xacml:1.0:function:any-of-any">
83  <Function
84  FunctionId="urn:oasis:names:tc:xacml:1.0:
85 ␣function:regexp-string-match"/>
86  <Apply
87  FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-bag">
88  <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
89  foo.bar
90  </AttributeValue>
91  </Apply>
92  <AttributeSelector
93  RequestContextPath="//context:ResourceContent/
94 ␣condition:eduPersonPrincipalName/text()"
95  DataType="http://www.w3.org/2001/XMLSchema#string"/>
96  </Condition>
97  </Rule>
98  <!-- ... weitere Freigaben ... -->
99  <Rule RuleId="sonstNichtsFreigeben" Effect="Deny"/>
100 <Obligations>
101 <Obligation
102 ObligationId="data-has-to-be-deleted-after-end-of-course"
```

```
103  FulfillOn="Permit">
104  <AttributeAssignment AttributeId="resource"
105  DataType="http://www.w3.org/2001/XMLSchema#anyURI">
106  urn:oasis:names:tc:xacml:1.0:resource:resource-id
107  </AttributeAssignment>
108  </Obligation>
109  </Obligations>
110  </Policy>
111 XACMLgroupNames:                    testGruppe
112 XACMLtargets:                       SP1
113 XACMLtargets:                       SP2
114 XACMLtargets:                       SP3
115 XACMLroles:                         defaultrole
```

8.3.5 Beispiel für Szenario 7

Das Beispiel für Szenario 7 zeigt Ausschnitte aus drei Policies, in denen jeweils die Priorität durch das Attribut `CombinerParameter` festgelegt wurde. Es wird davon ausgegangen, dass alle 3 Policies das selbe Target haben und somit zu einer Policy Menge zusammengefasst werden. Der Prioritätsalgorithmus für Policies verwendet zur Auswertung die Policy mit höchster Priorität. Da in diesem Beispiel zwei Policies die gleiche Priorität haben, wird die Policy verwendet, welche als erstes eingelesen wurde (firstApplicable). In diesem Fall würde die Regel der zweiten Policy umgesetzt werden.

```
1  <Policy>
2   <CombinerParameters>
3   <CombinerParameter ParameterName="ARPpriority">
4   50
5    </CombinerParameter>
6    </CombinerParameters>
7  <Target>
8   ...
9  </Target>
10 <Rule>
11  ...
12 </Rule>
13 </Policy>
14
15 <Policy>
16  <CombinerParameters>
17  <CombinerParameter ParameterName="ARPpriority">
18   100
19   </CombinerParameter>
20   </CombinerParameters>
21 <Target>
22  ...
23 </Target>
24 <Rule>
25  ...
26 </Rule>
27 </Policy>
28
29 <Policy>
30  <CombinerParameters>
```

```
31  <CombinerParameter ParameterName="ARPpriority">
32   100
33   </CombinerParameter>
34   </CombinerParameters>
35 <Target>
36  ...
37 </Target>
38 <Rule>
39  ...
40 </Rule>
41 </Policy>
```

8.3.6 Beispiel für Szenario 8

Dieses Beispiel zeigt, wie man eine ARP von der Tageszeit abhängig macht (Attribut zwischen 12 und 13 Uhr nicht freigeben (Zeilen 94 - 111), sonst schon (Zeile 114)), erlaubte Handlungen und Verwendungszwecke spezifiziert (Zeilen 70 - 92) und Rollen verwendet (Zeile 118). Durch das LDAP-Attribut `XACMLroles: atWork` gilt die ARP nur während der Arbeit, d.h. es wird davon ausgegangen, dass ein Mechanismus existiert, der Bobs Benutzerrolle zu Beginn und zum Ende seiner Arbeit entsprechend hinterlegt. Ist Bob also nicht mehr in der Arbeit, so ist seine Benutzerrolle auch nicht mehr `atWork`. Daraus folgt, dass folgendes Beispiel nicht mehr zur Entscheidung herangezogen würde, sondern eine ARP, die auf Bobs neue Rolle passt. Wie Benutzerrollen verwaltet und gesetzt werden, ist nicht Teil dieser Arbeit.

```
1  dn:                                XACMLpolicyId=arp,
2   ou=sitearps,ou=policies,dc=lxsidp02,dc=lrz-muenchen,dc=de
3  objectclass:                       XACMLpolicy
4  XACMLpolicyId:                     arp
5  XACMLpolicy:                       <Policy
6   xmlns="urn:oasis:names:tc:xacml:1.0:policy"
7   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
8   PolicyId="ARP"
9   RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:
10 ␣rule-combining-algorithm:ordered-deny-overrides">
11  <Description>
12   Attribut: EduPersonAffiliation
13   Bedingung:Attribut zwischen 12 Uhr und 13 Uhr
14   nicht freigeben
15   Zweck: authorization
16   Handlung: read
17   Obligation:-
18   Nutzer: https://lxssp02.lrz-muenchen.de
19   Kombination:-
20   Nicht-Benutzer-Attribute:Uhrzeit
21   Attribut EduPersonAffiliation zum Zweck der Autorisierung
22   zwischen 12 und 13 Uhr nicht zum lesen (Handlung) freigeben.
23   </Description>
24  <CombinerParameters>
25  <CombinerParameter ParameterName="ARPpriority">
26   100
27   </CombinerParameter>
28   </CombinerParameters>
29  <Target>
30  <Subjects>
```

```
31 <Subject>
32 <SubjectMatch
33 MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
34 <AttributeValue
35 DataType="http://www.w3.org/2001/XMLSchema#string">
36 https://lxssp02.lrz-muenchen.de
37 </AttributeValue>
38 <SubjectAttributeDesignator
39 DataType="http://www.w3.org/2001/XMLSchema#string"
40 AttributeId="urn:oasis:names:tc:xacml:1.0:
41 subject:service-provider"/>
42 </SubjectMatch>
43 </Subject>
44 </Subjects>
45 <Resources>
46 <Resource>
47 <ResourceMatch
48 MatchId="urn:oasis:names:tc:xacml:1.0:function:anyURI-equal">
49 <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#anyURI">
50 urn:mace:dir:attribute-def:eduPersonAffiliation
51 </AttributeValue>
52 <ResourceAttributeDesignator
53 DataType="http://www.w3.org/2001/XMLSchema#anyURI"
54 AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"/>
55 </ResourceMatch>
56 </Resource>
57 </Resources>
58 <Actions>
59 <AnyAction/>
60 </Actions>
61 </Target>
62 <Rule RuleId="Rule1" Effect="Deny">
63 <Target>
64 <Subjects>
65 <AnySubject/>
66 </Subjects>
67 <Resources>
68 <AnyResource/>
69 </Resources>
70 <Actions>
71 <Action>
72 <ActionMatch
73 MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
74 <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
75 read
76 </AttributeValue>
77 <ActionAttributeDesignator
78 DataType="http://www.w3.org/2001/XMLSchema#string"
79 AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"/>
80 </ActionMatch>
81 <ActionMatch
82 MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
83 <AttributeValue
84 DataType="http://www.w3.org/2001/XMLSchema#string">
85 authorization
86 </AttributeValue>
```

```
87  <ActionAttributeDesignator
88  DataType="http://www.w3.org/2001/XMLSchema#string"
89   AttributeId="urn:oasis:names:tc:xacml:1.0:action:purpose"/>
90  </ActionMatch>
91  </Action>
92  </Actions>
93  </Target>
94  <Condition
95  FunctionId="http://research.sun.com/projects/
96 ␣xacml/names/function#time-in-range">
97  <Apply
98  FunctionId="urn:oasis:names:tc:xacml:1.0:function:time-one-and-only">
99  <EnvironmentAttributeDesignator
100 DataType="http://www.w3.org/2001/XMLSchema#time"
101  AttributeId="urn:oasis:names:tc:xacml:1.0:environment:current-time"/>
102 </Apply>
103 <AttributeValue
104 DataType="http://www.w3.org/2001/XMLSchema#time">
105  12:00:00
106 </AttributeValue>
107 <AttributeValue
108 DataType="http://www.w3.org/2001/XMLSchema#time">
109  13:00:00
110 </AttributeValue>
111 </Condition>
112 </Rule>
113 <!-- ... Ausserhalb der Mittagszeit ... -->
114 <Rule RuleId="sonstFreigeben" Effect="Permit"/>
115 </Policy>
116 XACMLusers:                    Bob
117 XACMLtargets:                  https://lxssp02.lrz-muenchen.de
118 XACMLroles:                    atWork
```

8.4 Testumgebung

Um die beschriebene Konfiguration und die Beispiele, bzgl. LDAP und XACML-Policies, testen zu können, wurde auf virtuellen Servern des LRZ eine Testumgebung erschaffen. Die Testumgebung, in der die neue policy-basierte Privacy Managementarchitektur für Shibboleth getestet wurde, besteht aus einem Service Provider, einem Identity Provider und zwei LDAP-Servern, wie Abbildung 8.1 zeigt. Da ein WAYF für die Testzwecke nicht notwendig ist, wurde das System als „bilateral“ konfiguriert, d.h. Shibboleth kommt ohne WAYF aus, da der Identity Provider im Service Provider fest vorgegeben ist. Über die URL `https://lxssp02.lrz-muenchen.de/secure` kann auf vom Service Provider geschützte Inhalte zugegriffen werden (Beispiel für einen Benutzernamen und Passwort: ebertm, filou). Zur Speicherung der XACML-Policies (PAP) wurden zwei LDAP-Server (Version 2.2.27) eingerichtet. Ein LDAP-Server befindet sich auf dem Server des Identity Providers, der andere liegt auf einem eigenen Server. Dadurch konnte u.a. die verteilte Ablage von XACML-Policies getestet werden. Auf dem erstgenannten LDAP-Server wurden neben den XACML-Policies auch die Benutzerinformationen hinterlegt, d.h. der PIP realisiert. PIP und PAP müssen jedoch nicht zwingend auf dem selben LDAP-Server liegen. Mit Hilfe dieser Testumgebung konnte das neu entwickelte policy-basierte Privacy Management System ausführlich und erfolgreich getestet werden.

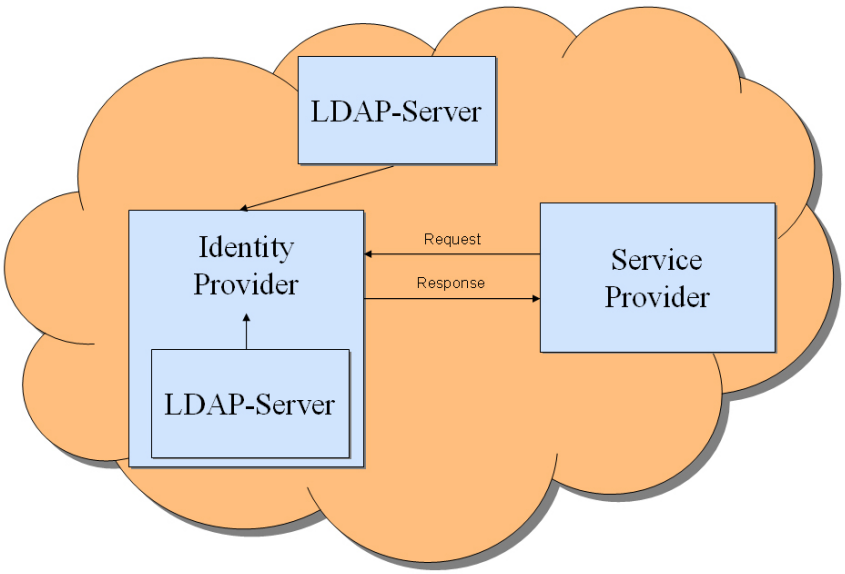

Abbildung 8.1: Testumgebung für Shibboleth

9 Zusammenfassung und Ausblick

Im Laufe dieser Diplomarbeit wurden Szenarien für das FIM-Umfeld definiert, aus denen eine Reihe von Anfroderungen an Policy Sprachen abgeleitet werden konnte. Diese Anforderungen wurden in einem Kriterienkatalog zusammengefasst, mit dessen Hilfe sechs Policy Sprachen auf ihre Eignung für eine policy-basierte Privacy Management Architektur untersucht und bewertet wurden. Dabei schnitt die Policy Sprache XACML am besten ab und wurde zur Konzeption und Implementierung der neuen Architektur herangezogen.

Es wurde eine policy-basierte Privacy Management Architektur geschaffen, welche die Mächtigkeit der ARP-Funktionalität von Shibboleth erweitert. In Policies können nun Bedingungen mit einer Vielzahl von vordefinierten Funktionen mit beinahe beliebiger Schachtelung erstellt werden. Dabei können sich die Bedingungen auf das freizugebende Attribut selbst, auf andere Attribute oder auf externe Abhängigkeiten beziehen. Des weiteren wurden neue Konstrukte, wie Obligationen, Handlungen und Verwendungszwecke eingeführt. Attribute wie Handlung, Zweck und Subjekt können mit Hilfe regulärer Ausdrücke zu Gruppen zusammengefasst werden, was die Policyerstellung erleichtert.

Die Speicherung und Kombinationsmöglichkeiten von ARPs wurde erheblich erweitert. So können nun Policies auf verschiedenen Servern mittels LDAP gespeichert und abgerufen werden. Durch den neu implementierten Prioritätsalgorithmis für Policies ist es möglich, eine Policyauswertung anhand ihrer jeweiligen Priorität durchzuführen. Auch für Regeln werden eine Vielzahl von vordefinierten Kombinierungsalgorithmen unterstützt.

In der neuen Architektur kann geregelt werden, für welche Benutzer eine Policy gelten soll. Eine Policy gilt nun also nicht mehr entweder für einen oder für alle Benutzer, sondern die Benutzer können explizit, durch die Angabe des Benutzernamens oder durch die Angabe einer Gruppe, zu welcher der Benutzer gehört, angegeben werden.

Des weiteren wurde der Grundstein für ein Rollenmodell gesetzt. Neben der Policy selbst lässt sich nun auch die Benutzerrolle angeben, für welche die entsprechende Policy gelten soll. Dadurch können Entscheidungen zusätzlich von der momentanen Rolle eines Benutzers (z.B.: Benutzer befindet sich in der Rolle eines Administrators) abhängig gemacht werden.

Die Architektur hat auch viele Ansatzpunkte geschaffen, an denen weitergearbeitet werden kann bzw. muss. So wurde in dieser Arbeit die ARP-Seite des Systems betrachtet, also die Seite des Identity Providers. Als weiteren Schritt muss auch die AAP-Seite, also der Service Provider, XACML-fähig gestaltet werden. Der Service Provider muss z.B. Verwendungszweck und gewünschte Handlung in einer Attributanfrage mit übermitteln, damit diese beim Identity Provider „real“ ausgewertet werden können. Momentan geschieht dies mit Hilfe vordefinierter Werte.

Ein weiterer offener Punkt ist die Behandlung von Obligationen. In der momentanen Architektur können Obligationen angegeben und an den PEP weitergeleitet werden, doch dieser kann diese noch nicht umsetzen. Es muss also ein System geschaffen werden, welches Obligationen vor der Attributfreigabe erzwingen kann. Dazu muss u.a. eine Menge von möglichen Obligationen definiert und deren Umsetzung implementiert werden.

Das Rollenmodell muss durch eine Instanz erweitert werden, welches Rollen kontrolliert vergeben und verwalten kann. Momentan können die Benutzerrollen beliebig und willkürlich gesetzt werden. Eine Überwachung bzw. Einschränkung findet nicht statt.

Sollte eine der nächsten Versionen des Sun XACML-PDP die Gruppierung von Resource-Attributen unterstützen, so kann auch dies in die Architektur übernommen werden. Dies würde die Policy-Erstellung vereinfachen, da dann nicht mehr für jedes Attribut einzeln eine Policy erstellt werden müsste.

Policies, welche in Abhängigkeit zu anderen Benutzern stehen, stellen einen weiteren interessanten Erweiterungsansatz dar. Neben eigenen Daten könnten dann auch Informationen anderer Benutzer zur Entscheidung herangezogen werden.

Zu guter Letzt wäre ein auf unsere Bedürfnisse abgestimmter XACML-Editor wünschenswert. Dadurch könnte die direkte Angabe von XML-Befehlen durch eine benutzerfreundliche und schnell erlernbare Grafikschnittstelle ersetzt werden, welche die Policyerstellung vereinfachen und robuster gegen Eingabefehler machen würde.

Folgende abschließende Tabelle zeigt die eben beschriebenen Punkte im Überblick:

Abgeschlossene Punkte	**Weiterführende Punkte**	**Noch offene Punkte**
(Komplexe) Bedingungen	Obligation	Gruppieren von Ressourcen
Externe Abhängigkeiten	Handlung	Attribute anderer Benutzer
Kombination von Policies	Zweck	XACML-Editor
Ressourcen	Rollenmodell	
Subjekt/Nutzer		
Gruppieren von Zweck/Handlung		
Gruppieren von Subjekten		
dezentrale Speicherung von Policies		
Angabe von Benutzer(-gruppen)		

Tabelle 9.1: Zusammenfassung

Die Diplomarbeit wird der Shibboleth-Community in Form eines Patches im Shibboleth WIKI (eine von der Community gewartete Plattform für Entwicklung, Konfiguration und sonstige Informationen im Bereich Shibboleth) zur Verfügung gestellt. Der Quelltext der Patch-Seite ist im Anhang (10) angegeben.

Link zur Patch-Seite:

`https://authdev.it.ohio-state.edu/twiki/bin/view/Shibboleth/ShibXACML`

10 Anhang

Die Angabe der im Internet veröffentlichten Patch-Seite der neuen Architektur dient an dieser Stelle der Vollständigkeit. Für eine übersichtliche Darstellung sollte die Seite im Browser unter der am Ende des vorherigen Kapitels angegebenen Adresse betrachtet werden. Die Patch-Seite wurde mit der im WIKI verwendeten Markup Sprache erstellt; dabei steht z.B. der Befehl `---+` für eine große Überschrift. Für genauere Informationen zur Formatierung im WIKI siehe: `https://authdev.it.ohio-state.edu/twiki/bin/view/TWiki/TextFormattingRules`.

Listing 10.1: Patch-Seite der neuen Architektur

```
1  ---+Shibboleth: XACML-ARPs
2  ----
3
4  This page provides a patch that is extending Shibboleth by a new ARP-
5  architecture, i.e.:
6
7     * using XACML as policy language
8     * using openLDAP for decentral storage of ARPs
9
10 After a short introduction, all required steps are explained to get
11 the extension running.
12
13 ---+++Introduction
14
15 The task of identity management is organization-internal and central ad-
16 ministration of data about coworkers, customers, partners and guests as
17 well as their access to local resources and services. This traditional
18 solution of identity management covers neither the rising necessity for
19 the access to organization-external services and resources nor the ef-
20 ficient user administration of partner enterprises (e.g. in B2B). Thus
21 the question is, how organization-internal as well as -external
22 information systems and their users can be integrated and administered
23 in domain-exceeding networks. Exactly hereby the federated identity
24 management (FIM) comes to cover those needs and requirements.
25
26 In federated identity management systems user-data, needed for authenti-
27 cation and authorization, has to be exchanged between different instances
28 (e.g. between two organizations, if a user of an organization wants to
29 use services of another organization). In order to run the exchange of
30 user information within a controlled framework, attributes release po-
31 licies (ARPs) are used. With the help of the ARPs, guidelines can be set
32 up, that restrict the transmission of data. By regulating the release of
33 data, the important aspect of data security can be guaranteed.
34
35 Regarding Shibboleth, yet the expressiveness of the Shibboleth-ARPs is
36 only sufficient for relative simple scenarios.
```

```
37
38 Within the scope of a diploma thesis at "Ludwig Maximilian Universität"
39 and "Leibniz-Rechenzentrum" in Munich, Shibboleth 1.3c !IdP has been pro-
40 totypically extended by a new ARP architecture using the powerful policy
41 language 'XACML' (see <a href="http://docs.oasis-open.org/xacml/2.0/
42 access_control-xacml-2.0-core-spec-os.pdf">XACML Specification</a>).
43
44 ---+++++Structure of the thesis:
45
46 After an introduction and description of  Shibboleth, the thesis de-
47 scribes scenarios within the range of FIM. From these scenarios criteria
48 are derived, which an ARP system must provide. In respect to these cri-
49 teria, well-known policy languages are examined regarding the suitability
50 for ARPs. The best suitable policy language is selected for the new
51 architecture and its use is described in detail. On basis of the selected
52 policy language a new policy based privacy management architecture for
53 Shibboleth is developed and imlpemented prototypically in Java. The im-
54 plementation uses an already existing policy decision point of Sun.
55 Additionally a new priority-based Policy-Combining-Algorithm is imple-
56 mented. Examples, describing the configuration and the use of the new
57 architecture, conclude the work.
58
59 ---+++++New architecture:
60
61 The new architecture is using XACML as policy language to describe ARPs.
62 In the new architecture can be expressed:
63
64    * attributes to release
65    * complex conditions (i.e. the use of several nested functions over
66      more than one attribute is supported)
67    * purposes
68    * actions
69    * obligations
70    * subjects (Service Provider)
71    * roles
72    * combining algorithms for rules/policies
73    * groupings of information
74    * policies depending on external information like date or time
75    * users or groups for which an ARP is valid
76
77 To see the complete work go to
78 <a href="http://www.nm.informatik.uni-muenchen.de/
79 pub/Diplomarbeiten/eber06/">
80 Diploma Thesis
81 </a>
82 (diploma thesis is written in German).
83
84 For a quick start see the following section.
85
86 ---+++Set up the new architecture
87
88 For detailed information about how to make Shibboleth 1.3c using the new
89 architecture, see the diploma thesis as stated above. The points below
90 are only summing up the steps to get Shibboleth using our investigated
91 XACML-ARPs:
92
```

```
93 ---+++++1) Apply Patch-file
94
95 Apply the attached patch-file 'shibPatch' (view and download it at the
96 end of this site) to the source code of Shibboleth 1.3.c to achieve the
97 new class files of shibboleth
98
99    * change directory to '.../shibboleth/src/'
100   * copy the patch-file to this directory
101   * type in the linux-command 'patch -p1 < shibPatch'
102   * compile the new source code
103   * replace all class files in your working shibboleth !IdP with the
104     new ones
105
106 ---+++++2) Configure openLDAP
107
108 Configure the ldap Server (for infos about using openLDAP see
109 <a href="http://www.openldap.org/doc/admin23/index.html">
110 openLDAP
111 </a>)
112 for enabling the storage of XACML-ARPs and usergroups
113
114   * set up one (or more) ldap-server(s) (e.g. located at your !IdP)
115   * configure the <verbatim>ArpRepository</verbatim>
116     in idp.xml like this:
117
118 <verbatim>
119 <ReleasePolicyEngine>
120  <ArpRepository implementation="edu.internet2.middleware.
121   shibboleth.aa.arp.provider.LdapArpRepository">
122     <Path>ldap://ecample.com:389/dc=example,dc=com
123     </Path>
124     <Path>
125     ...more LDAP-servers...
126     </Path>
127  </ArpRepository>
128 </ReleasePolicyEngine>
129 </verbatim>
130
131   * include the following two schemes in ldap,
132     for describing usergroups and ARPs:
133
134 <verbatim>
135 attributetype (1.1.1.1.1
136  NAME 'XACMLgroupName'
137  DESC 'name of the group'
138  EQUALITY caseIgnoreMatch
139  SUBSTR caseIgnoreSubstringsMatch
140  SYNTAX '1.3.6.1.4.1.1466.115.121.1.15' SINGLE-VALUE)
141 attributetype (1.1.1.1.2
142  NAME 'XACMLmembers'
143  DESC 'multi-valued: members of the group'
144  EQUALITY caseIgnoreMatch
145  SUBSTR caseIgnoreSubstringsMatch
146  SYNTAX '1.3.6.1.4.1.1466.115.121.1.15')
147 objectclass (1.1.1.2.1
148  NAME 'XACMLgroup'
```

```
149  STRUCTURAL
150  MUST (XACMLgroupName \$ XACMLmembers))
151
152
153 attributetype (2.1.1.1.0
154  NAME 'XACMLpolicyId'
155  DESC 'name of policy'
156  EQUALITY caseIgnoreMatch
157  SUBSTR caseIgnoreSubstringsMatch
158  SYNTAX '1.3.6.1.4.1.1466.115.121.1.15' SINGLE-VALUE)
159 attributetype (2.1.1.1.1
160  NAME 'XACMLpolicy'
161  DESC 'contains the policy'
162  EQUALITY caseIgnoreMatch
163  SUBSTR caseIgnoreSubstringsMatch
164  SYNTAX '1.3.6.1.4.1.1466.115.121.1.15')
165 attributetype (2.1.1.1.2
166  NAME 'XACMLusers'
167  DESC 'multi-valued: for whom policies are applied'
168  EQUALITY caseIgnoreMatch
169  SUBSTR caseIgnoreSubstringsMatch
170  SYNTAX '1.3.6.1.4.1.1466.115.121.1.15')
171 attributetype (2.1.1.1.3
172  NAME 'XACMLgroupNames'
173  DESC 'multi-valued: groups for which users policies are applied'
174  EQUALITY caseIgnoreMatch
175  SUBSTR caseIgnoreSubstringsMatch
176  SYNTAX '1.3.6.1.4.1.1466.115.121.1.15')
177 attributetype (2.1.1.1.4
178  NAME 'XACMLtargets'
179  DESC 'multi-valued: targets of policy'
180  EQUALITY caseIgnoreMatch
181  SUBSTR caseIgnoreSubstringsMatch
182  SYNTAX '1.3.6.1.4.1.1466.115.121.1.15')
183 attributetype (2.1.1.1.5
184  NAME 'XACMLroles'
185  DESC 'multi-valued: roles for which the policy is'
186  EQUALITY caseIgnoreMatch
187  SUBSTR caseIgnoreSubstringsMatch
188  SYNTAX '1.3.6.1.4.1.1466.115.121.1.15')
189 objectclass (2.1.1.2.1
190  NAME 'XACMLpolicy'
191  STRUCTURAL
192  MUST (XACMLpolicyId \$ XACMLpolicy)
193  MAY (XACMLusers \$ XACMLgroupNames \$ XACMLtargets \$ XACMLroles))
194 </verbatim>
195
196 ---+++++3) Create ARPs
197
198 Now you can create usergroups and XACML-ARPs, e.g.:
199
200 Entry for describing a group (stored under subtree 'ou=groups'):
201
202 <verbatim>
203 dn:                              XACMLgroupName=testGroup,
204                                  ou=groups,
```

```
205                                 dc=example,
206                                 dc=com
207 objectclass:                    XACMLgroup
208 XACMLgroupName:                 testGroup
209 XACMLmembers:                   user1
210 XACMLmembers:                   user2
211 </verbatim>
212
213 Entry for describing a policy, valid for the group stated above
214 (stored under subtree 'ou=sitearps,ou=policies'):
215
216 <verbatim>
217 dn:                             XACMLpolicyId=exampleArp,
218                                 ou=sitearps,
219                                 ou=policies,
220                                 dc=example,
221                                 dc=com
222 objectclass:                    XACMLpolicy
223 XACMLpolicyId:                  exampleArp
224 XACMLpolicy:                    <Policy>
225  ...see Example ARP...
226  </Policy>
227 XACMLgroupNames:                testGroup
228 XACMLroles:                     defaultrole
229 </verbatim>
230
231 ----+++Example ARP
232
233 Here is an example XACML-ARP. For description see the tag 'Description'.
234
235 <verbatim>
236  <Policy
237  xmlns="urn:oasis:names:tc:xacml:1.0:policy"
238  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
239  xmlns:context="urn:oasis:names:tc:xacml:1.0:context"
240  xmlns:condition="urn:mace:dir:attribute-def"
241  PolicyId="exampleArp"
242  RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:
243  rule-combining-algorithm:ordered-permit-overrides">
244  <Description>
245  Attribute to release: EduPersonNickname
246  Condition: only release eduPersonNickname, if
247  eduPersonPrincipalName has the value of foo.bar
248  Purpose: authorization
249  Action: read
250  Target: SP1 or SP2 or SP3
251  Obligation: delete data after end of term
252  Combination:
253  - ARPpriority: combines all found applicable policies based on priority
254    (parameter set in tag 'CombinerParameter')
255  - ordered-permit-overrides: combines rules within this policy
256  Role: defaultrole
257  </Description>
258  <PolicyDefaults>
259  <XPathVersion>
260  http://www.w3.org/TR/1999/Rec-xpath-19991116
```

```
261  </XPathVersion>
262  </PolicyDefaults>
263  <CombinerParameters>
264  <CombinerParameter ParameterName="ARPpriority">
265  100
266  </CombinerParameter>
267  </CombinerParameters>
268  <Target>
269  <Subjects>
270  <Subject>
271  <SubjectMatch
272  MatchId="urn:oasis:names:tc:xacml:1.0:function:regexp-string-match">
273  <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
274  SP1|SP2|SP3
275  </AttributeValue>
276  <SubjectAttributeDesignator
277  DataType="http://www.w3.org/2001/XMLSchema#string"
278  AttributeId="urn:oasis:names:tc:xacml:1.0:subject:service-provider"/>
279  </SubjectMatch>
280  </Subject>
281  </Subjects>
282  <Resources>
283  <Resource>
284  <ResourceMatch
285  MatchId="urn:oasis:names:tc:xacml:2.0:function:anyURI-equal">
286  <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#anyURI">
287  urn:mace:dir:attribute-def:eduPersonNickname
288  </AttributeValue>
289  <ResourceAttributeDesignator
290  DataType="http://www.w3.org/2001/XMLSchema#anyURI"
291  AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"/>
292  </ResourceMatch>
293  </Resource>
294  </Resources>
295  <Actions>
296  <AnyAction/>
297  </Actions>
298  </Target>
299  <Rule RuleId="Rule1" Effect="Permit">
300  <Target>
301  <Subjects>
302  <AnySubject/>
303  </Subjects>
304  <Resources>
305  <AnyResource/>
306  </Resources>
307  <Actions>
308  <Action>
309  <ActionMatch
310  MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
311  <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
312  read
313  </AttributeValue>
314  <ActionAttributeDesignator
315  DataType="http://www.w3.org/2001/XMLSchema#string"
316  AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"/>
```

```
317  </ActionMatch>
318  <ActionMatch
319  MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
320  <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
321  authorization
322  </AttributeValue>
323  <ActionAttributeDesignator
324  DataType="http://www.w3.org/2001/XMLSchema#string"
325  AttributeId="urn:oasis:names:tc:xacml:1.0:action:purpose"/>
326  </ActionMatch>
327  </Action>
328  </Actions>
329  </Target>
330  <Condition
331  FunctionId="urn:oasis:names:tc:xacml:1.0:function:any-of-any">
332  <Function
333  FunctionId="urn:oasis:names:tc:xacml:1.0:
334  function:regexp-string-match"/>
335  <Apply
336  FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-bag">
337  <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
338  foo.bar
339  </AttributeValue>
340  </Apply>
341  <AttributeSelector
342  RequestContextPath="//context:ResourceContent/
343  condition:eduPersonPrincipalName/text()"
344  DataType="http://www.w3.org/2001/XMLSchema#string"/>
345  </Condition>
346  </Rule>
347  <!-- ... nothing more to release ... -->
348  <Rule RuleId="releaseNothingMore" Effect="Deny"/>
349  <Obligations>
350  <Obligation
351  ObligationId="data-has-to-be-deleted-after-end-of-term"
352  FulfillOn="Permit">
353  <AttributeAssignment AttributeId="resource"
354  DataType="http://www.w3.org/2001/XMLSchema#anyURI">
355  urn:oasis:names:tc:xacml:1.0:resource:resource-id
356  </AttributeAssignment>
357  </Obligation>
358  </Obligations>
359  </Policy>
360 </verbatim>
361
362 ---
363
364 For errors or additional information see the "shib-error.log"
365 after authentication with Shibboleth.
366
367 It is worth mentioning that the new architecture must not be
368 seen as a finished work, in fact there are many newly-created starting
369 points.
```

Literaturverzeichnis

[AAR] OBERKNAPP, BERND: *Einführung in AAR und Shibboleth*. http://aar.vascoda.de/docs/workshop_20051012/Einfuehrung.ppt.

[ARP Patch] *Proposal for Shibboleth Attribute Release Policy Rule Constraint*. http://its.usc.edu/~bbellina/gds/software/shibboleth/ShibbolethRuleConstraint.pdf, 2006.

[ARP Shib] *IdPARPConfig*. https://authdev.it.ohio-state.edu/twiki/bin/view/Shibboleth/IdPARPConfig, 2006.

[DFN AAI] KAEHLER, U.: *DFN-AAI - Authentifizierungs- und Autorisierungs-Infrastruktur im DFN*. http://www.dfn.de/content/dienstleistungen/dfnaai/, 2006.

[E-P3P] PAUL ASHLEY, SATOSHI HADA, GÜNTER KARJOTH MATTHIAS SCHUNTER: *E-P3P Privacy Policies and Privacy Authorization*. http://portal.acm.org/ft_gateway.cfm?id=644538&type=pdf&coll=GUIDE&dl=ACM&CFID=15151515&CFTOKEN=6184618, 2002.

[EPAL Specification] PAUL ASHLEY, SATOSHI HADA, GÜNTER KARJOTH CALVIN POWERS MATTHIAS SCHUNTER: *Enterprise Privacy Authorization Language (EPAL 1.2)*. http://www.w3.org/Submission/EPAL/, 2003.

[Internet2 Shib] *Shibboleth Technical Overview*. http://shibboleth.internet2.edu/shib-tech-intro.html, 2006.

[openLDAP] FOUNDATION OPENLDAP: *Schema Specification*. http://www.openldap.org/doc/admin23/schema.html, 2005.

[P3P Specification] *The Platform for Privacy Preferences 1.0 (P3P1.0) Specification*. http://www.w3.org/TR/P3P/, 2002.

[P3P] MÖLLER, JAN: *FAQ Häufig gestellte Fragen zum Thema Datenschutz mit P3P*. http://www.datenschutzzentrum.de/faq/p3p.htm, 2003.

[PONDER] NICODEMOS DAMIANOU, NARANKER DULAY, EMIL LUPU MORRIS SLOMAN: *Ponder: A Language for Specifying Security and Management Policies for Distributed Systems*. http://www-dse.doc.ic.ac.uk/Research/policies/ponder/PonderSpec.pdf, 2000.

[SAML] TC, OASIS SECURITY SERVICES: *SAML V2.0 OASIS Standard Set*. http://docs.oasis-open.org/security/saml/v2.0/saml-2.0-os.zip.

[Shib Architecture] SCOTT CANTOR, TOM SCAVO: *Shibboleth Architecture*. http://shibboleth.internet2.edu/docs/draft-mace-shibboleth-tech-overview-latest.pdf, 2005.

[Shib Partners] INTERNET2: *Partner Info*. http://shibboleth.internet2.edu/docs/shibbing-news-Apr05.html#Partner_Info, 2006.

[TWIKI] BOREL, FRANCK: *DeploymentBackground*. https://authdev.it.ohio-state.edu/twiki/bin/view/Shibboleth/DeploymentBackground.

[WIKI Shib] *Shibboleth (Internet)*. http://de.wikipedia.org/wiki/Shibboleth_(Internet), 2006.

[XACML Comparison] ANDERSON, ANNE: *A Comparison of Two Privacy Policy Languages: EPAL and XACML*. http://research.sun.com/techrep/2005/smli_tr-2005-147/TRCompareEPALandXACML.html, 2005.

[XACML FIM] HOMMEL, WOLFGANG: *Using XACML for Privacy Control in SAML-Based Identity Federations*. http://www.mnm-team.org/pub/Publikationen/homm05a/, 2005.

[XACML PDP] SUN: *Sun's XACML Implementation*. http://sunxacml.sourceforge.net/, 2004.

[XACML Specification] MOSES, TIM: *eXtensible Access Control Markup Language (XACML) Version 2.0*. http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf, 2005.