INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Fortgeschrittenenpraktikum

Identitätsmanagement im Gridpraktikum mit Shibboleth 2

Maximilian Kölbl

INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Fortgeschrittenenpraktikum

Identitätsmanagement im Gridpraktikum mit Shibboleth 2

Maximilian Kölbl

Aufgabensteller:	Prof. Dr. Dieter Kranzlmüller
Betreuer:	Dr. Michael Schiffers Christoph Spielmann
Abgabetermin:	10. Mai 2010

Hiermit versichere ich, dass ich den vorliegenden Bericht selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

München, den 10. Mail 2010

.....
(Unterschrift des Kandidaten)

Abstract

Der Lehrstuhl für Kommunikationssysteme und Systemprogrammierung an der LMU bietet ab dem Sommersemester 2010 ein neues Praktikum zum Thema Grid Computing an. Diese Arbeit befasst sich mit der Infrastruktur für besagtes Grid Computing Praktikum. Ziel ist es, das Identitätsmanagement zu realisieren, so dass die Teilnehmer des Praktikums authentifiziert werden können.

Shibboleth 2, Gridshib-CA 2 sowie das Globus Toolkit 4 sind die Hauptkomponenten, die Identitätsmanagement im Grid ermöglichen. In dieser Arbeit wird unter Verwendung und Verbindung dieser verschiedenen Softwarekomponenten eine Umgebung geschaffen, in der sich ein Nutzer registrieren und ein Zertifikat anfordern kann, mit dem er sich im Grid bewegen kann. Dabei wird insbesondere darauf eingegangen, wie Shibboleth 2 arbeitet, um einen Nutzer zu authentifizieren.

Inhaltsverzeichnis

1 Einleitung **1**

2 Anmeldeserver **2**
2.1 Funktion des Anmeldeservers im Praktikum 2
2.2 Implementierung 2
2.2.1 Views 2
2.2.2 Grundgerüst 4
2.2.3 Datenbank 8

3 Tomcat 6 **9**
3.1 Realms 9
3.2 Einrichtung eines JDBC Realm 9

4 Shibboleth 2 **11**
4.1 Wie es funktioniert 11
4.2 IdP Authentifizierung und Sessions 12
4.2.1 Authentifizierung 12
4.2.2 Nutzer Session 12
4.2.3 Zusammenhang zwischen Authentifizierung und Session 13
4.3 Ablauf beim Zugriff auf geschützte Ressourcen 14
4.3.1 Nutzer will auf geschützte Ressource zugreifen 14
4.3.2 Nutzer authentifiziert sich beim Identity Provider 15
4.3.3 Der IdP schickt seine Antwort an den Service Provider 16
4.3.4 Service Provider erzeugt neue Session 16

5 Gridshib-CA 2 **17**

6 Globus Toolkit 4 **21**
6.1 Execution Management 22
6.2 Datentransfer 24
6.3 Monitoring und Discovery 24

7 Zusammenhang der Komponenten **26**
7.1 Ablauf: Von der Anmeldung zum Zertifikat 26
7.2 Anmeldung beim Anmeldeserver 27
7.3 Zusammenarbeit von Shibboleth Service Provider und Identity Provider . . . 28
7.4 Authentifizierung des Nutzers 29

8 Zusammenfassung **31**

Abkürzungsverzeichnis **32**

Abbildungsverzeichnis **33**

Literaturverzeichnis **34**

1 Einleitung

Die wachsende Bedeutung von Grid Computing in der Welt der Informatik führt dazu, dass der Lehrstuhl für Kommunikationssysteme und Systemprogrammierung an der LMU ab dem Sommersemester 2010 ein neues Praktikum zum Thema Grid Computing anbietet. In diesem Praktikum sollen die Teilnehmer lernen, wie Grids funktionieren und wie man sie nutzen kann.

Da in einem Grid nicht jeder beliebige Nutzer Zugriff auf alle Ressourcen haben soll, ist ein Identitätsmanagement erforderlich. Damit die Praktikumsteilnehmer auf Ressourcen im Grid zugreifen können, müssen sie zunächst authentifiziert und autorisiert werden. Bei der Frage danach, wie sich Identitäten in einer Grid Umgebung managen lassen, die auf dem Globus Toolkit basiert, kommt man schnell zu Shibboleth 2. Shibboleth 2 wurde vom Internet2 Projekt (`http://www.internet2.edu/`) entwickelt, um verteilte Authentifizierung und Autorisierung für Webanwendungen und Webservices zu ermöglichen. Es lässt sich an eine bereits bestehende Benutzerverwaltung anbinden und bietet die Möglichkeit, nach einmaliger Authentifizierung und Autorisierung auf alle geschützten Ressourcen zuzugreifen (Single-Sign-On). Damit Shibboleth 2 wie gewünscht funktionieren kann, ist eine Zusammenarbeit mit weiteren Softwarekomponenten erforderlich.

Zunächst muss eine Datenbank mit Nutzerdaten angelegt werden. Hierfür wird eine Anmeldeserver bereit gestellt, der von den Praktikumsteilnehmern über einen Browser erreichbar ist. Dieser Server legt die Daten der Teilnehmer in einer sqlite3 Datenbank ab.

Für den Zugriff auf vorhandene Nutzerdaten wird in dieser Arbeit Tomcat 6 verwendet. Tomcat bietet die Möglichkeit, so genannte Realms zu erschaffen. Diese stellen ein Set aus Nutzern mit zusätzlichen Attributen dar, die eine Authentifizierung und Autorisierung ermöglichen. Es wird ein Realm mit der Datenbank des Anmeldeservers erzeugt, aus welchem Shibboleth 2 dann die Informationen beziehen kann, die für die Authentifizierung und Autorisierung einzelner Nutzer erforderlich sind.

GridShib-CA 2 erzeugt schließlich die Proxy-Zertifikate für die Nutzer, nachdem die Authentifizierung über Shibboleth 2 erfolgreich war. Ein Grid-Mapfile, das die Subjects dieser Zertifikate auf lokale Accounts abbildet, kann dann beim Anmeldeserver angefordert werden.

In dieser Arbeit wird zunächst auf die wichtigsten Softwarekomponenten eingegangen und schließlich erklärt, wie diese zusammenarbeiten, um das Identitätsmanagement zu realisieren.

2 Anmeldeserver

2.1 Funktion des Anmeldeservers im Praktikum

Der mit Ruby erstellte Registration Server (oder Anmeldeserver) stellt den Praktikumsteilnehmern ein Formular zur Verfügung, über das sie sich mit einem Benutzernamen und einem Passwort registrieren können. Die angegebenen Daten werden in einer sqlite3 Datenbank gespeichert, auf die später zugegriffen wird, um den Nutzer gegenüber Shibboleth 2 zu authentifizieren.
Ausserdem erzeugt der Registration Server das Grid-Mapfile unter Verwendung der von der Gridshib-CA 2 ausgestellten Zertifikate.
Für mehr Informationen über Ruby siehe z.B. [Lon01], [Neu09].

2.2 Implementierung

2.2.1 Views

Im Verzeichnis views befinden sich die Dateien anmeldung.erb und home.erb. Diese Dateien stellen den Teil des Servers zur Verfügung, der dem Studenten im Browser angezeigt wird.

Listing 2.1: anmeldung.erb

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
3    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
4
5  <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en"
6  lang="en">
7  <head>
8    <title>Anmeldung</title>
9    <link rel="stylesheet" href="/css/minimal.css" type="text/css"
10 media="screen" title="no title" charset="utf-8"/>
11 </head>
12
13 <body>
14   <h1>Anmeldung</h1>
15
16   <p>Gib hier deine Daten ein, um dich anzumelden:</p>
17
18   <form action="/anmeldung" method="post" accept-charset="utf-8">
19     <fieldset id="anmeldung" class="">
20       <legend>Anmeldung</legend>
21
```

```
22        <% unless @user.errors.empty? %>
23          <p>Es sind Fehler aufgetreten:</p>
24
25          <ul class="errors">
26            <% @user.errors.values.flatten.each do |error|%>
27            <li><%= error %></li>
28            <% end%>
29          </ul>
30        <% end%>
31
32        <div <%= @user.errors[:name].empty? ? '' : 'class=
33 "error"'%>>
34          <input type="text" name="user[name]" value=
35 "<%= @user.name %>" id="name_id"/>
36          <label for="name_id">Benutzername:<t><t></label>
37        </div>
38        <div <%= @user.errors[:matnr].empty? ? '' : 'class=
39 "error"'%>>
40          <input type="text" name="user[matnr]" value=
41 "<%= @user.matnr %>" id="matnr_id"/>
42          <label for="matnr_id">Matrikelnummer:<t><t></label>
43        </div>
44
45        <div <%= @user.errors[:password].empty? ? '' : 'class=
46 "error"'%>>
47          <input type="password" name="user[password]" value=""
48 id="password_id"/>
49          <label for="password_id">Passwort:<t><t></label>
50        </div>
51        <div <%= @user.errors[:password].empty? ? '' : 'class=
52 "error"'%>>
53          <input type="password" name="user[password_confirmation]"
54 value="" id="password_confirmation"/>
55          <label for="password_confirmation">Bestätigung:<<t><t>
56 </label>
57        </div>
58
59        <button type="submit" name="submit">Einloggen &rarr;
60 </button>
61
62      </fieldset>
63    </form>
64
65 </body>
66 </html>
```

Es werden Eingabefelder für Benutzernamen, Matrikelnummer und Passwort erzeugt. Das Passwort muss in einem weiteren Feld bestätigt werden. Nach erfolgreicher Anmeldung erfolgt die Weiterleitung auf die von home.erb erzeugte Seite.

Listing 2.2: home.erb

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en"
 lang="en">
<head>
  <title>Home</title>
  <link rel="stylesheet" href="/css/minimal.css" type="text/css"
media="screen" title="no title" charset="utf-8"/>

</head>

<body>
  <h1>Uebersichtsseite</h1>

  <dl>
    <dt>Name:</dt>
    <dd><%= @user.name %></dd>
    <dt>Matrikelnummer:</dt>
    <dd><%= @user.matnr %></dd>
  </dl>

  <p><a href="/logout">Logout</a></p>
</body>
</html>
```

Dies ist eine einfach Übersichtsseite, auf der die eingegeben Anmeldedaten noch einmal angezeigt werden.

2.2.2 Grundgerüst

Das Grundgerüst des Servers bilden die Dateien certificate_server.rb, config.ru und user.rb.

Listing 2.3: config.ru

```


require 'dm-core'
require 'certificate_server'

DataMapper.setup( :default, "sqlite3://#{Dir.pwd}/users.db" )

use Rack::Session::Cookie, :key => 'rack.session',
```

```
9                               :path => '/',
10                              :expire_after => 2592000, # In seconds
11                              :secret => 'somesuperheavysecret'
12
13 run CertificateServer
```

Durch Aufruf von *rackup config.ru -p9393 -o projekt07.priv.lab.nm.ifi.lmu.de* bzw. */etc/init.d/regserver start* wird der Server auf Port 9393 gestartet.

Listing 2.4: certificate_server.rb

```
1
2  require 'sinatra'
3  require 'user'
4  require 'sqlite3'
5
6  class CertificateServer < Sinatra::Application
7    set :public, 'public'
8
9    before do
10     if session[:id]
11       @user = User.get(session[:id])
12     end
13   end
14
15   get '/' do
16     secure
17
18     erb :home
19   end
20
21   get "/logout" do
22     session = nil
23     redirect "/anmeldung"
24   end
25
26   get "/anmeldung" do
27     @user = User.new
28     erb :anmeldung
29   end
30
31   post "/anmeldung" do
32     @user = User.new(params["user"])
33
34     if @user.save
35       session[:id] = @user.id
36       db = SQLite3::Database.new( "users.db" )
37       db.execute("insert into roles values
38 ('"+@user.name+"','manager')")
```

```
39        redirect '/'
40      else
41        erb :anmeldung
42      end
43    end
44
45    get"/userlist" do
46      content_type "text/plain"
47      attachment "userliste.txt"
48         Dir['/opt/gridshib-ca-2.0.0/certs/*'].map do
49 |filename|
50         subject = `openssl x509 -in #{filename}
51 -noout -subject`.chomp
52          "#{subject}\t#gentoo\n"
53          end
54      end
55
56    get "/cert-:user.crt" do
57      secure
58    end
59
60    def secure
61      unless @user
62        redirect "/anmeldung"
63      end
64    end
65  end
```

Hier wird die eigentliche Funktionalität des Servers festgelegt. Besonders interessant ist folgender Teil (Zeile 45 - 54):

```
get"/userlist" do
  content_type "text/plain"
  attachment "userliste.txt"
     Dir['/opt/gridshib-ca-2.0.0/certs/*'].map do |filename|
     subject = `openssl x509 -in #{filename} -noout -subject`.chomp
      "#{subject}\t#gentoo\n"
      end
  end
```

Wenn ein Nutzer /userlist anfordert, werden sämtliche Dateien im Verzeichnis /opt/gridshib-ca-2.0.0/certs nach subjects durchsucht und diese dann in eine Datei 'userliste.txt', jeweils durch einen Tabulatorschritt getrennt, in eine Zeile mit "gentoo" geschrieben. Diese Datei stellt also ein Grid-Mapfile dar, bei dem die subjects der Gridshib Zertifikate jeweils auf den lokalen Account gentoo abgebildet werden.

Listing 2.5: user.rb

```
#{- encoding: utf-8 -}

require 'dm-core'
require 'dm-validations'
require 'dm-types'

class User
  include DataMapper::Resource

  attr_writer :password_confirmation

  property :id,         Serial

  property :name,       String,     :required => true,
                                    :messages => {
                                      :presence => "Name fehlt"
                                    }

  property :matnr,      String,     :required => true, :unique
=> true,
                                    :messages => {
                                      :presence => "Matrikelnummer
fehlt",
                                      :is_unique => "Matrikelnummer
bereits vergeben"
                                    }

  property :password,   String, :required => true,
                                    :messages => {
                                      :presence => "Passwort fehlt"
                                    }

  property :created_at, DateTime

  validates_with_method :password, :method => :matchings_
passwords, :if => :new_record?
  validates_with_method :password, :method => :pw_not_
empty, :if => :new_record?

  def matchings_passwords
    [password == @password_confirmation,
      "Passwörter stimmen nicht Überein"]
  end

  def pw_not_empty
```

```
46      [password != '',
47        "Passwort ist leer"]
48    end
49  end
50
51  if __FILE__ == $0
52    ::DataMapper.setup( :default, "sqlite3://#{Dir.pwd}/users.db" )
53    User.auto_migrate!
54  end
```

Hier wird festgelegt, welche Eingaben vom Benutzer bei der Anmeldung erwartet werden und wie diese in der Datenbank users.db abgelegt werden.

2.2.3 Datenbank

Die Daten der angemeldeten Nutzer werden in der sqlite3 Datenbank users.db abgelegt. Für mehr Informationen zu sqlite3 siehe [Chi04].
Beim Erstellen der Datenbank wurde folgendes Schema verwendet:

```
CREATE TABLE "roles" ("name" VARCHAR(50) NOT NULL, "role" VARCHAR(50));
CREATE TABLE "users" ("id" INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
"name" VARCHAR(50) NOT NULL, "matnr" VARCHAR(50) NOT NULL,
"password" VARCHAR(60) NOT NULL, "created_at" TIMESTAMP);
```

Im Table "roles" wird für jeden Nutzer einfach der jeweilige Benutzername und der Wert 'manager' eingetragen (siehe certificate_server.rb). Das ist erforderlich, damit Tomcat 6 ein Realm erzeugen kann, mehr dazu im folgenden Kapitel.

3 Tomcat 6

Apache Tomcat 6.0 stellt einen Servlet-Container zur Verfügung und ermöglicht das Ausführen von Java-Code auf Webservern. Als Servlets bezeichnet man Java-Klassen, deren Instanzen innerhalb eines Java-Webservers Anfragen von Clients entgegennehmen und beantworten.

Im vorliegenden Fall ermöglicht es Tomcat dem Nutzer, sich mit den beim Registration Server hinterlegten Daten bei der Gridshib-CA einzuloggen. Realisiert wird das durch die Verwendung eines JDBC Realm. Für mehr Informationen zu Tomcat und Realms siehe [Fou10]

3.1 Realms

[Fou10] definiert den Begriff Realm wie folgt:

A Realm is a "database" of usernames and passwords that identify valid users of a web application (or set of web applications), plus an enumeration of the list of roles associated with each valid user. You can think of roles as similar to groups in Unix-like operating systems, because access to specific web application resources is granted to all users possessing a particular role (rather than enumerating the list of associated usernames). A particular user can have any number of roles associated with their username.

Standardmäßig gibt es fünf Plugins für Realms:

- JDBCRealm: Für den Zugriff auf Authentifizierungsdaten, die in einer relationalen Datenbank abgelegt sind, auf die mit einem JDBC Treiber zugegriffen wird.
- DataSourceRealm: Für den Zugriff auf Authentifizierungsdaten, die in einer relationalen Datenbank abgelegt sind, auf die über eine named JNDI JDBC DataSource zugegriffen wird.
- JNDIRealm: Für den Zugriff auf Authentifizierungsdaten, die in einem LDAP basierten Verzeichnisserver abgelegt sind, auf den über einen JNDI Provider zugegriffen wird.
- MemoryRealm: Für den Zugriff auf Authentifizierungsdaten, die in einer In-memory object collection abgelegt sind, die von einem XML-Dokument aus initiiert wird (conf/tomcat-users.xml).
- JAASRealm: Für den Zugriff auf Authentifizierungsdaten durch das Java Authentication and Authorization Service (JAAS) Framework.

Es ist ausserdem möglich, eigene Realm Implementierungen zu schreiben und diese in Tomcat zu integrieren.

3.2 Einrichtung eines JDBC Realm

Im konkreten Fall wird ein JDBCRealm verwendet. Folgende Schritte sind nötig, damit auf die Datenbank des Registration Servers (users.db) zugegriffen werden kann:

1. Zunächst muss sichergestellt werden, dass die Datenbank folgende Struktur hat (siehe 2.2.3):

Eine Table, die eine Spalte für Benutzernamen und eine weitere für je ein zugehöriges Passwort enthält.

Eine weitere Table mit einer Spalte für Benutzernamen und einer Spalte für Rollen, die der jeweilige Nutzer haben soll.

2. Einen Datenbank Benutzernamen und ein Passwort festlegen, um Tomcat Leserechte zu geben. Das ist in diesem Fall nicht nötig, da eine sqlite3 Datenbank verwendet wird.

3. Den JDBC Treiber für sqlite3 von `http://www.zentus.com/sqlitejdbc/` herunterladen und in /opt/apache-tomcat-6.0.24/lib ablegen.

4. Das Realm Element in /opt/apache-tomcat-6.0.24/conf/server.xml wie folgt definieren:

```
<!-- REALM FOR SQLITE3 DATABASE -->
      <Realm className="org.apache.catalina.realm.JDBCRealm" debug="99"
        driverName="org.sqlite.JDBC"
        connectionURL="jdbc:sqlite:/home/globus/globus/identity-provider/
users.db"
        userTable="users" userNameCol="name" userCredCol="password"
        userRoleTable="roles" roleNameCol="role"/>
```

Wobei für userTable, userNameCol, userCredCol, userRoleTable und roleNameCol die Werte entsprechend ihrer Namen in der Datenbank gesetzt werden.

5. Tomcat neu starten.

Nachdem das Realm nun erzeugt wurde, kommen wir im nächsten Kapitel zu Shibboleth, einem System für Single-Sign-On.

4 Shibboleth 2

Unter einem Shibboleth versteht man im allgemeinen eine Art linguistischen Code. Ein Shibboleth kann ein Wort sein, das von einer bestimmten Gruppe von Personen auf besondere Weise ausgesprochen wird und somit die Zuordnung eines Sprechers zu dieser Gruppe ermöglicht. Insofern ist der Name für ein Produkt, das für Authentifizierung und Autorisierung verwendet wird, passend gewählt.

Das Einsatzgebiet von Shibboleth 2 wird in der Dokumentation ([Kli09]) umfassend beschrieben, es folgt ein zusammenfassender Auszug:

Shibboleth 2 ist ein Standard basiertes open source Softwarepaket für Single-Sign-On über Unternehmensgrenzen hinweg. Es implementiert SAML (Security Assertion Markup Language), um einen Rahmen für Single-Sign-On und Attributaustausch zu schaffen. Der Nutzer und dessen Organisation haben dabei Kontrolle über die Attribute, die jeder Anwendung zur Verfügung gestellt werden. SAML ist ein XML-basiertes Framework zum Austausch von Authentifizierungs- und Autorisierungsinformationen, das von OASIS (Organization for the Advancement of Structured Information Standards) entwickelt wurde. Ein durch Shibboleth 2 geregelter Zugriff dient der Verwaltung von Identitäten und Rechten.

Shibboleth ist ein open source Projekt, das kostenlos verfügbar ist und unter der Apache Software Lizenz veröffentlicht wird.

Die folgenden Abschnitte entstammen im wesentlichen der Shibboleth 2 Dokumentation.

4.1 Wie es funktioniert

Ein Nutzer authentifiziert sich mit seinen betrieblichen bzw. organisationsspezifischen credentials. Die Organisation (oder der Identity Provider) gibt die minimal benötigten Identitätsinformationen an den Service Manager weiter, um Autorisierungsentscheidungen zu ermöglichen.

Shibboleth 2 besteht aus zwei Hauptkomponenten:
1. Identity Provider (IdP) - Die Software, die von einer Organisation genutzt wird, deren Mitglieder zugriffsbeschränkte Dienste nutzen wollen
2. Service Provider (SP) - Die Software, die vom Anbieter der zugriffsbeschränkten Dienste genutzt wird

Shibboleth 2 stützt das Identitäts- und Zugriffsverwaltungssystem einer Organisation, so dass der Status der Beziehung des Nutzers zur Institution für die Vergabe von Zugriffsrechten auf Dienste entscheidend ist.

Man kann also sagen, dass Shibboleth es dem Nutzer ermöglicht, für eine Authentifizierung und Autorisierung benötigte Informationen über die eigene Person sicher zu übertragen und dass es verwendet werden kann, um Single-Sign-On zu realisieren.

Das folgende Diagramm aus der Dokumentation zeigt die Interaktion zwischen dem Nutzer, dem Identity Provider und dem Service Provider:

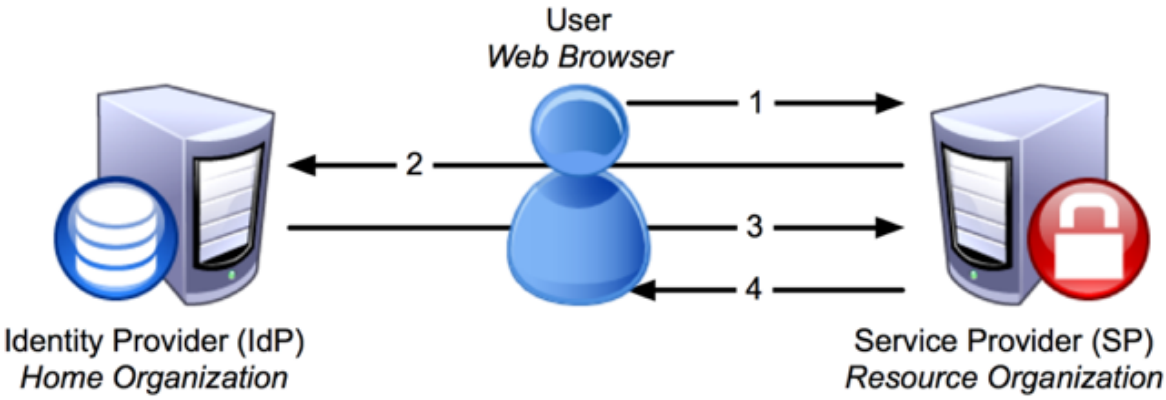

Abbildung 4.1: SSO-Flow (Quelle: spaces.internet2.edu)

1. Der SP stellt fest, dass der Nutzer versucht, auf geschützte Ressourcen zuzugreifen.
2. Der SP erzeugt ein Authentication Request und schickt es an den IdP des Nutzers.
3. Der IdP authentifiziert den Nutzer und schickt die Antwort zurück zum SP.
4. Der SP verifiziert die Antwort des IdP und schickt die Anfrage an die Ressource, die den ursprünglich angeforderten content liefert.

4.2 IdP Authentifizierung und Sessions

4.2.1 Authentifizierung

Der IdP nutzt LoginHandler, um einen Nutzer zu authentifizieren. Jeder LoginHandler ist verantwortlich dafür, einen oder mehrere Typen von Authentifizierungsprozessen zu ermöglichen. Einige solcher Möglichkeiten wären z.B. eine Kombination von Benutzername und Passwort gegen ein LDAP Verzeichnis zu prüfen, ein X.509 Zertifikat zu validieren, oder ein one-time Passwort zu prüfen. Jeder LoginHandler hat eine so genannte authentication method duration. Das ist ein Inaktivitäts-Timeout für die zugehörige Authentifizierungsmethode. Er legt die Zeit fest, für die diese Methode, nach vorheriger Authentifizierung, genutzt werden darf, um einen Nutzer per Single-Sign-On einzuloggen. Wird die Methode während dieser Zeit genutzt, so wird der Timeout zurückgesetzt.

4.2.2 Nutzer Session

Die Nutzer Session beinhaltet Informationen über einen Nutzer, z.B. welche Methode der Authentifizierung gerade aktiv ist und gegenüber welchem Service der Nutzer authentifiziert wurde. Diese Informationen werden vom IdP genutzt, um zu entscheiden, ob dem Nutzer bei einem Service Provider Single-Sign-On gewährt werden soll und um zu entscheiden, welche Dienste Logout Requests erhalten sollen und innerhalb des Attribute Resolver und Attribute Filtering Prozesses verwendet werden können. Eine Session hat einen Inaktivitäts-Timeout. Dieser Timeout wird zurück gesetzt, wann immer ein Nutzer bei einem Service Provider authentifiziert wird.

4.2.3 Zusammenhang zwischen Authentifizierung und Session

Der Zusammenhang zwischen der Authentifizierungsmethode und der Lebensdauer einer Session kann verwirrend sein. Die Session kontrolliert die Lebenszeit der Interaktion des Nutzers mit dem IdP. Das heisst: wird die Session beendet, weil sie z.B. ihren Timeout überschritten hat, ist die Authentifizierungsmethode irrelevant. Alle Methoden terminieren mit der Session.

Ein Beispiel: Der IdP hat eine Session Lifetime von 8 Stunden. Er hat zwei LoginHandler konfiguriert. Einer führt eine username / password (UP) Authentifizierung durch und hat eine Lebensdauer von 1 Stunde. Der andere führt eine X.509 Authentifizierung durch und hat eine Lebensdauer von 15 Minuten. Der Nutzer loggt sich zum Zeitpunkt 0 bei Service Provider 1 ein und authentifiziert sich beim IdP über den UP LoginHandler. Der Nutzer loggt sich bei SP 2 ein und authentifiziert sich über den X.509 LoginHandler (Abbildung 4.2).

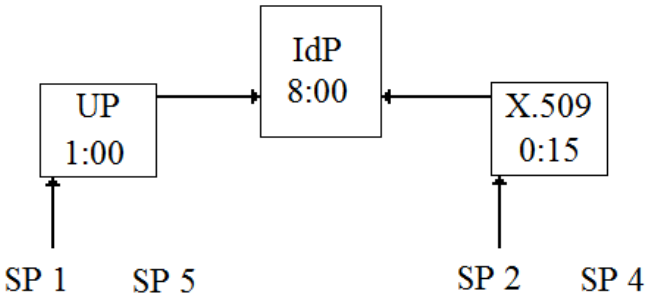

Abbildung 4.2: Zeitpunkt 0

Der Nutzer verlässt seinen Rechner und kommt 20 Minuten später zurück (Zeitpunkt 1). Jetzt will er sich bei SP 4 einloggen, wofür X.509 Authentifizierung nötig wäre. Der Nutzer wird aufgefordert, sich neu zu authentifizieren, weil die Lebenszeit der Authentifizierungsmethode abgelaufen ist (sie betrug 15 Minuten). Der Nutzer will sich bei SP 5 einloggen, wofür UP Authentifizierung benötigt wird. Er wird aufgefordert, sich neu zu authentifizieren, da die Lebenszeit dieser Methode noch nicht abgelaufen ist. Der Inactivity Timer wird zurückgesetzt auf 1 Stunde (Abbildung 4.3).

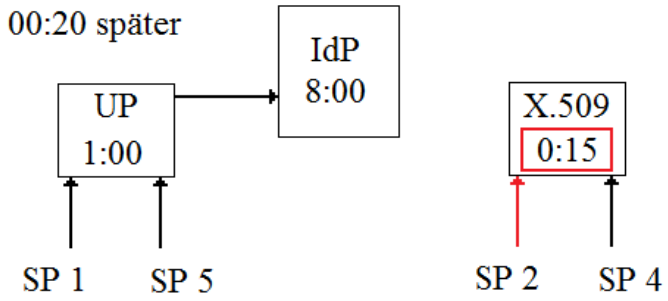

Abbildung 4.3: Zeitpunkt 1

Jetzt nehmen wir an, die UP Methode hat zum Zeitpunkt 2 eine Authentifizierungsdauer

von 10 Stunden und es ist zunächst niemand eingeloggt. Der Nutzer geht zu SP 1 und benutzt UP (Abbildung 4.4).

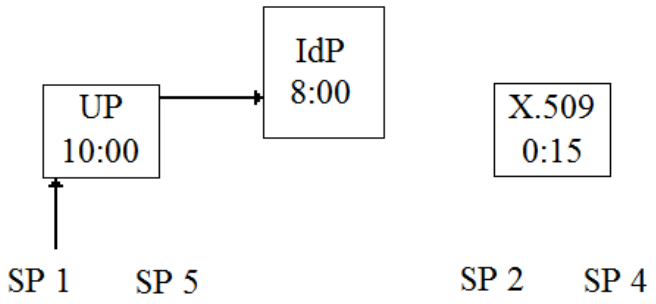

Abbildung 4.4: Zeitpunkt 2

Er verlässt den Rechner und kehrt 9 Stunden später zurück (Zeitpunkt 3). Er geht zu SP 5, der UP benötigt und wird aufgefordert, sich neu zu authentifizieren. In diesem Fall jedoch nicht etwa, weil die Lebensdauer der Authentifizierungsmethode abgelaufen ist, sondern weil die Session wegen Inaktivität beendet wurde und damit auch alle Authentifizierungsmethoden (Abbildung 4.5).

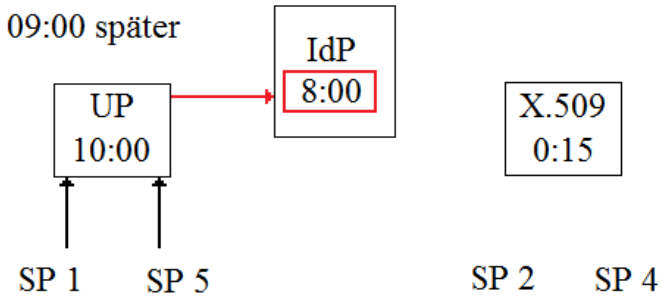

Abbildung 4.5: Zeitpunkt 3

4.3 Ablauf beim Zugriff auf geschützte Ressourcen

Shibboleth besteht im Wesentlichen aus zwei Komponenten: einem Identity Provider (IdP) und einem Service Provider (SP). Services bekommen Informationen über Nutzer vom Identity Provider. Diese Informationen werden vom Service Provider gesammelt, der dann den Zugriff auf geschützte Ressourcen ermöglicht.

4.3.1 Nutzer will auf geschützte Ressource zugreifen

Will ein Nutzer auf eine geschützte Ressource zugreifen, so fängt der Service Provider ihn zunächst ab. Die zu schützenden Ressourcen können in der httpd.conf oder in shibboleth2.xml festgelegt werden.

Ausgehend von dieser Konfiguration wählt der SP dann einen SessionInitiator, welcher den IdP ermittelt, an den der Nutzer weitergeleitet wird.

Der SessionInitiator wird benutzt, um Handler zu konfigurieren, die dafür verantwortlich sind, den Authentifizierungsvorgang gegenüber dem Service Provider zu initiieren und eine Session mit diesem zu eröffnen. Das wird als SP-iniitierter Ablauf bezeichnet, wobei ein Browser, der am Anwendungsende startet, an den Identity Provider weiter geleitet werden muss, um sich einzuloggen und anschließend mit den benötigten Login-Informationen zurückkehrt.

Die Einzelheiten des Authentication Request Prozesses variieren und die interne SessionInitiator API erlaubt es, den Service Provider mit vielen request Protokollen zu erweitern. Der Handler ist verantwortlich für sämtliche Details, die mit dem Stellen des Requests verbunden sind.

Eine Besonderheit dieser Art von Handler ist, dass sie oft implizit als Resultat des ersten Zugriffs auf eine geschützte Ressource laufen (z.B. wenn die requireSession content Einstellung aktiv ist). Für andere Arten von Anwendungsfällen, wie z.B. das "passive/lazy session" Feature, welches es einer Anwendung ermöglicht, das Erstellen einer Session zu verzögern, ist ein einfaches und erweiterbares Protokoll lokal in der Service Provider Software implementiert. Dieses Protokoll ermöglicht es Anwendungen, diese Handler zu starten, indem sie ein gewöhnliches HTTP redirect mit einem query string nutzen.

Die Möglichkeit, mehrere SesseionInitiator Handler zu konfigurieren und diese zu Ketten zu kombinieren ermöglicht es dem Entwickler, die Auswahl bestimmter SSO Protokolle zu kontrollieren, wenn mehr als eines verwendet werden kann. Dadurch können verschiedene Varianten von IdP Discovery implementiert werden.

Auf den genauen Ablauf der IdP Discovery wird hier nicht näher eingegangen, siehe hierzu `https://spaces.internet2.edu/display/SHIB2/IdPDiscovery`

4.3.2 Nutzer authentifiziert sich beim Identity Provider

Der Service Provider schickt einen Authentication Request an den Identity Provider. Dieser Request wird über den Browser verschickt und der Nutzer wird zu einem Endpoint beim IdP weitergeleitet, der als Single-Sign-On Service bezeichnet wird. Der IdP hat nun mehrere Möglichkeiten, den Nutzer zu authentifizieren. Diese Möglichkeiten sind:

1. Remote User: Dieser Login Handler ist im Grunde ein Java Servlet. Es handelt sich um ein Stück Java Code, das über eine URL adressiert wird. Dieser Code sucht nach einem Wert im REMOTE_USER Header und benutzt diesen als Benutzernamen für den Nutzer des Browsers. Die Grundannahme ist, dass dem Benutzer vertraut werden kann, wenn der Container (web server oder servlet container) es festgelegt hat. Daher muss für die Verwendung dieses Handlers sichergestellt sein, dass die URL des remote user servlet mit einem Authentifizierungsmechanismus innerhalb des Containers geschützt ist.

2. Username / Password: Der User wird auf eine Authentifizierungsseite geleitet. Die eingegebene Kombination von Benutzername und Passwort wird dann gegen ein LDAP Verzeichnis oder eine Kerberos 5 Domäne geprüft.

3. IP Adress: Vergleicht die IP Adresse des Nutzers gegen eine Liste von Adessen, um ihn zu identifizieren.

4. Previous Session: Dieser Handler kommt zum Einsatz, wenn eine existierende Session des Nutzers genutzt wird, um diesen zu authentifizieren (z.B. wenn der Identity Provider

einen SSO basierten sign-on durchführt).

Der IdP wählt eine dieser Möglichkeiten entsprechend den Regeln aus relying-party.xml und des definierten LoginHandler.

Im konkreten Fall wird Remote User Authentication gewählt, um die containterbasierte Authentifizierung zu ermöglichen. Siehe hierzu auch Kapitel 3: Tomcat 6.

4.3.3 Der IdP schickt seine Antwort an den Service Provider

Der IdP muss nun entscheiden, welche Information an den SP geschickt werden soll und wie diese zu verpacken ist. Zunächst sammelt der IdP mit dem Attribute Resolver ein Set von Attributen für den Nutzer. Er sammelt Nutzerdaten von allen unterstützten Quellen, transformiert diese wenn nötig und fügt jedem Attribut Kodierer hinzu.

Diese Attribute werden durch den Attribute Filter geschickt, welcher wie der Name vermuten lässt die Information für die Antwort reduziert. Dafür verwendet der Filter die in der attribute-filter.xml definierten Policies. Das Set freigegebener Attribute hängt meist vom SP ab. Die resultierende Information kann so wenig sein wie "jemand hat sich erfolgreich authentifiziert", oder jedes vorstellbare Attribut enthalten.

Die Information wird unter Zuhilfenahme der zuvor angehängten Kodierer in eine Form verpackt, die für die ausstehende Antwort geeignet ist, für gewöhnlich in eine SAML Assertion. Diese Assertion kann mit dem Key des IdP signiert sein und, im Falle einer SAML 2.0 Assertion, mit dem Key des SP verschlüsselt sein, um Sicherheit und Privatsphäre zu gewährleisten. Die Assertion (oder die als Artifact bezeichnete Referenz darauf) wird in eine Antwort gepackt, die durch den Browser zurück zum SP geschickt wird, zu einem Endpunkt, der als Assertion Consumer Service bezeichnet wird.

4.3.4 Service Provider erzeugt neue Session

Der Browser liefert die Antwort vom IdP an einen Assertion Consumer Service (ACS) Endpunkt beim SP. Die ACS Implementierung dekodiert die Nachricht, entschlüsselt die Assertion wenn nötig und führt einige Sicherheitsprüfungen durch. Wenn diese bestanden wurden, wird der SP eine neue Session erzeugen, nachdem er die Attribute und sonstigen Informationen aus der Nachricht extrahiert hat. Attribute werden unter Verwendung des Attribute Extractor des SP in eine passende Form übersetzt, durch einen Attribute Filter geschickt und in der neuen Session zusammen mit anderer relevanter Information gecached.

Der Nutzer wird dann wieder zur geschützten Ressource geleitet, auf die er nun innerhalb der neuen Session zugreifen kann.

5 Gridshib-CA 2

Im Praktikum erlaubt es GridShib-CA 2 den Teilnehmern, sich über Shibboleth zu authentifizieren. Nach dieser Authentifizierung wird ein kurzlebiges Zertifikat als Grid Credential für den Nutzer erzeugt, welches Zugriff auf Grid Ressourcen ermöglicht und z.B. mit dem Globus Toolkit 4 verwendet werden kann, auf das im folgenden Kapitel eingegangen wird.

GridShib-CA ist ein Softwareprodukt des CILogon Projekts. Das folgende Diagramm zeigt den standardmäßigen Workflow der GridShib-CA und entstammt der Dokumentation unter `http://gridshib.globus.org/docs/gridshib-ca-0.5.2/admin/process-flow.html`:

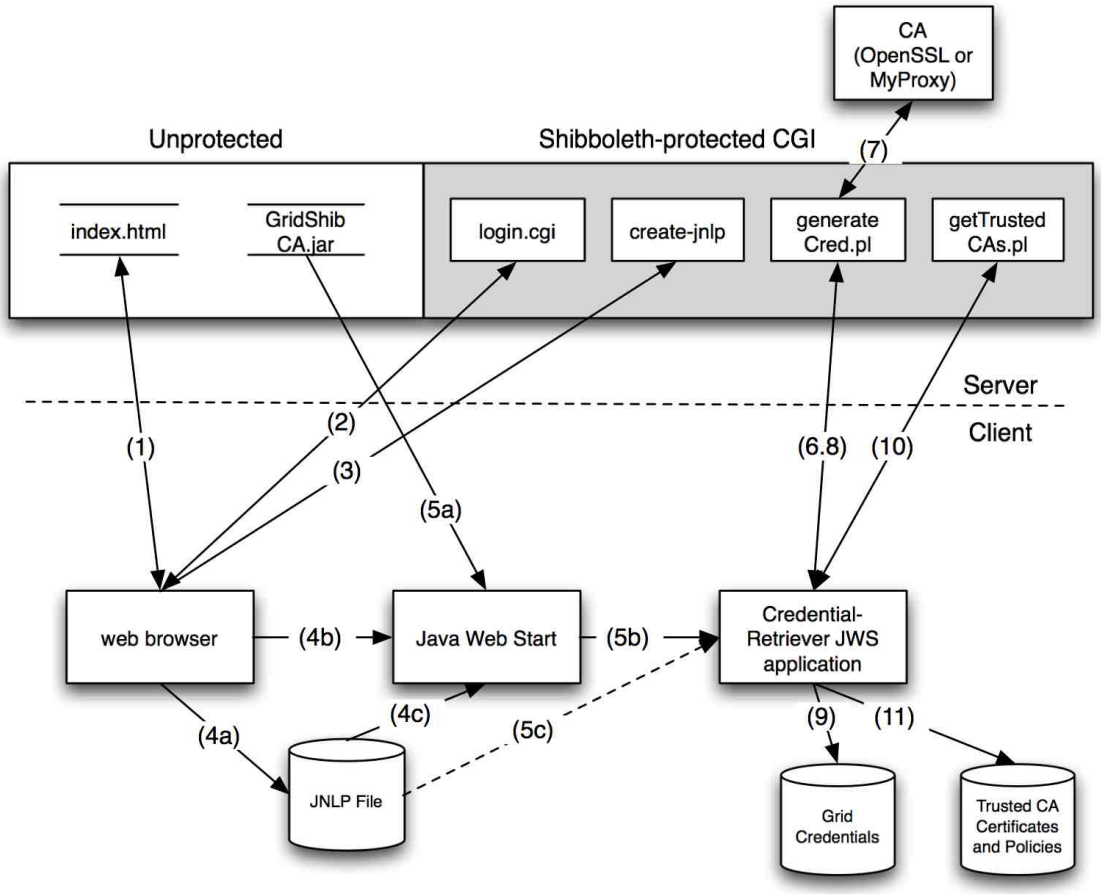

Abbildung 5.1: gridshib-ca-process-flow (Quelle: gridshib.globus.org)

Die folgenden Schritte beschreiben die normale Verwendung der GridShib-CA durch einen

Nutzer:

1. Der Nutzer greift mit einem Browser auf index.html zu. Durch Klicken auf den Submit Button wird er zum logon weitergeleitet.

2. Durch Klicken auf den Logon Button wird der Nutzer zu login.cgi geleitet. Dies löst die Shibboleth Authentifizierung aus. Die Hauptaufgabe von login.cgi ist es sicherzustellen, dass die Shibboleth Identität des Nutzers zugreifbar und valide ist. Der Nutzer wird aufgefordert, auf einen Submit Button zu klicken, um seine Credentials zu erhalten, was das create-jnlp CGI Skript startet. Zusätzlich setzt login.cgi einen Cookie und ein Element zum Schutz vor Cross-Site Request Forgery (CSRF).

Eine Cross-Site Request Forgery (zu deutsch etwa "Site-übergreifende Aufruf-Manipulation", meist XSRF oder CSRF abgekürzt) ist ein Angriff auf ein Computersystem, bei dem der Angreifer unberechtigt Daten in einer Webanwendung verändert. Er bedient sich dazu eines Opfers, das ein berechtigter Benutzer der Webanwendung sein muss. Mit technischen Maßnahmen (oder einfacher Täuschung des Opfers) wird hierzu aus dem Webbrowser des Opfers ohne dessen Wissen und Einverständnis ein kompromittierter HTTP-Request an die Webanwendung abgesetzt. Der Angreifer wählt den Request so, dass bei dessen Aufruf die Webanwendung die vom Angreifer gewünschte Aktion ausführt. (Quelle: Wikipedia)

3. Das create-jnlp Skript prüft das CSRF Element und den Cookie, die von login.cgi erzeugt wurden und stellt sicher, dass diese zusammenpassen. Dann erzeugt es das JNLP (Java Network Launching Protocol) file um die Java Web Start Anwendung zu starten. Das JNLP file enthält folgende Argumente für die CredentialRetrieve Java Anwendung:

3.a.) Die URL für das generateCred.pl Skript (Schritt 6)

3.b.) Den Shibboleth Session Cookie des Nutzers

3.c.) Ein verschlüsselter Token, der es dem CredentialRetriever erlaubt sicherzustellen, dass er rechtmässig gestartet wurde

3.d.) Anweisung, ob der client im Debug Modus ausgeführt werden soll oder nicht

3.e.) Die Lebensdauer, die der Nutzer für das Credential fordert

3.f.) Die URL, von der der Client vertrauenswürdige CAs bekommt (Schritt 10)

3.g.) Optional einen Parameter, der dem Client vorschreibt, seine gesammelten CAs nicht zu verwenden, um seine https Verbindungen zu validieren (Schritte 2, 3, 6 und 10).
Siehe hierzu auch Abbildung 5.2: Erzeugung und Inhalt des JNLP File.

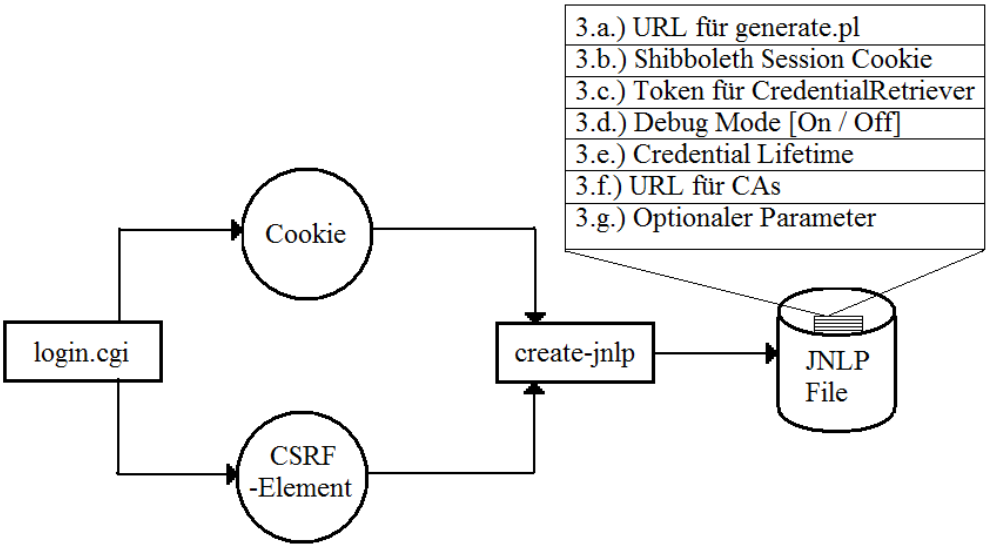

Abbildung 5.2: Erzeugung und Inhalt des JNLP File

4. Der Browser speichert das JNLP File und startet Java Web Start, wobei er das JNLP File weitergibt

5. Java Web Start lädt das GridShib jar file entsprechend dem JNLP File herunter und startet die CredentialRetriever Klasse aus dem jar. Dabei gibt es die Shibboleth Session, den Token und die URL aus dem JNLP File als Argumente weiter.

6. CredentialRetriever läuft, erzeugt ein Schlüsselpaar und ein Certificate Request. CredentialRetriever nutzt die Shibboleth Session, um sich gegenüber generateCred.pl zu authentifizieren, als wäre es der Browser des Nutzers. Es gibt folgendes an generateCred.pl weiter:
6.a.) certificateRequest: Das Certificate Request als PEM kodierten PKCS10.
6.b.) token: Das verschlüsselte Token, das von login.cgi erzeugt wurde.

7. generateCred.pl validiert das Token und gibt dann dem certificate request eine CA (entweder lokal OpenSSL basiert oder MyProxy basiert) zum signieren. Im Gegenzug erhält es das signierte Zertifikat.

8. generateCred.pl gibt das Zertifikat an CredentialRetriever zurück.

9. CredentialRetriever schreibt das Zertifikat und den privaten Schlüssel in das Verzeichnis, das vom Globus Toolkit erwartet wird.

10. Ist der Server konfiguriert, um den Client mit Zertifikaten vertrauenswürdiger CAs zu versorgen, verbindet der Client sich zum getTrustedCAs.cgi Skript und führt ein parameterloses GET aus. Das Skript liefert eine Liste von Dateien (CA Zertifikate und signing policies), die der Client installieren kann, unter Verwendung des folgenden Formats, wobei jede Datei

durch eine Zeile separiert ist, die den Dateinamen und einen festgelegten Prefix enthält:

```
-----File: aa99c057.0
...certificate data...
-----File: aa99c057.signing_policy
...signing policy data...
...and so forth...
```

11. Der Client schreibt die erhaltenen CA Dateien und Policies in sein Trusted Certificates Verzeichnis.

6 Globus Toolkit 4

Zentrale Anlaufstelle für Fragen rund um das Globus Toolkit ist die Globus Homepage `www.globus.org`.

Das Globus Toolkit wird dort beschrieben als ein Framework, mit dem sich ein Grid aufbauen lässt, der es seinen Teilnehmern ermöglicht, Rechenleistung, Datenbanken und andere Ressourcen sicher online zu teilen. Dabei spielt es grundsätzlich keine Rolle, wo auf der Welt sich diese Ressourcen befinden und zu welchen Organisationen sie gehören, solange diese Teil des Grids sind.

Das Toolkit stellt Mittel zur Verfügung, solche Ressourcen zu finden, zu verwalten und zu sichern. Führende IT Unternehmen setzten das Globus Toolkit ein, um kommerzielle Grids aufzubauen. Es besteht aus einem Paket von Komponenten, die von einander unabhängig oder zusammen benutzt werden können, um Anwendungen für den Grid zu entwickeln.

Einer Zusammenarbeit verschiedener Organisationen steht oft die Inkompatibilität von Ressourcen wie Datenbanken, Rechnern und Netzwerken im Wege. Diese Hindernisse zu beseitigen und eine reibungslose Zusammenarbeit zu ermöglichen ist das Ziel bei der Entwicklung des Toolkits. Es erlaubt es seinen Nutzern, auf entfernte Ressourcen zuzugreifen, als befänden sie sich im selben Raum, wobei die lokale Kontrolle darüber, wer und wann auf eigene Ressourcen zugreifen kann stets erhalten bleibt.

Das Globus Toolkit ist ein open source Projekt.

Als Grundlage für dieses Kapitel dienen zwei Paper von Ian Foster über das Globus Toolkit ([Fos05], [Fos06]), aus denen die folgenden Abschnitte weitgehend entliehen sind. Das Globus Toolkit besteht aus vielen Komponenten, die hier nicht alle näher beschrieben werden.

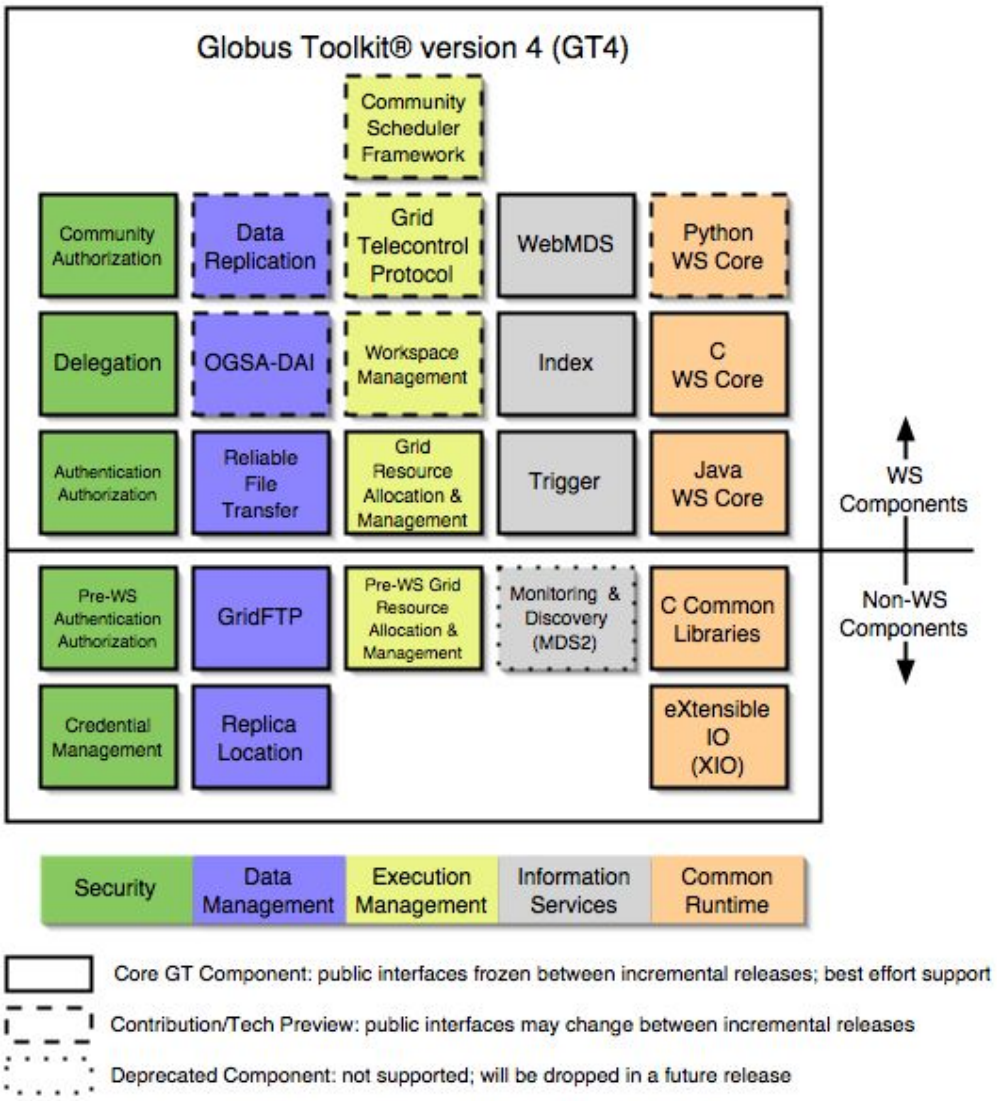

Abbildung 6.1: Komponenten des Globus Toolkit 4 (Quelle: globus.org)

6.1 Execution Management

Execution management Tools sind zuständig für Initiierung, Monitoring, Verwaltung, Scheduling und/oder Koordination verteilter Prozesse. GT4 unterstützt zu diesem Zweck das Grid Resource Allocation and Management (GRAM) Interface.

Das Globus Toolkit stellt Web Services zur Verfügung, mit denen Clients interagieren können, um Jobs auf lokalen oder entfernten Ressourcen auszuführen, zu überprüfen und zu beenden. Dieses System wird GRAM genannt, manchmal auch WS-GRAM, um es vom pre-Web Services System abzuheben, das eine ähnliche Funktionalität bietet.

Entfernten Nutzern soll es erlaubt werden, ein Programm auszuführen. Eine Möglichkeit besteht darin, dieses Programm als Web Service zugreifbar zu machen. Um das zu ermöglichen wird eine Web Service Operation implementiert, die besagtes Programm ausführt und

das Ergebnis an den Client zurück liefert. In vielen Fällen ist diese einfache Lösung geeignet, aber manchmal gibt es zusätzliche Anforderungen, die damit nicht erfüllt werden.

- Zustand. Der auszuführende Job kann während der Laufzeit input/output Operationen durchführen, die den Zustand der Ressource, auf der er läuft, oder deren Dateisystem beeinflussen. Daher ist es wichtig, dass ein Nutzer ein Request nicht einfach neu "submitten" kann, wenn er keine Bestätigung erhalten hat, da der Job vielleicht ausgeführt wurde und die Antwort verloren ging.
- User executables. Es kann sinnvoll sein, dem Nutzer zu erlauben, die Programme selbst zur Verfügung zu stellen, die ausgeführt werden sollen.
- Staging von input und output. Input und output Daten können groß und verteilt sein, es ist wichtig, diese Daten zwischenspeichern zu können.
- Streaming output. Manche Nutzer, vor allem interaktive, wollen Zugriff auf Output Daten während ein Job noch läuft.
- Kontrolle. Ein Nutzer benötigt unter Umständen die Möglichkeit, einen Job zu beenden, der viele Ressourcen belegt.
- Scheduler. Große Rechenressourcen arbeiten typischerweise unter der Kontrolle eines Schedulers, der Allokation und Priorisierung von Policies implementiert, während er die Ausführung aller Jobs für Effizienz und Performanz optimiert.
- Monitoring. Data staging und Scheduler offenbaren das Potential für komplexere Job Zustandsübergänge: Ein Job kann nicht einfach nur laufen, fertig oder fehlgeschlagen sein, sondern auch wartend, suspended etc.. Nutzer müssen in der Lage sein, den Zustand eines Jobs in Erfahrung zu bringen.

GRAM wurde entwickelt, um diesen Anforderungen gerecht zu werden. Wenn diese zusätzlichen Fähigkeiten nicht benötigt werden, da eine Anwendung nur überschaubare Mengen an Input und Output generiert, zustandslos operiert und oft ausgeführt wird, so kann evtl. auf GRAM verzichtet werden.

Das GRAM Interface begegnet den Anforderungen nach erweiterter Managemenentfunktionalität, indem es für jede erfolgreiche Job Submission eine zustandsbehaftete Entität, einen ManagedJob, auf der ausführenden Maschine erzeugt. Die Lebensdauer dieses ManagedJob entspricht der des zugehörigen Jobs bzw. ist etwas länger, damit der Client einen Jobzustand abfragen kann, nachdem der Job beendet ist.

Eine erfolgreiche GRAM Submit Operation liefert eine WS-Adressing Endpoint Reference (EPR) für den neuen ManagedJob zurück. Ein Client kann diese verwenden, um den Jobzustand zu erfragen, den Job zu beenden und / oder sich an den Job "anzuhängen", um bei Änderungen des Jobzustands oder output benachrichtigt zu werden. Der Client kann die EPR auch an andere Clients weitergeben, die damit dieselben Möglichkeiten haben, wenn sie dazu autorisiert sind. Das ManagedJob Konstrukt erlaubt es GRAM, viele der fortgeschrittenen Funktionen zu bieten, die im vorangegangen Abschnitt beschrieben wurden.

GRAM benötigt Informationen um zu entscheiden, ob ein Request von einem Nutzer akzeptiert wird und als welcher lokaler Nutzer dieses ausgeführt werden soll. Standardmäßig sucht GRAM hierzu das Grid-Mapfile unter /etc/grid-security. Dieses Mapfile bildet DNs auf lokale Nutzer ab, ein möglicher Eintrag wäre z.B.

"/O=Grid/OU=GlobusTest/OU=simpleCA-foo.bar.com/OU=bar.com/CN=John" john

6.2 Datentransfer

Globus Anwendungen müssen oft mit großen Datenmengen zurechtkommen, die sich an einem oder an verschiedenen Orten befinden können. Das ist ein komplexes Problem, dem keine einzelne Software ganz gerecht werden kann. Verschiedene Komponenten des Globus Toolkit 4 implementieren sinnvolle Mechanismen, die individuell oder gemeinsam genutzt werden können, um diesem Problem zu begegnen.

Die Globus Implementierung von GridFTP bietet Libraries und Tools für zuverlässigen, sicheren und hoch performanten Speicher zu Speicher und Disk zu Disk Datentransfer. Es hat laut globus.org 27 Gbit/s Ende zu Ende Verbindungen in weiträumigen Netzwerken erreicht und kann mit konventionellen FTP Clients und Servern interagieren. Viele Tools auf höheren Ebenen bauen auf den Leistugen von GridFTP auf.

Der Reliable File Transfer (RFT) Dienst sorgt für die zuverlässige Verwaltung von multiplen GridFTP Übertragungen. GRAM Dienste geben Data Staging Operationen an den RFT Dienst weiter. Das hat zur Folge, dass Job Submissions, die kein Staging benötigen keine Data Management Kosten verursachen. Wenn der RFT Dienst eine Data Staging Request erhält, initiiert er einen GridFTP Transfer zwischen der spezifizierten Quelle und dem Ziel.

Der Replica Location Service (RLS) ist ein dezentralisierter Dienst, der für die Wartung und den Zugriff auf Informationen über den Speicherort von replizierten Daten zuständig ist.

Der Data Replication Service (DRS) kombiniert RLS und GridFTP für das Management von Datenreplikation.

Data Access and Integration (OGSA DAI) Tools ermöglichen Zugriff auf und serverseitige Verarbeitung von relationalen und XML Daten.

6.3 Monitoring und Discovery

Monitoring und Discovery Dienste kümmern sich um die Beschaffung, Verteilung, Indexierung, Archivierung und sonstige Verarbeitung von Informationen über Konfiguration und Zustand von Diensten und Ressourcen. In manchen Fällen ist die Motivation dahinter, die Auffindung von Diensten und Ressourcen zu ermöglichen, in anderen das Monitoring des System Status zu ermöglichen.

Die Monitoring and Discovery System (MDS4) Komponente des Globus Toolkit 4 übernimmt die Aufgabe, Dienste und Ressourcen in verteilten Systemen zu finden und zu überwachen.

Monitoring bedeutet, Ressourcen und Dienste zu beobachten (z.B. Computer und Scheduler), zu solchen Zwecken wie dem Lösen von Problemen und der Rückverfolgung vom Gebrauch dieser Ressourcen und Dienste. Ein Nutzer kann ein Monitoring System benutzen, um Ressourcen zu indentifizieren, denen der Speicherplatz ausgeht, um entsprechende Schritte unternehmen zu können.

Discovery ist der Prozess des Findens einer passenden Ressource für die Ausführung einer Aufgabe, z.B. das Finden eines Rechenhosts, auf dem ein Job laufen kann. Dieser Prozess beinhaltet sowohl das Finden geeigneter Kandidaten wie auch die Auswahl eines passenden Kandidaten aus diesem Set.

Sowohl Monitoring als auch Discovery Anwendungen benötigen die Fähigkeit, Informatio-

nen von vielen, möglicherweise verteilten Quellen zu sammeln. Um dem zu entsprechen bietet MDS4 sogenannte Aggregator Services, die aktuelle Statusinformationen von registrierten Informationsquellen sammeln und browserbasierte Interfaces, Kommandozeilentools und Web Service Interfaces, die es Nutzern erlauben, auf gesammelte Informationen zuzugreifen.

MDS4 bietet drei verschiedene Aggregator Dienste mit verschiedenen Interfaces und Verhalten (obwohl alle auf einem gemeinsamen Rahmen aufbauen):

- MDS-Index für Xpath queries auf den neusten Werten von den Informationsquellen,
- MDS-Trigger um vom Nutzer spezifizierte Aktionen durchzuführen (wie z.B. eine mail schicken oder einen Eintrag ins Logfile setzen), wenn die gesammelten Informationen den festgelgten Kriterien entsprechen und
- MDS-Archiver um Information Source Werte in einer persistenten Datenbank zu speichern, auf die ein Client zugreifen kann, um alte Informationen anzufragen.

Der Schlüssel zum Verständnis von MDS4 ist das Aggregator-Information Source Framework. Die Grundlagen sind folgende:

- Informationsquellen sind explizit bei einem Aggreagtordienst registriert.
- Registrierungen haben eine begrenzte Lebensdauer. Werden sie nicht regelmäßig erneuert, laufen sie aus.
- Der Aggregator sammelt regelmäßig aktuelle Zustands- und Statusinformationen von allen registrierten Quellen.
- Der Aggregator macht dann alle gesammelten Informationen verfügbar über ein spezifisches Web Services Interface.

Nachdem die wichtigsten Komponenten des Identitätsmanagementsystems nun alle vorgestellt wurden, wird im nächsten Kapitel dargestellt, wie diese zusammenarbeiten.

7 Zusammenhang der Komponenten

Nachdem nun die einzelnen Komponenten des Systems vorgestellt wurden, wird in diesem Kapitel gezeigt, wie diese zusammen arbeiten, um die Verwaltung von Identitäten und Rechten im Grid Praktikum zu ermöglichen. Dazu wird zunächst der gesamte Ablauf beschrieben, der sich ergibt, wenn ein neuer Nutzer ein Zertifikat für das Grid von der GridShib-CA anfordert. Anschließend werden die interessantesten Schritte dieses Ablaufs im einzelnen näher erläutert.

7.1 Ablauf: Von der Anmeldung zum Zertifikat

Im Folgenden soll geschildert werden, was passiert, wenn ein neuer Nutzer sich anmelden möchte und welche Rolle die einzelnen Softwarekomponenten dabei spielen, auf die in den vorherigen Kapiteln eingegangen wurde. Insbesondere das Zusammenspiel des Shibboleth Service Providers mit dem Shibboleth Identity Provider soll hier noch einmal deutlich werden. Bei der Anmeldung eines neuen Nutzers ergibt sich folgender Ablauf (Abbildung 7.1):

1. Der Nutzer registriert sich beim Anmeldeserver, dieser läuft auf projekt07.priv.lab.nm.ifi.lmu.de auf Port 9393. Er bekommt dort ein Anmeldeformular präsentiert. Nachdem er dieses ausgefüllt hat, werden seine Daten in der Datenbank des Servers (users.db) abgelegt. Zusätzlich wird ihm in dieser Datenbank eine Rolle zugewiesen, die später von Tomcat für die Erzeugung des Realms benötigt wird.

2. Der Nutzer klickt auf den Shibboleth-Link der HTML-Seite unter
https://projekt07.priv.lab.nm.ifi.lmu.de :8443/gridshib-ca-2.0.0/

3. Durch Klick auf den Link wird er auf ein mittels Shibboleth gesichertes Verzeichnis verwiesen.

4. Der Shibboleth Service Provider schickt ein Authentication Request an den Shibboleth Identity Provider und leitet den Nutzer automatisch auf die Anmeldeseite des IdP weiter.

5. Der Shibboleth Identity Provider wird nun versuchen, den Nutzer zu authentifizieren. Dabei wird der Remote User Login Handler für containerbasierte Authentifizierung verwendet. Mit Hilfe von Tomcat kann der Shibboleth Identity Provider auf die Datenbank des Anmeldeservers zugreifen (users.db). Dadurch kann der Nutzer authentifiziert werden, falls er sich zuvor beim Anmeldeserver registriert hat.

6. Nach erfolgreicher Authentifizierung beim IdP wird der Nutzer wieder auf die Seite von Gridshib-CA geleitet. Er ist nun innerhalb der Session eingeloggt und hat Zugriff auf durch den SP geschützte Ressourcen.

7. Unter Zuhilfenahme von JWS erhält er schließlich das generierte Proxy-Zertifikat von der Gridshib-CA.

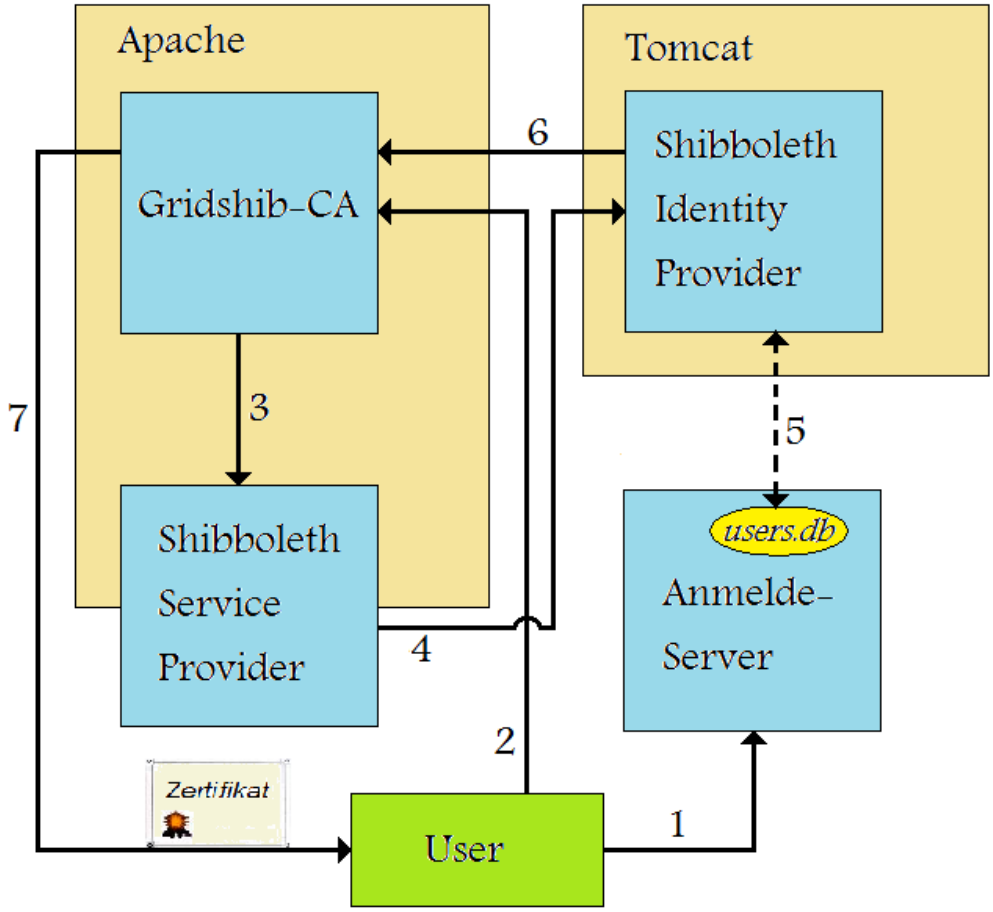

Abbildung 7.1: Ablauf

Beim Anmeldserver kann nun mittels http GET (projekt07.priv.lab.nm.ifi.lmu.de:9393/userlist) das Grid-Mapfile angefordert werden. Dieses Mapfile ist Teil des Globus Toolkit 4 und bildet die Subjects aus den von der Gridshib-CA generierten Zertifikaten auf lokale Accounts ab.

7.2 Anmeldung beim Anmeldeserver

In Schritt 1 legt der Anmeldeserver die Anmeldedaten jedes Studenten in einer sqlite3 Datenbank ab. Jedem Nutzer wird, zusätzlich zu den von ihm angegeben Daten, automatisch eine Rolle zugeordnet. Das ist erforderlich, damit von Tomcat eine Realm gebildet werden kann. Die Rollen dienen dazu, festzulegen auf welche Ressourcen die Nutzer später Zugriff haben. Ein Nutzer kann beliebig viele Rollen haben, in diesem Fall bekommt aber jeder nur eine zugewiesen, nämlich 'manager' (Abbildung 7.2).

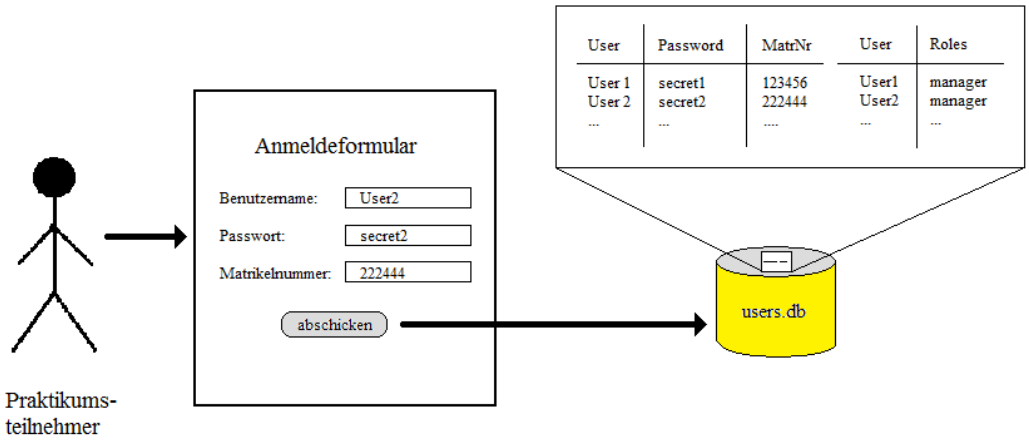

Abbildung 7.2: Anmeldung beim Anmeldeserver

7.3 Zusammenarbeit von Shibboleth Service Provider und Identity Provider

In den Schritten 3 bis 6 ist das typische Zusammenspiel von Shibboleth Identity Provider und Shibboleth Service Provider zu erkennen. Der Service Provider überwacht den Zugriff auf die geschützte Ressource, in diesem Fall ist das die GridShib-CA (Abbildung 7.3).

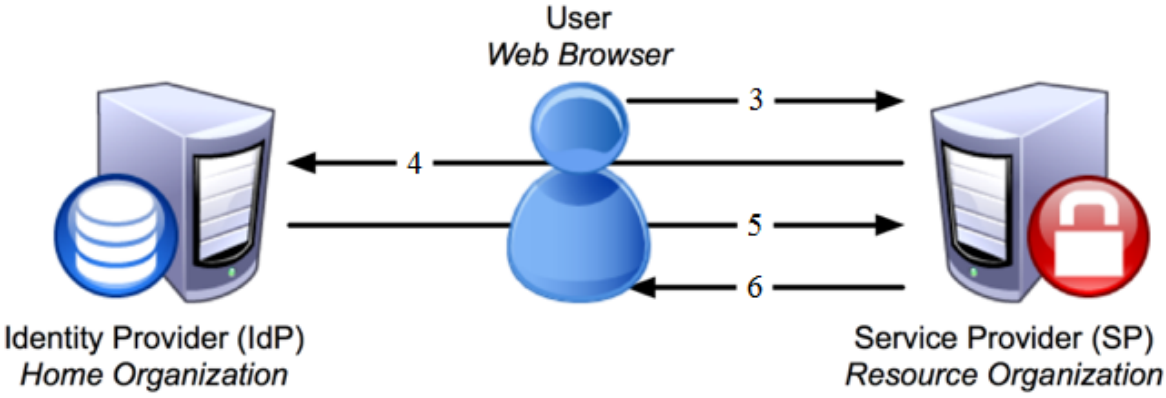

Abbildung 7.3: SSO-Flow (Quelle: spaces.internet2.edu)

3. Der SP stellt fest, dass der Nutzer versucht, auf die GridShib-CA zuzugreifen.
4. Der SP erzeugt eine Authentication Request und schickt diese an den IdP des Nutzers.
5. Der IdP authentifiziert den Nutzer mit Hilfe von Tomcat und der Datenbank des Anmeldeservers und schickt die Antwort zurück zum SP.
6. Der SP verifiziert die Antwort des IdP und gibt den Zugriff auf die GridShib-CA frei.

7.4 Authentifizierung des Nutzers

Die eigentliche Authentifizierung findet in Schritt 5 statt. Abbildung 7.4 zeigt, wie diese von statten geht. Der Shibboleth Service Provider schickt eine Authentication Request an den Identity Provider. Shibboleth 2 ist in diesem System für containerbasierte Authentifizierung konfiguriert, also wird der IdP nach einem Wert im REMOTE_USER Header suchen und diesen als Benutzernamen für den Nutzer am Browser verwenden. Hierfür ist der Inhalt der Datenbank des Anmeldesevers (users.db) entscheidend, da Tomcat 6 mit dieser Datenbank das Realm bildet, das alle registrierten Nutzer und deren Rollen enthält.

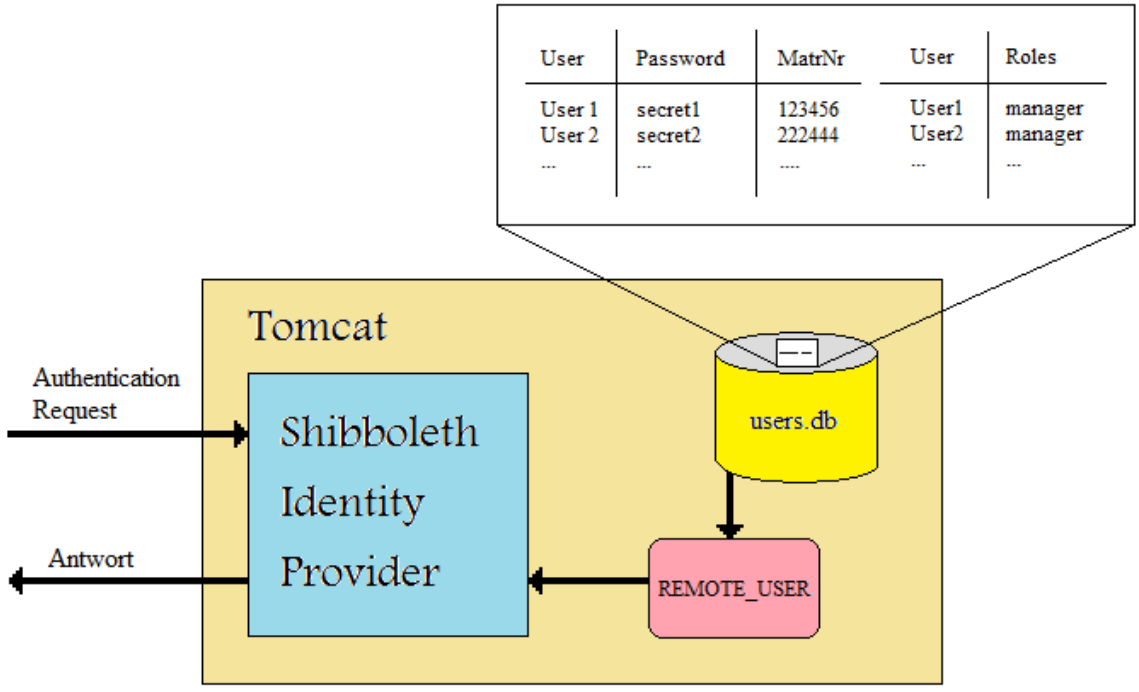

Abbildung 7.4: Authentifizierung durch den IdP

Für die Authentifizierung gegenüber Shibboleth wird ein JDBCRealm verwendet. Dabei handelt es sich um eine Implementrierung des Tomcat 6 Realm Interface, das mittels eines JDBC Treibers Nutzer aus einer relationalen Datenbank ausliest, in diesem Fall aus der sqlite3 Datenbank des Anmeldeservers. Bei der Verwendung von JDBCRealms ist folgendes zu beachten:

- Wenn ein Nutzer zum ersten Mal versucht, auf eine geschützte Ressource zuzugreifen, wird Tomcat die authenticate() Methode dieser Realm aufrufen. Daher werden alle Änderungen, die direkt an der Datenbank vorgenommen wurden (neue Nutzer, geänderte Passwörter oder Rollen, etc.) sofort berücksichtigt.

- Ist ein Nutzer einmal authentifiziert, so wird er, zusammen mit seinen Rollen, innerhalb Tomcats für die Dauer seines login im Cache behalten. Der im Cache gehaltene Nutzer wird nicht gespeichert und in einer neuen Session wiederhergestellt. Jede Änderung in der Datenbank wird für einen bereits authentifizierten Nutzer erst relevant, wenn er sich das nächste mal einloggt.

- Die Daten in den User und Roles Tables zu verwalten ist Aufgabe eigener Anwendungen, Tomcat bietet hierzu keine Möglichkeiten.

8 Zusammenfassung

An der Realisierung des Identitätsmanagements für das Grid Computing Praktikum waren verschiedene Softwarekomponeten beteiligt, die jeweils für sich genommen schon komplex sind. Diese Komponenten miteinander zu verbinden war nicht immer einfach, aber in jedem Fall lehrreich. Die LoginHandler vom Shibboleth bieten interessante Möglichkeiten, single sign-on Lösungen zu implementieren.

Die Möglichkeiten von Shibboleth und dem Globus Toolkit gehen weit über das hinaus, was in diesem Bericht Platz hatte und aufgrund der wachsenden Bedeutung von Grids werden sicher noch einige Arbeiten zu diesem Thema verfasst werden.

Auch das Ergebnis dieser Arbeit bietet noch Raum für Verbesserungen. So ist es z.B. momentan nicht möglich, aus den von der Gridshib-CA generierten Zertifikaten den zugehörigen Nutzer zu erkennen, da die CNs kryptische Bezeichnungen bekommen (z.B. CN=_8cf248c0f44c965c05772a8d0d1cfbcf). Daher werden im Mapfile alle CNs auf *einen* Account abgebildet. Ausserdem bekommt ein Nutzer jedes mal ein neues Zertifikat, wenn er eines anfordert, so dass es verschiedene Zertifikate für den selben Nutzer geben kann.

Abkürzungsverzeichnis

Abkürzung	Bedeutung
ACS	Assertion Consumer Service
CA	Certificate Authority
CSRF	Cross-Site Request Forgery
DAI	Data Access and Integration
DRS	Data Replication Service
EPR	Endpoint Reference
GRAM	Grid Resource Allocation and Management
GT 4	Globus Toolkit 4
IdP	Identity Provider
JAAS	Java Authentication and Authorization Service
JDBC	Java Database Connectivity
JNDI	Java Naming and Directory Interface
JNLP	Java Network Launching Protocol
JWS	Java Web Start
LDAP	Lightweight Directory Access Protocol
MDS	Monitoring and Discovery System
OASIS	Organization for the Advancement of Structured Information Standards
PKCS	Public Key Cryptography Standards
RFT	Reliable File Transfer
RLS	Replica Location Service
SAML	Security Assertion Markup Language
SP	Service Provider
SSO	Single Sign On

Abbildungsverzeichnis

4.1 SSO-Flow (Quelle: spaces.internet2.edu) 12
4.2 Zeitpunkt 0 13
4.3 Zeitpunkt 1 13
4.4 Zeitpunkt 2 14
4.5 Zeitpunkt 3 14

5.1 gridshib-ca-process-flow (Quelle: gridshib.globus.org) 17
5.2 Erzeugung und Inhalt des JNLP File 19

6.1 Komponenten des Globus Toolkit 4 (Quelle: globus.org) 22

7.1 Ablauf 27
7.2 Anmeldung beim Anmeldeserver 28
7.3 SSO-Flow (Quelle: spaces.internet2.edu) 28
7.4 Authentifizierung durch den IdP 29

Literaturverzeichnis

[Chi04] Chirico: *SQLite Tutorial*, 2004. http://freshmeat.net/articles/sqlite-tutorial.

[Fos05] Foster: *A Globus Primer*, 2005. http://www.globus.org/toolkit/docs/4.0/key/GT4_Primer_0.6.pdf.

[Fos06] Foster: *Globus Toolkit Version 4: A Software for Service-Oriented Systems*, 2006. http://www.globus.org/alliance/publications/papers/IFIP-2006.pdf.

[Fou10] Foundation, Apache Software: *Realm Configuration HOW-TO*, 1999-2010. http://tomcat.apache.org/tomcat-6.0-doc/realm-howto.html#JDBCRealm.

[Kli09] Klingenstein, et al.: *Shibboleth 2 Dokumentation*, 2009. https://spaces.internet2.edu/display/SHIB2/UnderstandingShibboleth.

[Lon01] Longman: *Programming Ruby: The Pragmatic Programmer's Guide*, 2001. http://ruby-doc.org/docs/ProgrammingRuby/.

[Neu09] Neukirchen: *Rack: a Ruby Webserver Interface*, 2007, 2008, 2009. http://rack.rubyforge.org/.