MyShibbolethAAI - Eine Erweiterung des Shibboleth Identity Providers 1.3 zur Authentisierung und Autorisierung unter Verwendung mehrerer Benutzerdatenbanken

Markus Grandpré, Rechenzentrum der Universiät Konstanz

markus.grandpre@uni-konstanz.de

Abstract: Die auf J2EE basierende `MyShibbolethAAI` Software ist eine Erweiterung des Shibboleth Identity Providers 1.3 zur Authentisierung und Autorisierung unter Verwendung mehrerer Benutzerdatenbanken. Dieses Dokument beschreibt sowohl die Motivation, weshalb die `MyShibbolethAAI` entwickelt wurde, als auch die Anforderungen an diese Software und ihre Implementierung.

Danksagung: Ich möchte mich bei Bernd Oberknapp, Jochen Lienhard und Franck Borel von der Universitätsbibliothek Freiburg für die Zusammenarbeit bei der Entwicklung der `MyShibbolethAAI` Software bedanken. Darüberhinaus hat Herr Borel mir freundlicherweise seine Implementierung eines Authentisierungsfilters zur Verfügung gestellt. Auch bei Prof. Dr. Marcel Waldvogel, Lutz Steinfeld und Michael Längle vom Rechenzentrum der Universität Konstanz möchte ich mich bedanken, die mich bei der Arbeit an diesem Projekt unterstützt haben.

1 Einleitung

Seit Sommer 2007 verwendet das Rechenzentrum der Universität Konstanz in Zusammenarbeit mit der Universitätsbibliothek Konstanz die vom Internet2/MACE Konsortium entwickelte Shibboleth 1.3 Software zur Authentisierung und Autorisierung von Mitgliedern der Universität Konstanz, welche das Angebot von ReDI vom Campus oder von zuhause aus nutzen wollen. Die Shibboleth 1.3 Software bietet die Möglichkeit, Benutzerinformationen kontrolliert und individuell für jeden Anbieter eines webbasierten Dienstes differenziert auszugeben und zudem die Identitätskontrolle und die Zugangsberechtigung innerhalb einer Föderation, einem Zusammenschluss von verschiedenen Identity- und Service Providern, welche den Datenaustausch der Benutzerinformationen untereinander abgestimmt haben, zu realisieren. [**?**, **?**]

1.1 Der Shibboleth Identity Provider 1.3

Der im folgenden kurz beschriebene Shibboleth Identity Provider 1.3 kann derart konfiguriert werden, dass für die Authentisierung und die Autorisierung eines Benutzers entweder die gleiche oder zwei unterschiedliche Benutzerdatenbanken verwendet werden können (siehe Abbildung **??**).

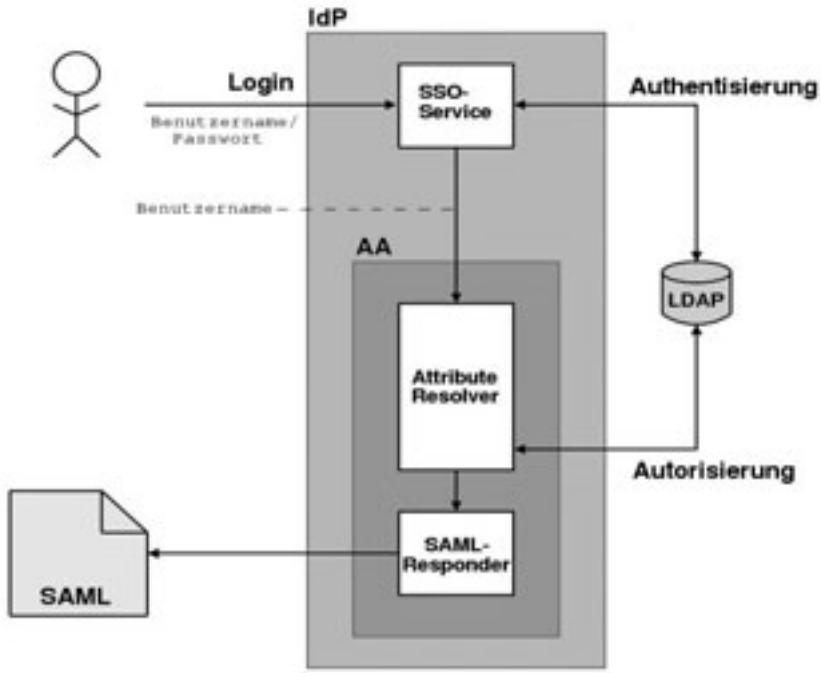

Abbildung 1: Authentisierung und Autorisierung unter Verwendung einer einzelnen Benutzerdatenbank

An der Authentisierung und der Autorisierung sind im Wesentlichen zwei Bausteine des in Java implementierten und webbasierten Shibboleth Identity Providers beteiligt. Der SSO Service dient zur Authentisierung eines Benutzers und verwendet dabei die vom Apache-Tomcat Applikationsserver angebotenen Authentisierungsmechanismen, welche die Authentisierung gegen ein relationales Datenbanksystem, einen Verzeichnisdienst oder gegen eine einfache auf XML basierende Datei unterstützen. Die Authentisierungsdaten eines Benutzers können Texteingaben (Benutzername/Passwort) oder auch digitale Signaturen (ein X509-Benutzerzertifikat) sein. Damit die Datensätze zur Authentisierung und Autorisierung stets dem gleichen Benutzer zugeordnet werden, leitet der SSO Service den Benutzernamen nach einer erfolgreichen Authentisierung an die Attribute Authority des Identity Providers weiter. Die Attribute Authority stellt anhand des empfangenen Benutzernamens eine in der Shibboleth Konfiguration festgelegte Verbindung zu einer Be-

nutzerdatenbank her, um die Autorisierungsdaten des betrefenden Benutzers zu erfragen. Webbasierte Dienste, die von einem Shibboleth Service Provider geschützt werden, definieren eigene Regeln zur Autorisierung, während ein Shibboleth Identity Provider nach der erfolgreichen Authentisierung eines Benutzers die Daten zur Evaluation der Autorisierungsregeln in Form eines SAML (Security Assertion Markup Language) Dokuments an den Service Provider versendet. [**?**, **?**]

1.2 Motivation und Anforderungen

Das Rechenzentrum der Universität Konstanz betreibt, ebenso wie die Universitätsbibliothek Konstanz, eine eigene Benutzerdatenverwaltung. Beide Verwaltungen arbeiten zwar auf derselben Datengrundlage, es bestehen dennoch Unterschiede in den Datensätzen der zur Authentisierung und Autorisierung jeweils verwendeten Benutzerdatenbanken. So verwendet das Rechenzentrum einen Mailalias und eine PopId als Benutzernamen, während die Bibliothek allgemein für jeden Benutzer eine Pseudomatrikelnummer vergibt. Darüberhinaus vergibt die Universitätsbibliothek für die Benutzung ihrer webbasierten Dienste ein eigenes Passwort an die Benutzer und pflegt auch die Daten ehemaliger Universitätsmitglieder und die Daten sog. "Walk-In-User" (Benutzer der Bibliothek, welche nicht Mitglied der Universität sind) in ihrem Datenbestand.

Da die beiden Benutzerverwaltungen verschiedene, nicht deckungsgleiche Benuzterdatenbanken betreiben und der Shibboleth Identity Provider entweder die gleiche oder bestenfalls nur zwei unterschiedliche Benutzerdatenbanken anfragen kann, entstand die Idee, den Identity Provider derart zu erweitern, dass mehr als zwei Benutzerdatenbanken zur Authentisierung und Autorisierung verwendet werden können, wobei die Benutzerdatenbank, welche vom SSO Service zur Authentisierung verwendet wird, auch von der Attribute Authority zum Lesen und Versenden der Autorisierungsdaten herangezogen werden soll (siehe Abbildung **??**). Welche Benutzerdatenbank angesprochen werden soll, hängt vom Format des Benutzernamens ab:

- Mailalias zur Authentisierung und Autorisierung eines Benutzers über das verschlüsselte `ldaps` Übertragungsprotokoll gegen einen OpenLDAP 2.0.53 Server [**?**]
- Pop-ID, zur Authentisierung und Autorisierung eines Benutzers über das verschlüsselte `https` Übertragungsprotokoll gegen ein einfaches Java Servlet, welches dem BIS Datenbanksystem vorgeschaltet ist [**?**]
- Matrikelnummer zur Authentisierung und Autorisierung eines Benutzers über das verschlüsselte `https` Übertragungsprotokoll gegen einen SOAP Service, welcher dem Libero WebOPAC Datenbanksystem vorgeschaltet ist [**?**].

Bei der Implementierung dieser Erweiterung soll der originale Java Quellcode von Shibboleth unverändert bleiben.

2 Realisierung

Um den Java Quellcode des Shibboleth Identity Providers bei der Implementierung dieser Erweiterung unverändert zu lassen, bietet es sich an, die von Shibboleth mitgelieferte Authentisierung auszuschalten und durch einen eigenen Authentisierungsfilter zu ersetzen, welcher neben dem Benutzernamen auch einen Bezeichner für die jeweilig verwendete Benutzerdatenbank an die Shibboleth Attribute Authority weiterleitet. [**?**, **?**]

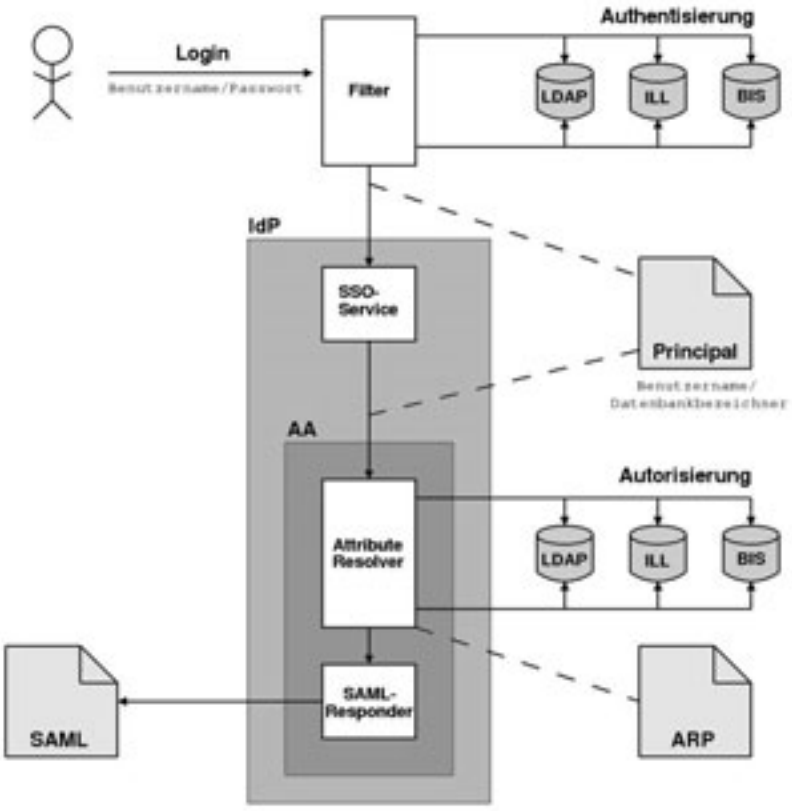

Abbildung 2: Authentisierung und Autorisierung unter Verwendung mehrerer Benutzerdatenbanken

Der Shibboleth Identity Provider bietet ebenfalls die `DataConnectorPlugIn` Schnittstelle an, die wir um eine eigeneImplementation erweitert haben, welche den Benutzernamen und den Datenbankbezeichner lesen kann, sodass die Shibboleth Attribute Authority die Autorisierungsdaten eines Benutzers aus derselben Benutzerdatenbank lesen kann, gegen die der Benutzer erfolgreich authentisiert wurde. [**?**]

2.1 Implementierung der Authentisierung

Um die von Shibboleth bereits implementierte Authentisierung durch einen eigenen Authentisierungsmechanismus zu ersetzen, haben wir in der `web.xml` Deskriptordatei der Identity Providers Webapplikation die Anweisungen zur Authentisierung gelöscht und dem SSO Service des Identity Providers ein `MyShibbolethAuthenticationFilter` Objekt vorangestellt, welches die `javax.servlet.Filter` Schnittstelle der J2EE Servlet API implementiert und alle Benutzeranfragen an den SSO Service abfängt, bevor der SSO Service aufgerufen wird. [**?**]

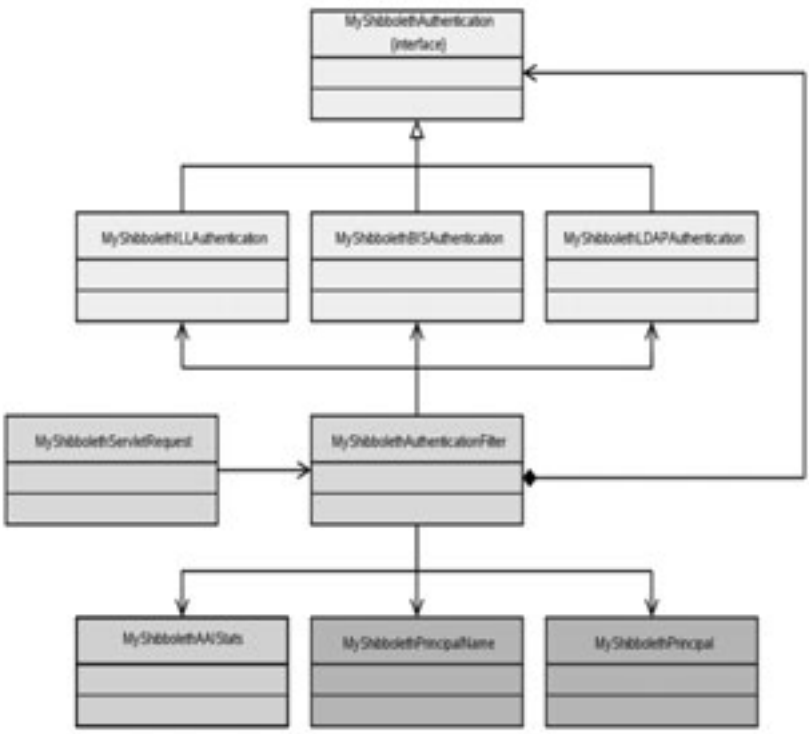

Abbildung 3: Klassendiagramm des `myshibbolethaai.authentication` Pakets

Der Filter wird mit den beiden JSP Seiten `login.jsp` und `login-error.jsp` initialisiert, auf welchen der anfragende Benutzer seinen Namen und sein Passwort in ein Formular eingeben kann. Sobald der Benutzer die Eingabe beendet hat, wird der Inhalt der Textfelder wieder an den SSO Service geschickt und somit vom Filter erneut abgefangen. Der Filter überprüft nun die Benutzereingaben auf Korrektheit und stellt eine Au-

thentisierungsanfrage an die jeweilige Benutzerdatenbank. Im Fehlerfall leitet der Filter die Benutzeranfrage auf die JSP Seite `login-error.jsp` zurück, um den Benutzer zu einer erneuten Eingabe aufzufordern. Der Filter unterscheidet daher, ob es sich bei der Benutzeranfrage um eine HTTP GET- oder um eine HTTP POST-Anfrage handelt:

- `GET`, die Anfrage ist entweder die erste Anfrage, welche vom Filter abgefangen wird oder der Filter ruft sich selber noch einmal auf, indem er die Benutzeranfrage an den SSO Service weiterleitet.
- `POST`, die Anfrage kommt von einer der beiden JSP Seiten

Erste Versuche, den Filter zu implementieren scheiterten daran, dass die Parameter der originalen Benutzeranfrage verlorengingen, sobald die Anfrage von einer der beiden JSP Seiten wieder an den Filter zurückgeleitet wurde. Das hatte zur Folge, dass die Authentisierung des Benutzers zwar abgeschlossen werden konnte, allerdings konnte der SSO Service wegen den fehlenden Parametern nicht mehr weiterarbeiten und stürzte mit zahlreichen Fehlermeldungen ab. Um diesen Fehler zu beheben, wurden die beiden JSP Seiten mit Hilfe der speziellen JSTL Tag-Bilbliothek erstellt, um die Parameter der originalen Benutzeranfrage für den Shibboleth Identity Provider zu sichern. [**?**, **?**]

Der Klasse `MyShibbolethAuthenticationFilter` fallen neben der Steuerung der Benutzeranfragen und der Überprüfung des Benutzernamens und des Passworts auf Korrektheit noch weitere Aufgaben zu:

- Authentisierungsanfrage an die Benutzerdatenbank anhand des Formats des Benutzernamens
- Kodierung des Benutzernamens und des Bezeichners der verwendeten Benutzerdatenbank
- Erzeugung eines neuen Java `Principal` Objekts bei erfolgreicher Authentisierung mit dem kodierten Benutzernamen und Datenbankbezeichner
- Erzeugung einer neuen Benutzeranfrage, welche neben den gesicherten Parametern der orginalen Anfrage auch das initialisierte `Principal` Objekt an den SSO Service weiterleitet

Nachdem die Benutzereingabe auf Korrektheit überprüft wurde, wird der Benutzername auf sein Format geprüft und ein von der `MyShibbolethAuthentication` Schnittstelle abgeleitetes Objekt erzeugt, über welches der Benutzername und das Benutzerpasswort an die jeweilige Benutzerdatenbank gesendet werden kann (siehe Abbildung **??**). Sobald der Benutzer von der Benutzerdatenbank erkannt wurde, werden der Benutzername und der Bezeichner der zur Authentisierung verwendeten Benutzerdatenbank kodiert:

```
Benutzername@Datenbankbezeichner
```

und als Namensattribut einem neu erzeugten `MyShibbolethPrincipal` Objekt hinzugefügt. Das neu erzeugte `Principal` Objekt muss als Attribut in die bestehende HTTP Sitzung des Benutzers eingetragen werden, um somit zu gewährleisten, dass der Filter, der bei jedem Aufruf das Sitzungsattrribut `Principal` überprüft, den Authentisierungsprozess nicht noch einmal ausführt. Beim letzten Durchlauf des Filters wird die originale Benutzeranfrage in eine neue Form gebracht und ein Objekt der Klasse `MyShibboleth-ServletRequest` erzeugt, welches die Methoden `getRemoteUser` und `getUser-Principal` des originalen `javax.servlet.HttpServletRequest` Objekts überschreibt und das mit Benutzernamen und Bezeichner kodierte Namensattribut an den SSO Service weiterleitet. [**?**, **?**]

```
if (req.getMethod().equals("GET")) {
  Principal principal = (Principal) req.getSession().getAttribute("principal");

  if (principal == null) {
    String url = this.loginPage + "?" + req.getQueryString();
    redirect(res, url);
  } else {
    chain.doFilter(new MyShibbolethServletRequest(req, principal), response);
  }
} else {
  this.username = req.getParameter("username");
  this.password = req.getParameter("password");
  String url = "SSO?" + req.getQueryString();

  if (username == null || password == null) {
    forward(req, res, this.errorPage + "?" + req.getQueryString());
  }

  if (isAuthenticated(this.username, this.password)) {
    this.myShibbolethPrincipal = new MyShibbolethPrincipal(
       this.principalName.encode(this.username, this.repository));
    req.getSession().setAttribute("principal", this.myShibbolethPrincipal);
    redirect(res, url);
  } else {
    forward(req, res, this.errorPage + "?" + req.getQueryString());
  }
}
```

2.2 Implementierung der Autorisierung

Die Klasse `MyShibbolethAttributeConnector` implementiert die von Shibboleth bereits mitgelieferte `DataConnectorPlugIn` Schnittstelle des Identity Providers und verwendet Objekte, die von der `MyShibbolethAuthorization` Schnittstelle abgeleitet sind, um die Autotrisierungsdaten aus derselben Benutzerdatenbank zu lesen, gegen die sich der Benutzer erfolreich authentisiert hat (siehe Abbildung **??**). Einem Objekt der Klasse `MyShibbolethAttributeConnector` fallen die folgenden Aufgaben zu:

- den Benutzernamen und den Datenbankbezeichner wieder zu dekodieren
- die Attributnamen aus der Attribute Release Policy des Identity Providers zu lesen

- anhand des Benutzernames, des Datenbankbezeichners und der Attributnamen die Autorisierungsdaten des Benutzers zu finden
- die gefundenen Attributwerte in eine Hashtabelle einzufügen, welche ebenfalls von der Shibboleth Attribute Authority gelesen wird

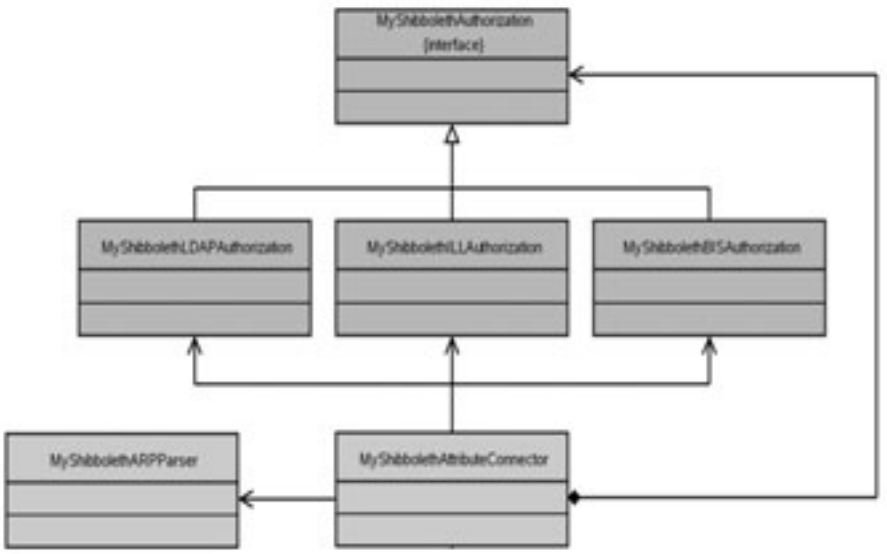

Abbildung 4: Klassendiagramm des `myshibbolethaai.authorization` Pakets

Der Benutzername und der Bezeichner der Benutzerdatenbank können mittels einer Referenz auf das vom SSO Service weitergeleitete `Principal` Objekt gelesen werden, indem das Namensattribut des `Principal` Objekts wieder dekodiert wird. [**?**]

```
String[] decodedPrincipalName = principalName.decode(principal.getName());
String   username = decodedPrincipalName[0];
String   repository = decodedPrincipalName[1];
Vector   attribute_names = this.myShibbolethARPParser.getAttributeNames(requester);
```

Da Shibboleth die Möglichkeit bietet, Benutzerinformationen kontrolliert und individuell für jeden Service Provider differenziert auszugeben, ist es nötig, den Datenbedarf des jeweiligen Service Providers zu ermitteln.

```
<Target>
 <Requester>urn:mace:uni-kn:emma_sp</Requester>
</Target>
<Attribute name="urn:mace:dir:attribute-def:eduPersonPrincipalName">
 <AnyValue release="permit"/>
</Attribute>
```

So kann anhand der `providerId` bzw. `requesterId` des Service Providers in der Attribute Release Policy des Identiy Providers nach den Attributnamen gesucht werden, deren Werte der Service Provider zur Evaluation seiner Autorisierungsregeln braucht. [**?**]

Die Schnittstelle `MyShibbolethAuthorization` und deren Implementationen werden benötigt, um die Autorisierungsdaten aus dem Benutzerdatenbank zu lesen. Die gefundenen Attributwerte werden dabei in eine Hashtabelle geschrieben. [**?**]

```
if (repository.equals("LDAP")) {
  this.myShibbolethAuthorization = new MyShibbolethLDAPAuthorization();
  this.attributes =
    this.myShibbolethAuthorization.getAttributeValues(username, attribute_names);
} else if (repository.equals("ILL")) {
  this.myShibbolethAuthorization = new MyShibbolethILLAuthorization();
  this.attributes =
    myShibbolethAuthorization.getAttributeValues(username, attribute_names);
} else if (repository.equals("BIS")) {
  this.myShibbolethAuthorization = new MyShibbolethBISAuthorization();
  this.attributes =
   myShibbolethAuthorization.getAttributeValues(username, attribute_names);
} else {
  // unknown user repository
  this.attributes = null;
}
```

Die Klasse `MyShibbolethAttributeConnector` wird in der `resolver.xml` Konfigurationsdatei des `AttributeResolvers` neben den entsprechenden Attributen, welche die Autorisierungsdaten eines Benutzers bezeichnen, definiert. Die Deklarierung eines `ScriptletAttributeDefinition` Objekts bietet die Möglichkeit, Java Quellcode in die Konfiguration der Klasse `AttributeResolver` einzubinden, um die Hashtabelle mit den gefundenen Attributnamen und Attributwerten zu füllen. [**?**, **?**]

```
<ScriptletAttributeDefinition id="urn:mace:dir:attribute-def:cn">
 <DataConnectorDependency requires="myshibbolethattributeconnector"/>
 <Scriptlet>
  Attributes attributes =
    dependencies.getConnectorResolution("myshibbolethaai_attributeconnector");
  Attribute eduPersonPrincipalName = attributes.get("cn");

  if (eduPersonPrincipalName != null &amp;&amp; eduPersonPrincipalName.size() > 0)
  {
    resolverAttribute.addValue(attributes.get("eduPersonPrincipalName").get(0));
  }
 </Scriptlet>
</ScriptletAttributeDefinition>
...
<CustomDataConnector
  id="myshibbolethaai_attributeconnector"
  class="myshibbolethaai.authorization.attributes.MyShibbolethAttributeConnector"
/>
```

Die Attribute Authority kann nun fortfahren die gefundenen Autorisierungsdaten aus der Hashtabelle zu lesen, um das benötigte SAML-Dokument zu generieren und an den anfragenden Shibboleth Service Provider zu senden. [**?**, **?**, **?**]

3 Anwendung und Ausblick

Seit November ist die `MyShibbolethAAI` Software fehlerfrei in Produktion. Sie verfügt aber nur über einen beschränkten Funktionsumfang, da bis heute nur die Klasse `MyShibbolethLDAPAuthorization` implementiert ist, welche lediglich die Anfragen nach den Autorisierungsdaten an den LDAP Benutzerdatenbank stellt. Diese Einschränkung ergibt sich aus dem Umstand, da sowohl der SOAP-Service der Libero WebOPAC-Datenbank der Universitätsbibliothek als auch das Java Servlet, welches dem BIS-Datenbanksystem des Rechenzentrums vorgeschaltet ist, derzeit nicht in der Lage sind, die geforderten Autorisierungsdaten zurückzuliefern.

Mit Ausblick auf die Entwicklung des neuen Shibboleth Identity Providers 2.0 stellt sich die Frage, ob der neue Identity Provider das Problem der Authentisierung und Autorisierung unter Verwendung mehrerer Benutzerdatenbanken bereits gelöst haben wird oder mit wievel Aufwand die nun beschriebene `MyShibbolethAAI` Software in den neuen Identity Provider integriert werden muss.

Literatur

[1] REGIONALE DATENBANK-INFORMATION BADEN-WÜRTTEMBERG, http://www-fr.redi-bw.de

[2] INTERNET2/MACE, http://shibboleth.internet2.edu

[3] CANTOR, SCOTT, *"User Authentication and Subject Identifiers in Shibboleth"*, https://spaces.internet2.edu/display/SHIB/IdPUserAuthnConfig

[4] CANTOR, SCOTT, *"IdPAttributeConfig"*, https://spaces.internet2.edu/display/SHIB/IdPAttributeConfig

[5] CANTOR, SCOTT, *"Attribute Release Policies"*, https://spaces.internet2.edu/display/SHIB/IdPARPConfig

[6] INTERNET2/OPENSAML, https://spaces.internet2.edu/display/OpenSAML/Home

[7] OPENLDAP PROJECT, http://www.openldap.org

[8] W3C, *"SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)"*, http://www.w3.org/TR/soap12-part1

[9] J2EE API SPECIFICATIONS, http://java.sun.com/javaee/5/docs/api

[10] JAVASERVER PAGES TECHNOLOGY, http://java.sun.com/products/jsp

[11] JAVA SECURITY, http://java.sun.com/javase/technologies/security

[12] JAVA NAMING AND DIRECTORY INTERFACE, http://java.sun.com/products/jndi