Die Benutzerverwaltung der Digitalen Bibliothek MyCoRe in einem Shibboleth und LDAP Umfeld

Diplomarbeit

Joachim Meyer

Fakultät für angewandte Wissenschaften
Institut für Informatik

Albert Ludwigs Universität Freiburg

September 2006

Erklärung

Hiermit bestätige ich, dass ich diese Arbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Ferner bestätige ich, dass alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder unveröffentlichten Schriften entnommen sind, als solche gekennzeichnet sind.

Darüber hinaus wurde diese Arbeit weder als Ganzes noch in Auszügen für eine andere Prüfung oder Prüfungsleistung angefertigt.

Freiburg, den

Joachim Meyer

Danksagungen

An dieser Stelle möchte ich mich bei allen bedanken, die zum Gelingen dieser Arbeit beigetragen haben. Ein ganz besonderer Dank gilt zunächst Detlev Degenhardt. Er war als ständiger Betreuer meiner Arbeit immer eine große Hilfe. Mit ihm konnte ich jederzeit ausführlich die Probleme und Fragestellungen meiner Arbeit diskutieren.

Weiterer Dank gilt dem Team der UB Freiburg, besonders Jochen Lienhard und Franck Borel, das mich durch ihr fundiertes Wissen tatkräftig unterstützt hat. Bei der Mycore-Community bedanke ich mich ebenso, und hoffe einen kleinen Beitrag zur Erweiterung von Mycore geleistet zu haben.

Weiterhin bedanke ich mich bei Prof. Dr. Schneider für die Bereitstellung des Themas.

Bei Claudia und Nils möchte ich mich für das Korrekturlesen der Arbeit bedanken. Schließlich bedanke ich mich bei meinen Eltern für die Unterstützung während des gesamten Studiums.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Wer das Internet häufig für berufliche und private Zwecke einsetzt, stellt schnell fest, dass viele Seiten im Netz für den Zugriff einen Nutzer-Account verlangen. Ein Nutzernamen und entsprechendes Passwort ist schnell gewählt. Das Problem ist aber die immer größer werdende Anzahl von Kombinationen, die man sich merken muss. Vom schriftlichen Notieren oder wiederholtem Benutzen des Passworts wird aus offensichtlichen Gründen dringend abgeraten. Folglich bleibt nur, sich die Kombination aus Nutzernamen und Passwort für jede Seite einzeln zu merken.

Richtet man den Blick ein wenig spezifischer auf den Bereich einer größeren Bildungseinrichung wie beispielsweise der einer Universität, bietet sich ein ähnliches Bild. Jeder Nutzer ist meist gezwungen, für jeden Dienst einen eigenen Account zu beantragen. Dies fängt beim Benutzen der Rechnerpools an und geht bis zum Zugriff auf Ergebnisse der eigenen Klausuren. Für jeden Dienst wird eine extra Zugangsberechtigung benötigt. Dies erzeugt außerdem für jedes System einen gewissen administrativen Aufwand.

Gleichzeitig gibt es gerade im universitären Umfeld immer mehr Bedarf an Angeboten von digitalen Inhalten für die Studenten. Dabei kann es sich um digitale Aufzeichnungen von Vorlesungen, Archive von wissenschaftlichen Arbeiten oder Fotos von Kunstwerken handeln. Der Zugriff sollte von überall möglich sein, aber trotzdem auf bestimmte Nutzer beschränkt bleiben. Das Problem der unzähligen Accounts für einen Nutzer wird sich also in Zukunft noch verschärfen.

Für das Mitglied einer Universität wäre folglich das primäre Ziel, einen einzigen Account zu erhalten, der zumindest alle web-basierten Dienste abdeckt. Jeder Nutzer erhält einen Nutzernamen und ein Passwort für alle Angebote der Universität, die über das Internet erreichbar sind. Insgesamt würde sich der Aufwand durch die Verwendung eines solchen Systems, sowohl auf der administrativen Seite, als auch auf der Seite der Nutzer, reduzieren.

Ein System, das die Accounts aller Nutzer einer Universität verwaltet, muss kompatibel

zu allen angeschlossenen Diensten sein. Jede Art von web-basierter Anwendung muss daher eine Schnittstelle bieten, um das Anmelden eines Nutzers an einer zentralen Stelle zu akzeptieren. Gleichzeitig darf der Funktionsumfang jeder einzelnen Ressource nicht beeinträchtigt werden.

Diese Arbeit beschäftigt sich hauptsächlich mit zwei verschiedenen Systemen. Das erste System nennt sich Shibboleth - eine mächtige und flexible Lösung zum Schutz von digitalen Inhalten in einer web-basierten Umgebung. Shibboleth ist genau so eine Art von Lösung, die dem Nutzer einen Account für alle Dienste bietet. Es lässt sich für eine Zugriffskontrolle beliebiger webbasierter Systeme einsetzen. Das zweite System nennt sich Mycore, und ist eine multimediale Digitale Bibliothek. Mit Mycore können verschiedenste digitale Inhalte über das Internet einfach und zugangsbeschränkt angeboten werden. Es handelt sich um die Art von System, das zukünftig immer mehr eingesetzt werden wird, da das Angebot von digitalen Inhalten jeglicher Art immer wichtiger werden wird. Beide Systeme sind Open Source und dadurch kostenlos einsetzbar.

Das Ziel ist die reibungslose Zusammenarbeit beider Systeme - dafür muss Mycore mit einer Schnittstelle für den Einsatz von Shibboleth auszustatten werden. Mycore besitzt eine eigene Nutzerverwaltung. Somit muss bisher für jede konkrete, auf Mycore basierenden Anwendung ein extra Account für jeden Nutzer vergeben werden. Hier stellt sich die Frage, wie dies bei dem Einsatz von Shibboleth aussieht. Lässt sich Shibboleth ohne weiteres in Mycore integrieren und was für Probleme ergeben sich dabei?

Zusätzlich wird der Einsatz einer einfacheren externen Nutzerverwaltung für Mycore mittels LDAP untersucht werden. Dabei handelt es sich im Vergleich zu Mycore um eine passive Datenbank für die Accounts von Nutzern. Dadurch ist der Einsatz von LDAP für eine Vielzahl von unterschiedlichen Diensten auch schwieriger zu realisieren. Shibboleth und LDAP sind unterschiedliche Systeme und ihr Einsatz mit Mycore kann durch einen Vergleich besser bewertet werden. Gleichzeitig wird insgesamt ein größeres Spektrum an Möglichkeiten abgedeckt, als dies bei der alleinigen Betrachtung von Shibboleth der Fall wäre.

Durch den Fokus auf zwei große unterschiedliche Bereiche befindet sich nicht die gesamte Theorie am Anfang der Arbeit. Sowohl Shibboleth als auch Mycore werden zunächst getrennt voneinander betrachtet, wobei jeweils auf theoretische und praktische Aspekte der Systeme eingegangen wird. Kapitel 2 dient zur Einführung in Shibboleth und zeigt das Potential des Systems.

In Kapitel 3 wird dies dann von der praktischen Seite beleuchtet. Ein einfache Testseite und ein Java Servlet werden mit Hilfe von Shibboleth geschützt. Es wird gezeigt, wie man Shibboleth in einer webbasieren Umgebung einsetzen kann.

Mycore und speziell dessen Nutzerverwaltung wird zu Beginn von Kapitel 4 vorgestellt. Danach folgt eine theoretischen Betrachtung des Einsatzes von Shibboleth in Mycore.

Kapitel 5 wiederum zeigt die praktische Umsetzung des zuvor diskutierten Themas, nämlich den praktischen Einsatz von Shibboleth anhand von prototypisch entworfenen

Klassen.

Wie sich LDAP in das System von Mycore integrieren lässt, wird in Kapital 6 dargestellt. Der Inhalt erstreckt sich von der Einführung in LDAP bis hin zur konkreten Implementierung in Mycore.

Kapitel 2

Shibboleth

2.1 Bedeutung des Wortes „Shibboleth“

Das Wort „Shibboleth“ wurde bereits in der Bibel erwähnt. Dort wurde es konkret benutzt um Feind von Freund zu unterscheiden. Nach dem Gewinn einer großen Schlacht war oft nicht klar, wer nun Angehöriger des eigenen Stammes und wer ein Feind war. Wie konnte man nun feststellen, wer nur vorgab zu den Gewinnern zu gehören? Die Lösung war ein Wort, das nur Mitglieder des eigenen Stammes richtig aussprechen konnten. Daher musste jeder, dessen Stammeszugehörigkeit unklar war, in dem konkret genannten Fall das Wort „Shibboleth“ auszusprechen. Die feindliche Volksgruppe hatte eine andere Sprache und konnte in der Regel das Wort nicht richtig aussprechen. Sie sagten daher „Sibboleth“. Somit war der Stamm in der Lage, alle eigenen Leute eindeutig zu identifizieren. So entstand die Bedeutung des Wortes „Shibboleth“.

Auch im letzten Jahrhundert wurden während des zweiten Weltkrieges Shibboleths benutzt, um feindliche Spione zu enttarnen. Es musste ein Wort gefunden werden, das von Freund und Feind unterschiedlich ausgesprochen wird. Holländer beispielsweise sprechen bei manchen Worten das „Sch“ als getrennte Laute „s“ und „ch“ aus, wobei Deutsche dies als einen Laut aussprechen. Als konkretes Wort wurde „Scheveningen“ benutzt. Es beginnt mit „sch“ und wird von Deutschen und Holländern unterschiedlich ausgesprochen. Mit diesem Shibboleth hat das holländische Militär versucht deutsche Spione zu enttarnen.

Im Bereich Computer bekommt das Wort „Shibboleth“ eine leicht abgewandelte Bedeutung. Das Konzept von Shibboleth bedeutet etwas zu testen, und je nach Ergebnis eine Aktion auszuführen. Das klassische Beispiel ist ein Login an einem beliebigen System. Der Nutzer hat dazu ein Passwort oder ein Schlüssel auf einem Chip. Daran ist zu sehen, dass sich zwei Klassen von Shibboleth unterscheiden lassen: Das Passwort ist etwas, von dem ich Kenntnis habe, der Schlüssel auf dem Chip ist etwas, das ich besitze.

2.2 Grundprinzip von Shibboleth

Abbildung 2.1: Das Logo von Shibboleth

Shibboleth [1, 2] ist ein System, das grundsätzlich für den Schutz von beliebigen webbasierten Systemen oder Diensten eingesetzt werden kann. Es ist ein vielfältiges und umfangreiches Werkzeug, das von einfachem Schutz bis zur komplizierten Rechteverwaltung alles bietet. Dabei geht es auch um den Austausch bzw. die Bereitstellung von Attributen eines Nutzers in einer webbasierten Umgebung. Shibboleth kann für ein angeschlossenes System die gesamte Nutzerverwaltung übernehmen. Zusätzlich wird durch Information über den einzelnen Nutzer eine differenzierte Zugriffskontrolle möglich. Bei Shibboleth handelt es sich um eine so genannte MiddleWare, die zwischen dem Client des Nutzers und dem Server der Webanwendung angesiedelt ist.

2.3 Vorteile von Shibboleth

Was genau bietet Shibboleth nun und worin liegt der Vorteil? Betrachtet man eine Reihe von Webanwendungen in einer Organisation, so stellt man fest, dass meist jede Anwendung seine eigene Verwaltung der Nutzer und deren Rechte hat. Greift ein Nutzer auf verschiedene Systeme zu, so muss er sich immer wieder erneut anmelden. Dabei hat er meist unterschiedliche Nutzernamen und Passwörter, die er sich merken muss. Aus der Sicht der Anbieter ergeben sich gewisse Aufgaben, die immer wieder getrennt bearbeitet werden müssen. An jedem System müssen die Nutzer verwaltet werden. Zusätzlich müssen Informationen zu jedem Nutzer gespeichert werden, falls es eine Zuweisung von verschiedenen Rechten für eine Autorisierung im System gibt. Es ist auch notwendig, an jedem System für ausreichende Sicherheit zu sorgen. Meist wird ein System, auf das nur von einer bestimmten Gruppe von Nutzern zugegriffen werden soll, durch den ausschließlich lokalen Zugang gesichert. Diese Vorgehensweise schränkt die Nutzungsmöglichkeiten stark ein.

Shibboleth übernimmt nun diese Aufgaben für die Anwendung bzw. Ressource. Ressource bedeutet hier ein beliebiger Inhalt auf einem Webserver, auf den normalerweise über einen Webbrowser zugegriffen werden kann. Das Spektrum reicht von einer einzelnen statischen HTML-Seite bis zu einer komplexen Webanwendung. Wichtig jedoch ist, dass Shibboleth nur für eine webbasierte Ressource eingesetzt werden kann.

Durch den Einsatz von Shibboleth ergeben sich zunächst folgende Vorteile:

- Es ist keine Nutzerverwaltung auf der Seite der Anwendung mehr nötig.
- Ressourcen können differenzierter geschützt werden, da eventuell mehr Informationen über einen Nutzer zur Verfügung stehen.
- Eine Integration eines bestehenden Systems ist oft leicht möglich.
- Der Aspekt der Sicherheit muss nur einmal betrachtet werden. Dadurch erhöht sich die Sicherheit der angeschlossenen Ressourcen. Gleichzeitig entfällt der Aufwand an jeder einzelnen Anwendung.
- Der Nutzer benötigt für alle angeschlossenen Ressourcen nur einen Nutzernamen und ein Passwort. Und er kann durch einmaliges Anmelden auf mehrere Ressourcen zugreifen.
- Die Nutzung einer Ressource wird durch Shibboleth möglicherweise unabhängig vom Standort.
- Statische Inhalte können sehr einfach geschützt werden.
- Shibboleth ist selbst Open-Source und basiert auf einer Reihe von Open-Source Projekten. Dadurch ist es kostenlos und bietet Sicherheit durch die Veröffentlichung aller internen Mechanismen.

Dies sind die grundsätzlichen Vorteile, die Shibboleth bietet. Später werden an entsprechender Stelle noch weitere aufgezeigt werden (siehe Kapitel 3 und 4).

2.4 Das Single Sign-On (SSO) Prinzip

Wie bereits erwähnt muss sich ein Nutzer nur ein einziges mal im Shibboleth System anmelden. Dazu benutzt er seine UserID und sein Passwort. Anschließend hat er Zugriff auf alle angeschlossenen Ressourcen, soweit es seine Berechtigungen zulassen. Dies nennt man das Single Sign-On (SSO) Prinzip. Der Nutzer meldet sich einmal an und ist somit für jede angeschlossene Anwendung authentifiziert.

Shibboleth ist so ausgelegt, dass dieses Prinzip immer zur Anwendung kommt. Der Nutzer kann durch einmalige Authentifizierung auf mehrere Ressourcen hintereinander zugreifen. Er besitzt nur einen Nutzernamen und ein Passwort, mit dem er sich einmalig zu Beginn seines Zugriffs im System anmelden muss. Wird die Arbeit für eine längere Zeitspanne unterbrochen, wird eine erneute Anmeldung erforderlich.

Das SSO Prinzip ist eine der größten Stärken von Shibboleth. Es erleichtert den Aufwand auf der Seite des Nutzers in erheblichem Maße. Gleichzeitig ergeben sich auf der Seite der Anwendung einige Vorteile, da der Nutzer nach der Authentifizierung automatisch von der Ressource identifiziert werden kann und bei entsprechender Berechtigung sofort den Zugriff erhält.

2.5 Komponenten von Shibboleth

Shibboleth selbst ist in verschiedene Komponenten unterteilt. Es bietet sich an, zunächst diese zu besprechen, bevor auf ein konkretes Beispiel eingegangen wird. Durch diese Struktur ist Shibboleth so modular aufgebaut, dass einfach neue Ressourcen in das System aufgenommen werden können.

2.5.1 Identity Provider (IdP)

Der zentrale Punkt in einem Shibboleth System ist der Identity Provider (IdP). Dieser verwaltet die Nutzer und die Daten, die für jeden Nutzer gespeichert sind. Sinnvollerweise gibt es pro Organisation genau einen IdP - eine Universität beispielsweise hat somit einen IdP. Jeder Nutzer, der auf eine angeschlossene Ressource zugreifen möchte, muss sich über einen IdP einloggen.

Wie die Informationen über einen Nutzer beim IdP gespeichert werden, ist zunächst beliebig. Es kann sich um einen LDAP-Server handeln, von dem der IdP die benötigten Daten erhält. Oder die Daten werden direkt auf dem Server des IdPs in einer Datenbank gespeichert. Wichtig ist, dass jede Ressource mit dem IdP verschlüsselt kommuniziert, um Vertraulichkeit zu gewährleisten.

Im IdP werden neue Nutzer hinzugefügt oder die Daten von Nutzern geändert. Der IdP ist somit der einzige Ort im System, an dem administrative Aufgaben erledigt werden müssen. Dies ist ein großer Vorteil, da es pro Organisation häufig nur einen IdP gibt. Es ist nicht mehr nötig an jeder einzelnen Ressource die Nutzer zu verwalten.

Für die Sicherheit der Nutzerdaten muss nur zentral am IdP gesorgt werden. Da Administration der Nutzer und Authentifizierung der Nutzer beim IdP erledigt werden, muss er ausreichend geschützt werden. Dabei ist sowohl die technische Seite, als auch die Kontrolle der Personen, die Zugang zum IdP haben, zu beachten. Wiederum entfällt diese Aufgabe an jeder Ressource. Ist der IdP ausreichend geschützt, so ergibt sich diese Sicherheit, zumindest in Bezug auf die Daten der Nutzer, automatisch für alle angeschlossenen Ressourcen.

2.5.2 Service Provider (SP)

Jede einzelne Ressource im System kommuniziert nicht direkt mit dem IdP. Dazu gibt es den so genannten Service Provider (SP). Dieser schützt eine Web-Anwendung bzw. eine Ressource direkt am Webserver. Ein SP kann nur zum Schutz von Web-Anwendungen eingesetzt werden, und nicht für eine beliebige Anwendung. Der SP befindet sich also auf dem selben Rechner bzw. Server wie die zu schützende Ressource. Will ein Nutzer auf eine am Shibboleth System angeschlossene Ressource zugreifen, so erhält zunächst der SP die Anfrage. Dieser entscheidet dann, was als nächstes passiert. Hat der Nutzer sich

noch nicht eingeloggt, so wird er zum IdP weitergeleitet. Authentifiziert sich dort der Nutzer erfolgreich beim IdP, so wird diese Information dem SP übermittelt. Zusätzlich können dies noch weitere Daten des Nutzers sein, je nach Einstellungen des Systems. Der SP gibt dann den Zugriff frei und der Nutzer erhält die angeforderten Inhalte der Ressource.

Wie funktioniert die Kommunikation zwischen IdP und SP von technischen Seite betrachtet? Hier bietet „Shibboleth Architecture“ einen guten Einstieg in die bei Shibboleth eingesetzten Techniken [3]. Im Abschnitt 2.11 wird auf des zugrunde liegende Protokolls SAML eingegangen.

Im System gibt es für jeden Webserver einen eigenen SP. Der SP ist so konfiguriert, dass er mit dem IdP verschlüsselt kommunizieren kann. Hat der Nutzer sich vor dem Zugriff auf eine Ressource bereits am IdP authentifiziert, so erhält der Nutzer sofort den Zugriff auf diese Ressource.

Soll eine neue Anwendung oder Ressource auf einem neue Webserver an das Shibboleth System angeschlossen werden, so wird ein neuer SP benötigt. Der SP muss am Server installiert und konfiguriert werden. Die Ressource ist damit geschützt und im Shibboleth System integriert. Dadurch hat jeder Nutzer beim IdP einen Zugriff auf die neue Ressource.

2.5.3 WAYF oder Lokalisierungsdienst

Es gibt nun auch die Möglichkeit, dass ein Shibboleth System mehrere IdPs besitzt. Will ein Nutzer auf eine Ressource zugreifen, so weiß der SP zunächst nicht, zu welchem IdP er die Anfrage weiterleiten soll. Dazu wird noch ein weiterer Service dazwischen geschaltet. Dieser nennt sich WAYF (Where Are You From?) oder Lokalisierungsdienst.

Greift ein Nutzer auf eine Ressource zu, so leitet der SP ihn zum WAYF weiter. Hier erhält der Nutzer ein Auswahlmenü mit allen im System verfügbaren IdPs. Der Nutzer wählt den IdP aus, an dem er einen Account besitzt. Dies ist üblicherweise der IdP seiner Heimatorganisation - beispielsweise die Universität, bei der er Mitglied ist. Nun leitet der WAYF die Anfrage an den ausgewählten IdP weiter. Ein WAYF wird bei einer so genannten Federation benötigt, was in Abschnitt 2.9 erklärt wird.

2.6 Beispiel

In diesem Unterkaptiel wird ein typischer Ablauf in einem Shibboleth System dargestellt. In dem Beispiel geht es um die Kommunikationswege für einen Nutzer, der auf zwei Ressourcen auf unterschiedlichen Servern zugreifen möchte. Abbildung 2.2 zeigt, wie die einzelnen Komponenten des Shibboleth Systems nacheinander angesprochen werden.

Der Nutzer greift zunächst auf Ressource A zu (1). Der Webserver enthält einen aktiven

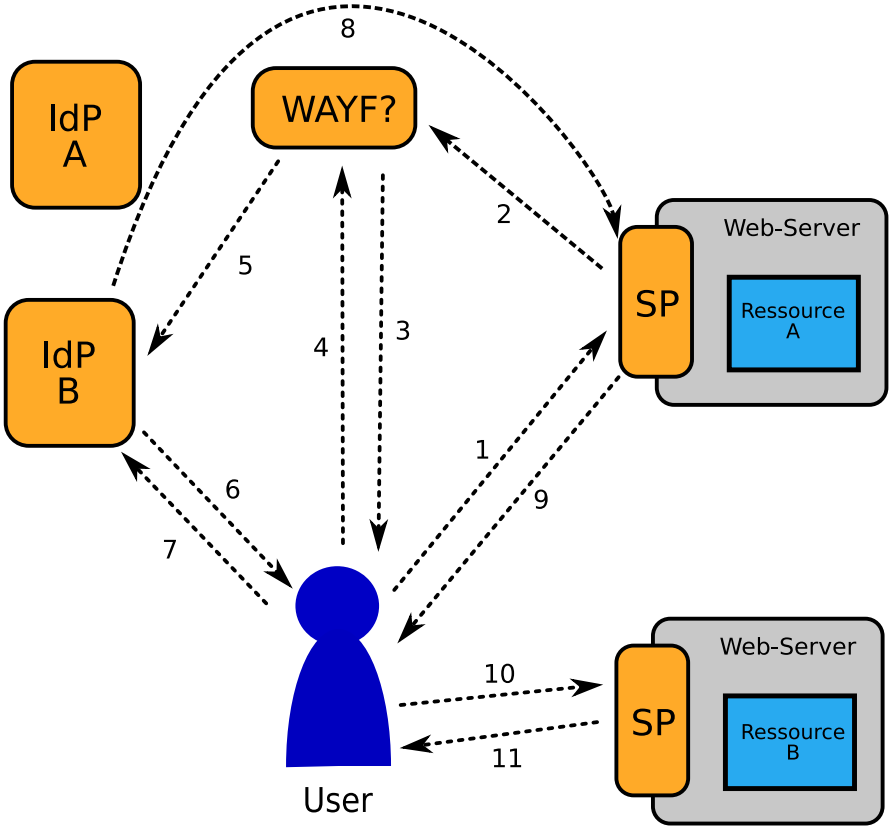

Abbildung 2.2: Ablauf eines Zugriffs auf zwei verschiedene Ressourcen

SP, der alle Anfragen an den Server überprüft. Da der Nutzer noch unbekannt ist und sich nicht im Shibboleth System authentifiziert hat, verweigert der SP den Zugriff auf die Ressource. Er leitet den Nutzer automatisch zu dem WAYF weiter (2). Dieser fordert den Nutzer dazu auf, seinen IdP zu wählen (3). Der Nutzer trifft seine Wahl und sendet diese Information dem WAYF (4). Anschließend wird der Nutzer (in diesem Fall) zum gewählten IdP B geleitet (5). Hat der Nutzer sich am IdP erfolgreich authentifiziert (6+7), muss er sich ab diesem Zeitpunkt nicht mehr am System anmelden. Der IdP leitet den Nutzer wieder automatisch zu dem SP zurück, der um die Authentifizierung gebeten hat (8). Der SP gewährt nun den Zugriff auf die Ressource A und der Nutzer erhält seinen angeforderten Inhalt (9).

Durch den Zugriff auf Ressource B, die sich auf einem anderen Server befindet, kann man einen der Vorteile von Shibboleth erkennen. Der Nutzer will auf die Ressource B zugreifen (10). Der SP prüft den Zugriff und stellt fest, dass der Nutzer sich bereits zuvor über einen IdP im System authentifiziert hat. Der Zugriff für den Nutzer erfolgt unmittelbar, ohne dass er sich erneut anmelden muss (11).

2.7 Attribute als Weg zur Autorisierung

Bisher wurde noch nicht erklärt, wie eine Ressource einen Nutzer eindeutig identifizieren kann. Im einfachsten Szenario enthält der SP nur die Information, dass der Nutzer sich erfolgreich über den IdP authentifiziert hat. Die Ressource verhält sich im System nur passiv und hat keinen Einfluss auf das Shibboleth System. Der SP lässt alle authentifizierten Nutzer ohne Ausnahme auf die Ressource zugreifen. Der Nutzer bleibt für den SP anonym. Dies ist sicherlich für einige, besonders statische Ressourcen, völlig ausreichend.

Wie kann man aber einerseits den SP selektiver werden lassen und andererseits der Ressource mehr Informationen über den Nutzer zur Verfügung stellen? Dies ist über den Einsatz von so genannten Attributen möglich. Diese sendet der IdP auf Anfrage an den SP. Dabei ist das sicherlich am häufigsten genutzte Attribut die UserID. Damit können Nutzer von einem IdP eindeutig zugeordnet werden.

Für den Einsatz von Attributen muss zwischen zwei Einsatzgebieten unterschieden werden.

2.7.1 Attribute für den SP

Für eine Ressource, die selbst keine Logik für eine Selektion des Zugriffs bietet, kann der SP eine Autorisierung durchführen. Der SP fordert dazu bestimmte Attribute des Nutzers vom IdP. Aufgrund dieser Attribute gewährt er den Zugriff nur, wenn der Nutzer dazu autorisiert ist. Er lässt also nicht mehr automatisch alle Nutzer auf eine Ressource zugreifen, die sich erfolgreich am IdP authentifiziert haben. Dadurch erreicht der Schutz einer Ressource eine neue Qualität.

Ein Beispiel für ein Attribut ist die Rolle in einer Organisation. An einer Universität wären solche Rollen „Student“ oder „Professor“, um nur zwei zu nennen. Enthält eine Ressource Informationen, die nur für Professoren bestimmt sind, so kann man den SP so einstellen, dass nur Nutzer mit dem Wert „Professor“ beim Attribut „Rolle“ Zugriff auf die Ressource erhalten. Der IdP sendet nach jeder Authentifizierung eines Nutzers das Attribut Rolle an den SP. Dann verweigert der SP allen Nutzern, die nicht die Rolle „Professor“ innehaben, den Zugriff auf die Ressource.

Der SP kann auch so konfiguriert werden, dass er eine Kombination von Attributen überprüft. Logisch lassen sich die Kombinationen mit „Und“ und „Oder“ verknüpfen. Der SP wird dadurch zu einer Art Ressource Manager. Allerdings sollte man beachten, dass der SP nur für einfachere Fäll von unterschiedlichen Zugriffsrechten eingesetzt werden sollte, ähnlich dem vorher genannten Beispiel. Eine komplexere Unterscheidung von Zugriffen, wie beispielsweise einzelnen ausgesuchten Nutzern, die kein gemeinsames Attribut besitzen, den Zugriff zu gewähren, sollte damit nicht realisiert werden. In dem Fall müsste jeder einzelne Nutzer manuell in der Konfiguration eingetragen werden.

2.7.2 Attribute für die Ressource

Als zweite Möglichkeit kann man der Ressource selbst Attribute der Nutzer zur Verfügung stellen. Dies bietet sich für Ressourcen an, die eine eigene Nutzerverwaltung oder zumindest eine interne Zugriffslogik besitzen. Dadurch kann beispielsweise mit dem Attribut „UserID“ der Nutzer identifiziert werden. Damit kann die Ressource alle Aktionen eines Nutzers eindeutig zuweisen.

Durch weitere Attribute kann die Ressource Entscheidungen in Bezug auf die Autorisierung treffen. In einigen Fällen kann sicherlich eine bestehende Logik an das Shibboleth System angepasst werden. Mit Hilfe der Attribute erhält die Ressource durch Shibboleth Informationen über den Nutzer. Diese bieten eine Fülle von Möglichkeiten, um verschiedene Bereiche mit unterschiedlichen Zugriffsrechten in der Ressource zu etablieren. Die Ressource kann also mit Hilfe der Attribute Inhalte differenziert anbieten. Dies kann sogar so weit gehen, dass von der Ressource automatisch generierte Dokumente bereits mit Daten des Nutzers versehen werden.

2.8 Datenschutz

Fordert eine Ressource oder ein SP Attribute vom IdP, so sind dies möglicherweise auch Informationen, die der Nutzer nicht preisgeben möchte. Wie kann er nun erreichen, dass seine Daten geschützt werden? Das Shibboleth System verhält sich so, dass zunächst keine Attribute vom IdP zum SP übertragen werden. Fordert der SP Attribute vom IdP an, so werden nur diese und nicht alle Attribute des Nutzers übertragen.

Sollte der SP Attribute vom IdP anfordern, so muss der Nutzer darauf Einfluss nehmen können. Daher kann der Nutzer beim Shibboleth System angeben, welche seiner Daten an einen SP übertragen werden dürfen. Dies kann er dem Administrator mitteilen und dieser setzt für die einzelnen Daten die jeweiligen Freigaben. Stimmt er einer Freigabe von bestimmten Attributen nicht zu, so werden diese nicht zum SP und damit zur Ressource gesendet. Der Nachteil, der sich für einen Nutzer ergeben kann, ist, dass ein SP oder eine Ressource ihm den Zugriff aufgrund der fehlenden Attribute verweigert.

Das Setzen der Freigaben für die persönlichen Daten ist grundsätzlich eine optionale Funktion in Shibboleth. Man kann ein System auch so konfigurieren, dass automatisch alle Daten an den SP übertragen werden, falls dieser sie anfordert.

Shibboleth bietet also eine sehr gute Grundlage für die Einhaltung des Datenschutzes. Sollte das konkrete Shibboleth System die oben genannte Funktionalität anbieten, so hat der Nutzer den Zugriff auf die Freigabe seiner persönlichen Daten.

2.9 Federation

Schließen sich mehrere Organisationen mit einem Shibboleth System zusammen, so nennt man dies eine Federation. Denkbar wäre eine Federation, der mehrere Universitäten angehören. Hier ist es wichtig sicherzustellen, dass bei allen IdPs die gleichen Attribute vorhanden sind. Dies wird in der Spezifikation der Federation definiert. Somit vereinbaren alle Mitglieder die Attribute, die für jeden Nutzer angeboten werden.

Beispielsweise schließen sich die Universitäten A, B und C zu einer Federation zusammen. Will ein Student der Universität A auf eine Ressource der Universität B zugreifen, so wird er von dem entsprechenden SP zunächst zum WAYF weitergeleitet. Dort wählt er seine Organisation aus, in diesem Fall den IdP der Universität A, und loggt sich dann dort ein. Der SP der Ressource der Universität B erhält dann die Attribute vom Nutzer und entscheidet über die Autorisierung. Ist eine Bedingung für den Zugriff z.B. derart, dass der Nutzer Mitglied der Universität B sein muss, so erhält er folglich keinen Zugriff auf die Ressource.

2.10 Die Technik von Shibboleth

Eine Implementation von Shibboleth gibt es in C++ und Java. Allerdings wird die Java Version in der 1.x Reihe nicht weiterentwickelt. Zur Version 2.0 soll für beide Programmiersprachen eine Variante mit identischen Funktionen veröffentlicht werden. Daher ist Shibboleth in der Version 1.x nur in einer C++ Serverumgebung einsetzbar. Empfohlen wird daher als Webserver der Apache Server[1]. Der SP besteht aus zwei Teilen. Davon ist eines ein Apache Modul, das dafür sorgt, dass der Zugriff auf ein Ressource im Server geschützt werden kann. Wie dies genau passiert, wird im nächsten Kapitel anhand der Beispiele gezeigt. Der zweite Teil ist eine eigenständiges Programm, der so genannte Daemon. Dieser dient zur Speicherung der Daten über die Sitzung der jeweils aktiven Nutzer und erledigt den Hauptteil der Aufgaben im Shibboleth System. Mehr dazu ebenso im nächsten Kapitel anhand der praktischen Beispiele.

Die Kommunikation wird durch den Einsatz von SSL (Secure Socket Layer) verschlüsselt. Dazu werden an jedem SP und IdP asynchrone Schlüssel erzeugt. Jede Komponente im System kennt daher alle öffentlichen Schlüssel der Kommunikationspartner. Somit wird dafür gesorgt, dass kein Angreifer, der die Kommunikation zwischen IdP und SP abhört, die Sicherheit des Shibboleth Systems gefährden kann. Der Zugriff auf die Ressource selbst ist nicht notwendigerweise verschlüsselt. Dies hängt ganz von den Ansprüchen der Ressource selbst ab.

[1] www.apache.org - Open Source Webserver

2.11 SAML

Mit welcher Art von Protokoll kommunizieren die einzelnen Komponenten von Shibboleth? Hier kommt das SAML Protokoll ins Spiel. Shibboleth basiert auf diesem Protokoll, das von OASIS (Organization for the Advancement of Structured Information Standards) als Standard verabschiedet wurde. Die Beschreibung und die technischen Dokumente zu SAML von OASIS findet man unter der Webseite des Projektes [6]. SAML bedeutet Security Assertion Markup Language und basiert auf XML. Dabei handelt es sich bei SAML lediglich um eine Spezifikation eines Protokolls. Eine konkrete Implementation bietet SAML nicht.

Das Hauptziel von SAML ist das Lösen des Single Sign-On Problems. Es gibt für SSO verschiedene andere Techniken, die untereinander nicht kompatibel sind. Folglich muss ein authentifizierter Nutzer sich in einem fremden System erneut anmelden. SAML versucht genau dies zu ermöglichen. Für Systeme aus dem Bereich Authentifizierung und Autorisierung bietet SAML eine Grundlage, die eine Interoperabilität zwischen zwei verschieden Systemen ermöglicht.

Shibboleth setzt konkret auf das SAML Protokoll auf. Shibboleth in der Version 1.3 basiert noch auf SAML 1.1. Die zukünftige Version 2.0 von Shibboleth wird auf SAML 2.0 basieren, das signifikante Neuerungen mit sich bringt. SAML wurde entwickelt, um Daten zur Authentifizierung und Autorisierung zwischen zwei Sicherheitsbereichen auszutauschen. Dies sind im Falle von Shibboleth SP und IdP. Der SP möchte einen Nutzer beim IdP authentifizieren und leitet den Nutzer zum IdP weiter. Hat der Nutzer sich erfolgreich authentifiziert, schickt der IdP eine SAML Nachricht an den SP. Beispielsweise sieht eine solche Nachricht folgendermaßen aus:

```
1  <saml:Assertion
2   xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion"
3   MajorVersion="1" MinorVersion="1"
4   AssertionID="..."
5   Issuer="https://idp.org/saml/"
6   IssueInstant="2002-06-19T17:05:37.795Z">
7   <saml:Conditions
8    NotBefore="2002-06-19T17:00:37.795Z"
9     NotOnOrAfter="2002-06-19T17:10:37.795Z"/>
10  <saml:AuthenticationStatement
11     AuthenticationMethod="urn:oasis:names:tc:SAML:1.0:am:password"
12     AuthenticationInstant="2002-06-19T17:05:17.706Z">
13    <saml:Subject>
14      <saml:NameIdentifier
15        Format="urn:oasis:names:tc:SAML:1.1:nameid-format:emailAddress">
16        user@mail.idp.org
17      </saml:NameIdentifier>
18      <saml:SubjectConfirmation>
19        <saml:ConfirmationMethod>
20          urn:oasis:names:tc:SAML:1.0:cm:artifact
21        </saml:ConfirmationMethod>
22      </saml:SubjectConfirmation>
```

```
23      </saml:Subject>
24    </saml:AuthenticationStatement>
25  </saml:Assertion>
```

Listing 2.1: SAML-Nachricht Authentifikation

In Zeile 1-9 stehen einige Informationen zu der Art des Protokolls und der Gültigkeitsdauer. In Zeile 11 befindet sich die Information, dass der Nutzer sich mit seinem Passwort authentifiziert hat. Der Block von Zeile 13-23 beschreibt den Nutzer bzw. das so genannte Subjekt. Hier wird dem SP die Email-Adresse des Nutzers übermittelt (16). Dies muss nicht zwingend der Fall sein. Ohne diese Adresse würde der Nutzer für den SP anonym bleiben.

In dem Beispiel fehlen noch Attribute, die der SP vom IdP erhalten kann. Dies wird an einem weiteren Beispiel veranschaulicht:

```
1  <saml:Assertion
2    xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion"
3    MajorVersion="1" MinorVersion="1"
4    Issuer="https://idp.edu/saml/" ...>
5    <saml:Conditions NotBefore="..." NotAfter="..."/>
6    <saml:AuthenticationStatement
7      AuthenticationMethod="urn:oasis:names:tc:SAML:1.0:am:X509-PKI"
8      AuthenticationInstant="...">
9      <saml:Subject>...</saml:Subject>
10   </saml:AuthenticationStatement>
11   <saml:AttributeStatement>
12     <saml:Subject>...</saml:Subject>
13     <saml:Attribute
14       AttributeName="urn:mace:dir:attribute-def:eduPersonScopedAffiliation"
15       AttributeNamespace="urn:mace:shibboleth:1.0:attributeNamespace:uri">
16       <saml:AttributeValue Scope="idp.edu">
17         member
18       </saml:AttributeValue>
19       <saml:AttributeValue Scope="idp.edu">
20         student
21       </saml:AttributeValue>
22     </saml:Attribute>
23   </saml:AttributeStatement>
24 </saml:Assertion>
```

Listing 2.2: SAML-Nachricht Authentifikation

Die ersten fünf Zeilen sind identisch mit dem vorigen Beispiel. Bei der Authentifizierung ist eine andere Methode gewählt worden (7). Der Nutzer hat sich mit einem Public Key authentifiziert. Dies wurde möglicherweise mit einer Smart Card oder einem anderen Mechanismus realisiert. Die genaue Beschreibung des Nutzers bzw. des Subjekts in den Zeilen 9 und 12 wurde zugunsten einer besseren Übersicht weggelassen. Ab Zeile 11 beginnt die Beschreibung der Attribute. Dazu wurde das Attribut `eduPersonScopedAffiliation` gewählt, was den Status eines Nutzers in einer Organisation im Bereich der Bildung beschreibt. Dieses Attribut enthält zwei Werte: Der

Nutzer ist Mitglied der Organisation (17) und er ist Student (20).

Natürlich kommunizieren die einzelnen Komponenten in Shibboleth nicht direkt über das SAML Protokoll. Dazu wird SAML durch das SOAP Protokoll gekapselt. SOAP stand ursprünglich für Simple Object Access Protocol. Diese Beschreibung wird aber inzwischen abgelehnt, da sie irreführend ist. SOAP wurde zuerst von Microsoft entwickelt, ist aber inzwischen unter der Obhut des World Wide Web Konsortiums (W3C). SOAP ist ein Protokoll, um neben dem Datenaustausch zwischen zwei Systemen auch so genannte Remote Procedure Calls auszuführen. Eine Remote Procedure Call steht für den Aufruf einer Funktion bei einem externen System. Das SAML Protokoll benötigt genau diese Eigenschaften. Hier ruft der SP Funktionen des IdPs auf. SOAP selbst wird wiederum durch das HTTP Protokoll gekapselt. Eine gute Beschreibung mit einigen Beispielen ist bei Wikipedia unter dem Stichwort SAML zu finden [8].

Der Webbrowser des Nutzers wird jeweils an die entsprechende Komponente mittels eines HTTP Redirects weitergeleitet. Bei einem Redirect verweist der Webserver beim Zugriff auf eine bestimmte Adresse den Webbrowser automatisch zu einer neuen Adresse. Diese Technik wird in Shibboleth eingesetzt. Wird die Authentifizierung eines Nutzers durch den SP gefordert, leitet dieser den Browser auf die entsprechende Seite des IdPs weiter. Nach dem erfolgreichen Anmelden wird mit Hilfe eines Redirects wieder der Zugriff auf die ursprünglich gewählte Adresse veranlasst.

2.12 Standards für Attribute

Die Attribute eines Nutzers werden wie bereits erwähnt beim IdP gespeichert. Nun ist es sinnvoll, in Hinblick auf den Zusammenschluss von mehreren Systemen zu einer Federation dafür zu sorgen, dass die Attribute untereinander kompatibel sind. Dies wird durch Standards erreicht. Für die meisten benötigten Attribute gibt es solche Standards. Möchte man zu jedem Nutzer an der Universität speichern, ob er Student, Mitarbeiter oder eine andere beliebige Funktion hat, so sollte dazu ein bestimmtes Attribut benutzt werden. Dies ist in diesem Fall das Attribut `eduPersonPrimaryAffiliation`. Die geläufigen Attribute gibt es als Standard für Shibboleth und Projekten aus dem Bereich der Bildung. Zu finden ist der Standard als Teile des MACE Projekts auf einer speziellen Webseite [7].

2.13 Zusammenfassung

Shibboleth ist von der Konzeption her eine so genannte Middleware. Es versucht die beiden Aufgaben Authentifizierung und Autorisierung von einer Ressource oder einer Web-Anwendung loszulösen. Diese Anforderungen tauchen oft auf und müssen normalerweise in jeder Anwendung getrennt betrachtet werden. Ebenso muss der Sicherheitsaspekt immer wieder berücksichtigt werden, was meist einen großen Aufwand bedeutet

- einerseits die technische Realisierung, andererseits die Kontrolle derjenigen Personen, die Zugriff auf die Nutzerdatenbank haben. Durch den Einsatz von Shibboleth wird dies alles zentral verwaltet. Jede einfache Ressource ohne eigene Nutzerverwaltung oder Zugriffsbeschränkung kann damit sehr einfach eine Zugriffskontrolle erhalten. Eine Web-Anwendung, die zwischen verschiedenen Nutzern unterscheidet, kann nun über die Attribute, die sie vom IdP erhält, den Nutzer mit seinen Rechten im System abbilden.

Kapitel 3

Aufbau eines einfachen Shibboleth Systems

3.1 Schutz einer HTML-Seite

Um einen Einstieg in die Installation eines Shibboleth Systems zu erhalten, wird als erstes gezeigt, wie eine statische HTML-Seite geschützt werden kann. Es sei angenommen diese befindet sich unter der URL `http://www.example.org/portal/index.html` und wird über einen Standard-Webserver angeboten. Ohne ein angeschlossenes Shibboleth System kann die Seite wie gewohnt über die Adresse angesprochen werden. Der Browser des Nutzers zeigt den Inhalt der Seite an. Ziel ist es nun, dass der Zugriff auf die Seite nur noch für Nutzer möglich ist, die sich erfolgreich über den IdP einloggen.

3.1.1 Aufbau des Systems

Für den Einsatz von Shibboleth werden verschiedene Komponenten benötigt. Das Hauptaugenmerk liegt zunächst auf der Authentifizierung. Ein erfolgreiches Anmelden beim IdP reicht daher als Berechtigung für den Zugriff auf die Seite aus.

Es wird im Folgenden davon ausgegangen, dass auf einen bereits bestehenden Identity Provider (IdP) zugegriffen wird. Auf die Installation eines IdPs und dessen Konfiguration wird nicht eingegangen. Im Rahmen der Arbeit wurde ein bereits bestehendes Testsystem inklusive IdP genutzt. Dieser enthält eine Reihe von Accounts, die zum Testen eines Shibboleth Systems benötigt werden.

Nun werden die Softwarekomponenten beschrieben, die in dem gesamten System zusammenarbeiten. Je nach Komplexität der Einstellungen wird kurz auf die Konfiguration eingegangen oder auf externe Quellen verwiesen. Wichtig ist, dass der komplette Prozess für den Aufbau des Systems zum Testen dargestellt wird.

3.1.2 Apache2 HTTP Server

Als Webserver wird der Apache HTTP Server[1] in der Version 2 benutzt. Der Apache Webserver, womit im Folgenden immer der HTTP Server gemeint ist, ist einer der am weitesten verbreiteten Webserver im Internet. Er ist Open-Source und wird, wie eine Reihe von weiteren wichtigen Open-Source Projekten, von der Apache Software Foundation[2] betreut. Der Apache Webserver bietet die HTML Seite an, auf die der Nutzer mittels eines Webbrowsers zugreifen kann. Für die Anleitung zur Installation eines Apache HTTP Servers wird auf die Homepage des Projektes verwiesen. Es werden grundsätzlich zunächst keine speziellen Einstellungen für den Apache Webserver benötigt. Der Rechner, auf dem der Apache Webserver installiert wird, enthält auch die entsprechende Datei `index.html`. Gegebenenfalls muss der Zugriff von außerhalb des Rechners aktiviert werden, falls nur ein Zugriff von `localhost` möglich sein sollte.

3.1.3 Shibboleth Indentity Provider (IdP)

Der IdP verfügt über eine Datenbank mit allen Nutzern und ihren Daten. Der IdP befindet sich auf einem anderen Server, der sich wiederum an einer beliebigen Stelle im Netz befinden kann. Alle Ressourcen, die mit dem Shibboleth System geschützt werden, sind dann an diesen IdP angebunden. Auf den Aufbau des IdPs wird wie bereits erwähnt nicht speziell eingegangen, sondern er wird als gegeben vorausgesetzt.

3.1.4 Shibboleth Service Provider (SP)

Der SP und der Apache Webserver werden auf dem selben Rechner installiert. Für die Installation werden einige weitere Software Komponenten benötigt. Dazu gehört beispielsweise OpenSAML[3], wobei es sich um eine konkrete Implementierung des SAML Protokolls handelt. Der SP der Testanwendung muss beim IdP eingetragen werden, damit dieser den SP erkennt und mittels Zertifikaten eine sichere Kommunikation über SSL möglich ist. Auf die weiteren Komponenten und Einstellungen zur Installation wird nicht näher eingegangen. Ein gute Anleitung zur Installation bietet z.B. das Projekt AAR[4] (verteilte Authentifizierung, Autorisierung und Rechteverwaltung) der Bibliothek der Universität Freiburg [10].

Der Service Provider besteht aus zwei Komponenten. Ein Teil, der so genannte Shibboleth Daemon, ist eine eigenständige Anwendung und übernimmt die meisten Aufgaben des SPs. Er speichert Informationen über Nutzer, generiert die SAML Meldungen, u.v.m. Wichtig ist auch, dass der Shibboleth Daemon beim Bootvorgang gestartet werden sollte.

[1] httpd.apache.org
[2] www.apache.org
[3] www.opensaml.org
[4] aar.vascode.de

Er muss spätestens vor dem Start des Apache Webservers aktiv sein, damit der Zugriff auf den Webserver immer durch Shibboleth geschützt ist.

Wie funktioniert nun der Schutz der HTML-Seite, die der Apache Webserver anbietet? Dazu wird ein so genanntes Modul für den Apache Webserver benötigt, das die zweite Komponente des SPs darstellt. Durch eine spezielle Architektur ist der Apache Webserver einfach durch Module erweiterbar. Jedes Modul kann über eine eigene Datei konfiguriert werden. In der des Shibboleth Moduls werden die Einstellungen für den zu schützenden Bereich des Webservers festgelegt. Das Shibboleth Modul nennt sich mod_shib und wird über die Datei `mod_shib.conf` konfiguriert. Hier kann wie beim Apache Webserver üblich mit den Tags `<Directory>`, `<Location>` oder `<Files>` gearbeitet werden. Diese legen einen Bereich für bestimmte Einstellungen des Webservers fest. Konkret also Dateien, Verzeichnisse oder URLs. Als Beispiel nun der Schutz der Seite, die sich unter der URL `http://www.example.org/portal/index.html` befindet:

```
1 ...
2 <Location /portal>
3     AuthType shibboleth
4     ShibRequireSession On
5     require valid-user
6 </Location>
7 ...
```

Listing 3.1: Auszug aus der Konfigurationsdatei mod_shib.conf des Shibboleth Moduls

Durch diese Einstellung ist nicht nur die Datei `index.html` geschützt, sondern jede weitere Seite bzw. Datei, auf die über die URL `http://www.example.org/portal/` zugegriffen werden kann. Zeile 3 und 4 geben an, dass der Schutz durch Shibboleth für die angegebene URL aktiviert ist. Mit Hilfe eines authentifizieren Nutzers (Zeile 5) wird der Zugriff gewährt.

Damit ein SP mit einem IdP kommunizieren kann, sind noch einige weitere Einstellungen beim SP nötig. Diese werden hauptsächlich in der Datei `shibboleth.xml` vorgenommen. Weiterhin müssen noch SSL Zertifikate ausgetauscht werden. Eine detaillierte Beschreibung der einzelnen Einstellung ist in einigen Quellen bereits ausreichend vorhanden. Wie bereits erwähnt, bietet das Shibboleth Projekt AAR einen guten Einstieg, ebenso wie auch das Schweizer Shibboleth Projekt von SWITCH[5] eine gute Beschreibung zum Erstellen eines SPs bereitstellt [4].

3.1.5 Ergebnis

Alle Anfragen an URLs, die mit `http://www.example.org/portal` beginnen, also auch `http://www.example.org/portal/index.html`, werden durch das Shibboleth Modul im Apache Webserver gefiltert. Hier wird eine gültige Session von Shibboleth verlangt. Folglich erhält der SP bei einer Anfrage an diese spezielle URL die Aufgabe, einen

[5]The Swiss Education & Research Network - `www.switch.ch`

gültigen Nutzer vom IdP anzufordern. Sollte dies gelingen, so wird der Zugriff gewährt und der Apache Webserver sendet die angeforderten Daten an den Browser des Nutzers. Andernfalls wird der Zugriff verweigert, da kein erfolgreicher Login beim IdP erfolgt ist.

Ob und welche Attribute des Nutzers der IdP sendet, ist an dieser Stelle noch nicht näher definiert. Es wird nur gefordert, dass der Nutzer sich am IdP einloggen kann bzw. einen Account hat. Sollte dies der Fall sein, so wird erfolgreich eine Session erzeugt und der SP gibt den Zugriff frei. Dabei bedeutet Session, dass sowohl der SP als auch der Client des Nutzers (Browser) einige Informationen zum Nutzer und dessen Anmeldung beim IdP speichert. Zusätzlich können dort noch Attribute vorhanden sein, die aber erst später betrachtet werden (siehe Abschnitt 3.3.3).

Mit dieser einfachen Konfiguration kann der Zugriff auf die statische HTML-Seite gesteuert werden. Natürlich ist ebenso der Schutz von komplexeren Webanwendungen analog zu realisieren. Wie schon zuvor gezeigt, kann der Schutz für einen bestimmten Präfix einer URL aktiviert werden. Folglich ist die Technik der Webanwendung, die der Apache Server anbietet, völlig beliebig. Falls nun die HTML-Seite dynamisch durch PHP[6] Code oder eine andere Programmier- oder Skriptsprache, die der Apache Webserver unterstützt, erzeugt wird, ändert dies an dem Schutz und der dazu nötigen Konfiguration nichts.

Dies zeigt, dass ein bestehendes Shibboleth System mit einem SP sehr flexibel ist. Durch die oben gezeigten Einstellungen kann schnell ein weiterer Bereich des Angebots des Apache Webservers geschützt werden. Dabei reichen die Einstellungsmöglichkeiten für den Schutz durch Shibboleth von einzelnen Dateien im Dateisystem bis hin zu ausgewählten URL-Bereichen.

3.2 Schutz von Servlet-basierten Anwendungen

Wie können nun Webanwendungen, die aus dem Java-Bereich stammen, geschützt werden? Diese benutzen meist die Java Server Pages oder Java Servlets Technologie, um dynamisch HTML-Seiten zu erzeugen. Eine Einführung in diese Technik bietet ein entsprechendes Tutorial [17]. Der Apache Webserver unterstützt keine Java-basierten Webanwendungen. Der Schutz einer Servlet-basierten Anwendung ist jedoch ein wichtiger Punkt, da Mycore, das im nächsten Kapitel vorgestellt wird, auf dieser Technologie basiert.

[6] www.php.net

3.2.1 Tomcat als Servlet Container

Es stellt sich die Frage, wie Servlets durch Shibboleth geschützt werden können. Für diese Zweck wird der Tomcat Webserver[7] (im August 2006 in der Version 5.5 für Java 5.0) eingesetzt. Er ist, wie der Apache Webserver, Open Source und wird von der Apache Software Foundation betreut. Tomcat fungiert als so genannter Container für eine Webanwendung, die aus Servlets oder JSPs besteht.

Auf die Installation eines Tomcat Webservers sowie dessen Einstellungen wird nicht weiter eingegangen, da dazu ausreichend Dokumentation an verschiedene Stellen vorhanden ist. Die Dokumentation zu Tomcat bietet eine erste Anlaufstelle. Viele Informationen über den Tomcat Server sowie die nötigen Einstellung sind detailliert in "Tomcat 5 - Einsatz in Unternehmensanwendungen mit JSP und Servlets" von Lajos Moczar beschrieben [11]. Es erhält auch viele Anregungen für einen effizienteren Einsatz von Tomcat.

Der Tomcat Webserver bietet beispielsweise über Servlets dynamisch generierte HTML-Seiten an. Der Nutzer kann auf diese mit einem Webbrowser wie gewohnt zugreifen. Wie kann nun der Zugriff auf Tomcat bzw. auf bestimmte Bereiche des Webservers durch Shibboleth beschränkt werden? Dazu wird nicht etwa ein neues SP Modul im Tomcat Webserver eingesetzt., sondern es wird weiterhin der Apache Webserver benötigt. Dieser kann mit Hilfe seines Shibboleth Moduls auch den Schutz für Tomcat übernehmen. Erreicht wird dies dadurch, dass der Apache Webserver als eine Art Zwischenstation für die Kommunikation des Webbrowsers und Tomcat dient.

Für die Kommunikation der beiden Webserver Apache und Tomcat wird ein weiteres Apache Modul namens mod_jk benötigt. Von einer Benutzung des Moduls mod_jk2 ist abzusehen, da dieses Modul nicht mehr weiter entwickelt wird. Mit dem mod_jk Modul können Anfragen auf bestimmte Bereiche des Apache Webservers zum Tomcat Webserver weitergeleitet werden. Dadurch ist es möglich, Tomcat ebenfalls mit dem Shibboleth Modul des Apache Webservers schützen zu lassen.

Das Modul mod_jk ist nicht immer in der aktuellen Version fertig compiliert verfügbar. Beim Einsatz im Betriebssystem Linux bieten einige Distributionen noch den mod_jk2 Connector als vorgeschlagenes Paket an. Daher ist es meist sinnvoll, sich Quellen herunterzuladen und selbst zu compilieren. Eine gute Anleitung ist auf der Webseite von HowToForge zu finden [14].

Nach der Installation des Apache Moduls muss dieses noch entsprechend konfiguriert werden. Dazu gibt es zwei Dateien, auf die im weiteren Verlauf eingegangen wird. In der Datei `workers.properties` befinden sich die Einstellungen, die den Tomcat Webserver betreffen (Listing 3.2).

```
1 worker.list=server1
2 worker.server1.port=8009
3 worker.server1.host=localhost
```

[7] tomcat.apache.org

```
4 worker.server1.type=ajp13
```

Listing 3.2: Konfiguration von Tomcat für das Apache Moduls mod_jk - workers.properties

In Zeile 1 wird der Tomcat Webserver mit dem Namen `server1` definiert. Er befindet sich auf dem selben Rechner bzw. Server und kann mittels eines speziellen Protokolls angesprochen werden. Dies wird in Zeile 2-4 festgelegt.

Die Einstellungen für das Modul werden über die Datei `mod_jk.conf` vorgenommen (Listing 3.3).

```
1 JkWorkersFile "workers.properties"
2 JkLogFile "log/mod_jk.log"
3 JkLogLevel warn
4 JkMount /portal server1
5 JkMount /portal/* server1
```

Listing 3.3: Konfiguration des Apache Moduls mod_jk - mod_jk.conf

Zeile 1 verweist auf die Konfiguration des Tomcat Servers aus Listing 3.2. Die Einstellungen zu den Log Dateien sind in den nächsten beiden Zeilen zu finden. Wichtig sind Zeile 4 und 5. Dort wird festgelegt, dass alle Anfragen an das Verzeichnis `portal` beim Apache Webserver an Tomcat weitergeleitet werden. Dies ist nur ein Beispiel eines einzelnen Bereiches. Hier sind weitere Definitionen denkbar.

Nun darf aber keine direkte Kommunikation zwischen dem Tomcat Webserver und dem Webbrowser des Nutzers mehr möglich sein, sonst könnte man das Shibboleth System umgehen und eine Sicherheit wäre nicht vorhanden. Dazu sind Änderungen in der Konfiguration des Tomcat Webservers nötig. Am wichtigsten in diesem Zusammenhang ist die Datei `server.xml`. Das folgende Listing 3.4 zeigt einen Auszug aus der entsprechenden Datei:

```
1 ...
2 # <Connector port="8080"/>
3 ...
4
5 ...
6 <Connector port="8009" protocol="AJP/1.3" />
7 ...
```

Listing 3.4: Auszug aus der Tomcat Konfigurationsdatei server.xml

Meistens gibt es in der Konfiguration des Tomcat Servers zwei Tags mit dem Namen Connector. Dabei sollte der Connector mit dem Port 8080, der normalerweise für Anfragen über das HTTP Protokoll zuständig ist, auskommentiert werden (Zeile 2). Zusätzlich gibt es noch einen Connector für die Kommunikation mit dem Apache Modul. Somit ist kein direkter Zugriff auf den Tomcat Server mehr möglich. Nur Anfragen über den Apache Server sind noch zugelassen. Sollte es noch weitere Ports geben, die über einen Connector-Tag aktiviert sind, müssen diese ebenso deaktiviert werden. Siehe hierzu auch die entsprechenden Stelle, in dem zuvor genannten Buch zu Tomcat [11, Seite 586].

3.3 Weitere Funktionalität von Shibboleth

Bevor ein Servlet als Beispiel besprochen wird, soll im Folgenden zunächst auf einige weiteren Eigenschaften von Shibboleth eingegangen werden. Diese bieten zusätzliche Funktionalität beim Einsatz von Shibboleth. Das abschließende Beispiel wird diese dann praktisch aufzeigen.

3.3.1 Die Lazy Session

Wird eine Webanwendung wie bisher gezeigt durch Shibboleth geschützt, so ist der Zugriff nur nach erfolgreicher Anmeldung des Nutzers über den IdP möglich. Die Ressource ist also grundsätzlich geschützt. Ein Sequenzdiagramm für den Zugriff auf beispielsweise ein Portal, das von einem geschützten Tomcat Webserver angeboten wird, ist in Abbildung 3.1 dargestellt. Der Nutzer wird automatisch zum IdP weitergeleitet, wo er sich erfolgreich anmelden muss. Nur dann erhält dieser den Zugriff auf den Tomcat Webserver bzw. das Portal.

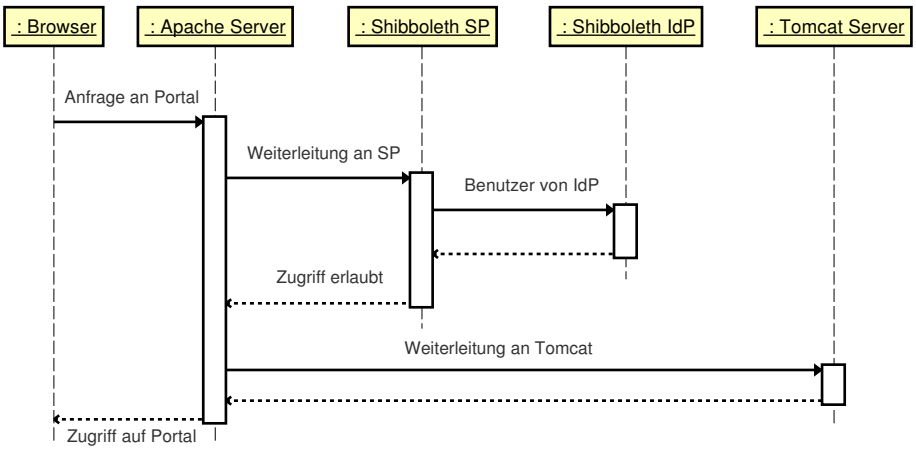

Abbildung 3.1: Schutz des Portals mit Shibboleth (Normal)

Der Zugriff auf die mittels Shibboleth geschützte Webanwendung ist somit eher restriktiv. Ist der Schutz aktiviert, ist kein Zugriff ohne Authentifizierung mehr möglich. Dies ist sicherlich in vielen Fällen das Ziel des Einsatzes von Shibboleth. Aber es gibt auch Webanwendungen, die selbst die Kontrolle über den Schutz behalten wollen. Es sind Bereiche in der Webanwendung vorhanden, die keinen Schutz durch Shibboleth benötigen. Somit ist eine Art Gastzugang möglich. Diese Anforderung kann mit Hilfe der so genannten Lazy Session bei Shibboleth erfüllt werden.

Bei der Lazy Session verhält sich Shibboleth faul oder träge, wie bereits durch das engl. Adjektiv „lazy“ angedeutet wird. Der Zugriff auf eine geschützte Seite wird zuerst einmal zugelassen. Shibboleth überwacht die Seite, hält sich aber zunächst im Hintergrund und

übergibt der Anwendung die Kontrolle über die Sicherheit. Dazu müssen die Einstellungen in dem Apache Modul mod_shib geändert werden. Listing 3.5 zeigt die Änderungen im Vergleich zur bisherigen Version in Listing 3.1.

```
1 ...
2 <Location /portal>
3     AuthType shibboleth
4     ShibRequireSession Off
5     require Shibboleth
6 </Location>
7 ...
```

Listing 3.5: Auszug aus der Konfigurationsdatei mod_shib.conf des Shibboleth Moduls mit Lazy Session

Die Änderungen befinden sich in den Zeilen 4 und 5. Durch diese Konfiguration ist der Zugriff auf den angegebenen Bereich zunächst nicht durch Shibboleth geschützt. Allerdings kann die Webanwendung selbst den Nutzer zum Einloggen auffordern. Dazu wird ein spezieller Link zum Aufruf des SP Moduls benötigt. Dieser leitet dann zum angeschlossenen IdP weiter. Damit kann der Nutzer sich über den IdP einloggen und erhält Zugriff auf den geschützten Bereich der Webanwendung. Hierbei ist jedoch zu beachten, dass die Kontrolle über die geschützten Bereiche einzig und allein durch die Webanwendung selbst gesteuert werden. Wie dies genau erreicht werden kann, wird im nächsten Abschnitt gezeigt.

Ein Sequenzdiagramm in Abbildung 3.2 zeigt den Zugriff auf das Portal. Wie man im ersten Teil erkennen kann, erlaubt der SP den Zugriff auf den Tomcat Webserver zunächst automatisch. Danach loggt sich der Nutzer über den IdP ein und er ist im Shibboleth System authentifiziert.

Wichtig an dieser Stelle ist auch noch der Fall, bei dem der Nutzer sich bereits zuvor für einen Zugriff auf einen anderen SP im System authentifiziert hat. Damit ist er im Shibboleth System angemeldet. Bei Shibboleth mit Lazy Session bekommt der entsprechende SP zunächst nicht mit, dass der Nutzer sich bereits angemeldet hat. Er lässt den Zugriff ohne eine Prüfung zu. Will der Nutzer sich dann über den entsprechenden Link einloggen, so leitet der SP ihn wie gewohnt zum IdP weiter. Dieser erkennt, dass der Nutzer bereits authentifiziert ist, und schickt dem SP diese Informationen. Es sind keine weiteren Eingaben des Nutzers nötig. Der Nutzer ist bereits angemeldet, wie dies durch das SSO-Prinzip vorgesehen ist.

3.3.2 Erkennen einer Shibboleth Session

Wie kann eine Webanwendung genau feststellen, ob sich ein Nutzer bereits über den IdP authentifiziert hat? Dies wird mit Hilfe der Shibboleth Session erreicht. Der Begriff Shibboleth Session bedeutet, dass ein Benutzer sich über einen IdP angemeldet und der SP daraufhin den Zugriff gewährt hat. Somit ist für den Nutzer eine aktive Shibboleth Session vorhanden.

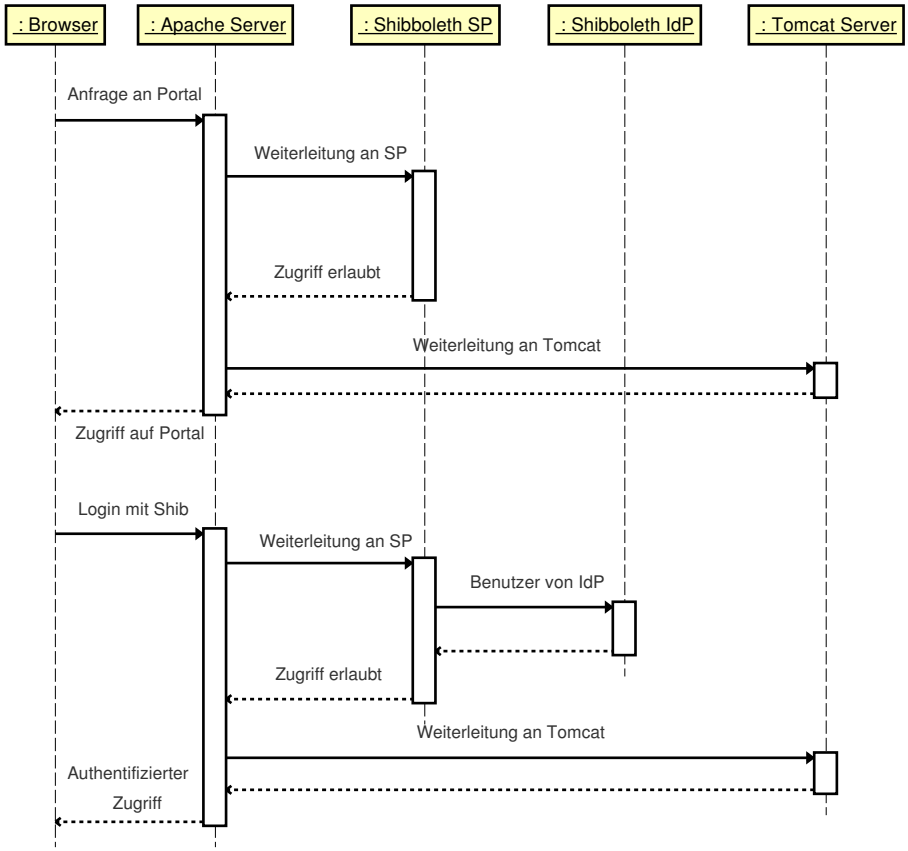

Abbildung 3.2: Schutz des Portals mit Shibboleth (Lazy Session)

Die Session enthält Informationen über die Authentifizierung des Nutzers und gegebenenfalls noch weitere Daten über den Nutzer in Form von Attributen (siehe Unterkapitel 2.7). Dabei sind bestimmte Informationen zur Authentifizierung immer verfügbar und Attribute des Nutzers optional. Das SP Modul im Apache Webserver macht diese dann für die Webanwendung zugänglich. Dazu werden die Daten in den Header der HTTP-Anfrage an den Webserver geschrieben. Die Webanwendung kann aus dem Header dann die gewünschten Informationen auslesen. Dies ist vor allem für Shibboleth mit Lazy Session wichtig, da die Webanwendung wissen muss, ob sich ein Nutzer bereits authentifiziert hat.

3.3.3 Einsatz von Attributen

Beide Arten von Shibboleth, sowohl mit als auch ohne Lazy Session, lassen sich durch den Einsatz von Attributen erweitern. Diese werden vom IdP zur Verfügung gestellt und bei einer erfolgreichen Authentifizierung an den SP weitergeleitet. Dieser überprüft dann die Einstellungen für die Attribute, die sich in der Datei `AAP.XML` befinden. Dort wird

festgelegt, welche Attribute unter welchem Namen in den Header geschrieben werden. Alle Attribute, die dort nicht genannt werden, sind auch nicht für die Webanwendung zugänglich. Daher sind nicht alle Attribute, die ein IdP zur Verfügung stellt, auch automatisch verfügbar.

Das wichtigste Attribut ist meistens die UserID bzw. der Login-Name des Nutzers. Dieses ist üblicherweise bei jedem IdP verfügbar und kann auch entsprechend an den SP weitergeleitet werden. Dort kann der Name in den Header der HTTP-Anfrage geschrieben werden. Somit kann die Anwendung diesen auslesen und den Benutzer eindeutig identifizieren.

3.4 Ein Beispiel: Das SimpleLoginServlet

Die zuvor besprochenen Neuerungen werden anhand eines Beispiels verdeutlicht. Dieses ist sowohl für Shibboleth mit als auch ohne Lazy Session einsetzbar. Zusätzlich soll auch gezeigt werden, wie der Tomcat Webserver geschützt werden kann.

Bei Shibboleth ohne Lazy Session muss sich der Nutzer zunächst am IdP anmelden. Dann gibt der SP den Zugriff für den Nutzer auf das Servlet frei. Dabei wird die UserID in den Header der HTTP-Anfrage geschrieben und das Servlet kann diese auslesen. Als Ausgabe wird im Webbrowser die UserID des Nutzers angezeigt. Dabei wird davon ausgegangen, dass der SP die UserID immer in den Header der HTTP-Anfrage schreibt bzw. die entsprechende Einstellung in der Konfiguration der Attribute vorhanden ist.

Im Fall von Shibboleth mit Lazy Session ist der Nutzer zunächst nicht authentifiziert und somit keine Shibboleth Session für ihn vorhanden. Zur Überprüfung der Shibboleth Session wird daher überprüft, ob die UserID im Header vorhanden ist.

Der Browser des Nutzers zeigt in diesem Fall den Status „nicht eingeloggt“ an. Weiterhin ist noch der spezielle Link zum Aufruf des SPs vorhanden. Nach der Weiterleitung zum IdP kann sich der Nutzer anmelden. Ruft er den Link auf, so wird er nach dem erfolgreichen Anmelden beim IdP wieder zum Servlet zurück geleitet. Der Browser des Nutzers zeigt dann die UserID und den Status „eingeloggt“ an.

```
1  ...
2  protected void processRequest(HttpServletRequest request,
3                                HttpServletResponse response)
4    throws ServletException, IOException {
5
6    String userid = "";        // Login Name vom IdP
7
8    response.setContentType("text/html;charset=UTF-8");
9    PrintWriter out = response.getWriter();
10   out.println("<html>");
11   out.println("<head>");
12   out.println("<title>SimpleLoginServlet</title>");
13   out.println("</head>");
```

```
14    out.println("<body>");
15    out.println("<h1>Simple Login Servlet</h1>");
16    out.println("</br>");
17    out.println("</br>");
18
19    username = request.getHeader("Shib-UserID");
20    if (username == null) {
21      out.println("Kein Nutzer eingeloggt!</br></br>");
22      out.print("<a href=\"http://www.example.org/Shibboleth.sso/WAYF/IDPs?
23  target=http://www.example.org/portal/SimpleLoginServlet\">");
24      out.print("Login ueber Shibboleth");
25      out.print("</link>");
26    } else {
27      out.println("UserID: ");
28      out.println(userid);
29      out.println(" eingeloggt");
30    }
31    out.println("</body>");
32    out.println("</html>");
33    out.close();
34  }
35  ...
```

Listing 3.6: Auszug aus SimpleLoginServlet.java

Listing 3.6 zeigt die wichtigste Methode des SimpleLoginServlets. Sie bearbeitet die HTTP-Anfrage und gibt eine entsprechende Antwort zurück. In Zeile 19 wird überprüft, ob der Header mit dem Namen `Shib-UserID` vorhanden ist. Dieser muss in den Einstellungen zu den Attributen beim SP identisch benannt werden. Sollte der Header vorhanden sein, ist der Nutzer eingeloggt und erhält die entsprechende Antwort.

Zeile 22 und 23 enthalten den Link für den Login beim IdP. Hier wurde eine fiktive Adresse gewählt. Diese muss an den entsprechenden WAYF oder IdP angepasst werden. Er besteht aus drei Teilen. Der erste Teil dient zum Aufruf des SP Moduls des Apache Webservers. Als nächstes wird der am Shibboleth System angeschlossene IdP (WAYF) angegeben. Der letzte Teil nach `target` gibt die Zieladresse an, die nach dem Login beim IdP, aufgerufen werden soll. In diesem Fall wird das Servlet wiederholt aufgerufen.

3.4.1 Ergebnis

Mit dem SimpleLoginServlet kann die Funktionsweise von Shibboleth sowohl mit also auch ohne Lazy Session veranschaulicht werden. Durch die Lazy Session gewinnt der Schutz eine neue Qualität, da die Anwendung einen ungeschützten Bereich etablieren kann. Will der Nutzer nun bestimmte Dienste innerhalb der Anwendung in Anspruch nehmen, so muss er sich unter Umständen anmelden - je nach Anforderung des Systems. Die Anwendung selbst steuert den Zeitpunkt der Authentifizierung.

Wichtig bei der Lazy Session ist, dass die Anwendung selbst einen Teilaspekt der Sicher-

heit des Systems übernehmen muss. Sollen bestimmte Bereiche nur mit einer gültigen Session zugänglich sein, so muss darauf geachtet werden, dass dies immer überprüft wird. Da Shibboleth zunächst keine Einschränkungen des Zugriffs vorgibt, ist jeder beliebige statische oder dynamische Inhalt zugänglich. Daher sollte Lazy Session nur gewählt werden, wenn auf jeden sicherheitsrelevanten Inhalt nur dynamisch zugegriffen werden kann. Dann kann immer überprüft werden, ob eine gültige Session vorhanden ist und somit ob der Nutzer auf den Inhalt zugreifen darf.

Ein Nutzer kann mit Hilfe der UserID eindeutig zugeordnet werden. Die Anwendung kann damit dem Nutzer persönliche Einstellungen anbieten oder seine Aktivitäten eindeutig zuordnen. Über die Attribute des IdPs kann der Nutzer genau zugeordnet werden. Falls vorhanden können seine persönlichen Einstellungen geladen werden, oder ihm werden andere individuelle Inhalte präsentiert.

Kapitel 4

MyCore und Shibboleth

In diesem Kapitel geht es um den Einsatz von Shibboleth für Mycore. Zunächst wird darauf eingegangen werden, um was es sich bei Mycore genau handelt und was für ein Konzept sich dahinter verbirgt. Dabei wird zuerst allgemein auf die Nutzer und deren Rollen in Mycore eingegangen werden. Weiterhin wird gezeigt werden, was für Prozesse sich dadurch ergeben. Danach wird Mycore vorgestellt werden. Dann wird versucht werden, die Verknüpfungspunkte von Shibboleth und Mycore aufzuzeigen. Dabei werden verschiedene Ansätze betrachtet und deren Vor- und Nachteile diskutiert werden. Dies bleibt aber noch theoretisch. Eine prototypische Implementierung für den Einsatz von Shibboleth in Mycore wird erst im nächsten Kapitel besprochen werden.

4.1 Dokumenten- und Publikationsserver

Mycore ist ein so genannter Dokumenten- oder Publikationsserver und dient in erster Linie zur Archivierung von elektronischen Dokumenten. Als weitere Funktionalität bietet er die Möglichkeit des Publizierens von Dokumenten. Dabei handelt es sich um einen Prozess, an dem mehrere Personen mit unterschiedlichen Rollen beteiligt sind. Im Folgenden wird erst einmal allgemein auf solch eine Art von Server eingegangen werden, ohne den Blick speziell auf Mycore zu richten.

4.1.1 Dokumente

Es stellt sich zunächst die Frage, was man genau unter einem solchen Dokument im Rahmen eines Dokumenten- und Publikationsservers versteht. Ein solches Dokument umfasst ein weites Spektrum von Inhalten. Es kann sich zum Beispiel um ein eingescanntes Bild oder auch um ein reines Textdokument handeln. Zunächst muss zwischen verschiedenen Arten von Dokumenten unterschieden werden. Man kann sie grob in drei unterschiedliche Klassen unterteilen.

- **Dokument mit analoger Vorlage**: Es gibt viele Dokumente, die nur in analoger Form zur Verfügung stehen. Dazu gehören u. a. alte Bücher, handschriftliche Aufzeichnungen, Münzen und Gemälde. Um Zugriff auf solche Dokumente zu erhalten, muss man vor Ort die entsprechende Einrichtung aufsuchen.

  Durch die Möglichkeit, die Dokumente zu digitalisieren, kann man diese durch einen Dokumentenserver besser zugänglich machen. Dies kann durch Scannen, Fotografieren oder andere Techniken erreicht werden. Die durch diesen Prozess erhaltenen Digitalisate werden in den Server übertragen und mit entsprechenden Metainformationen versehen, wie etwa Autor, Jahr, usw.

  Damit ist der Zugriff nicht mehr lokal beschränkt. Durch die angebundenen Informationen, über die jedes Dokument im Server verfügt, kann es auch über eine Suchfunktion gefunden werden.

- **rein digitales Dokument**: Die nächste Art von Dokument hat keine analoge Vorlage mehr. Es wurde digital erzeugt und kann daher direkt in den Server eingestellt werden. Die Metainformationen werden wie zuvor mit dem Dokument verknüpft, wodurch eine Suche im Server möglich ist.

  Ein solches Dokument ist meist nicht endgültig, sondern es können neue Versionen erzeugt werden. Der Autor kann an dem Dokument Änderungen vornehmen, was zu einer neuen Version führt. Ein Dokument erhält damit eine Versionsnummer oder das Datum der letzten Änderung. Der Server muss daher die Funktionalität bieten, die das Ändern dieser Dokumente erlaubt.

  Konkret kann es sich bei diesen Dokumenten zum Beispiel um einen Studienplan für eine Studienrichtung handeln. Immer wenn es etwa eine Änderung der Prüfungsordnung gibt, muss das Dokument auch entsprechend angepasst werden. Oder das Dokument enthält Ergebnisse aktuell laufender Forschungen, wobei ständig Änderungen vorgenommen werden müssen.

- **Publikation**: Eine Publikation ist nur auf den ersten Blick ähnlich zu dem rein digitalen Dokument, da es handelt sich eher um eine Erweiterung handelt. Das Dokument wird digital erzeugt und unterliegt gewissen inhaltlichen Änderungen. Somit können wie zuvor ständig neue Versionen des Dokuments erzeugt werden.

  Stellt man sich einen Publikationsprozess in einem Verlag vor, so hat dieser mehrere Phasen. Ein Autor erstellt ein Dokument und diese erste Version wird auf den Server geladen. Nun kann ein Lektor bzw. Reviewer sich das Dokument ansehen und eine Bewertung dazu schreiben. Daraufhin kann das Dokument wieder vom Autor geändert werden. Der Lektor bzw. Reviewer kann nun seine Bewertung ändern oder löschen. Handelt es sich zum Beispiel um eine Dissertation, so kann der Autor nach dem Abgabetermin keine Änderungen mehr vornehmen. Nach der offiziellen Abgabe kann die Arbeit von einer oder mehreren Personen bewertet werden, wobei diese Bewertungen als endgültig eingestuft werden.

  Eine Publikation führt also dazu, dass ein Dokument zusätzlich einen bestimmten

Status erhält. Es befindet sich im Prozess zu einem fertigen Produkt. An der Erzeugung des Dokuments sind gleichzeitig mehrere Personen mit unterschiedlichen Funktionen beteiligt.

4.1.2 Die verschiedenen Nutzer

Für die drei unterschiedlichen Arten von Dokumenten ergeben sich eine Reihe von Nutzern mit unterschiedlichen Rollen. Die einzelnen Typen von Nutzern werden im Folgenden erklärt:

- **einfacher Nutzer/Gast**: Der einfache Nutzer des Dokumenten- oder Publikationsservers hat eine passive Rolle im System. Er greift auf einzelne Dokumente nur lesend zu. Er ändert keine Dokumente und hat auch nicht die Möglichkeit, neue in das System einzustellen. Je nach Art des Inhalts des Servers gibt es noch eine Zugriffsbeschränkung für gewisse Dokumente. Außerdem muss der Nutzer die entsprechenden Rechte besitzen, damit der Zugriff gewährt wird.
- **Inhaltelieferant**: Dieser Typ von Nutzer hat die Rolle eines Lieferanten. Er hat das Recht neue Dokumente in das System einzustellen. Diese Dokumente können eine analoge Vorlage besitzen. Beispielsweise eine Datenbank mit Fotos von alten Gemälden. Der Inhaltelieferant erweitert nach und nach das Archiv. Oder es handelt sich um digitale Dokumente und es gibt verschieden Versionen eines Dokuments. Dieser Nutzer darf daher neue Dokumente einstellen und diese gegebenenfalls auch ändern. Ob der Nutzer alle Dokumente ändern darf oder nur die, die er selbst eingefügt hat, hängt von der Konfiguration des Systems ab.
- **Autor**: Der Autor kann neue Dokumente in das System einstellen. Dabei sind diese Dokumente in der Regel von ihm erzeugt worden. Damit hat nur er das Recht, seine Dokumente zu ändern. Zusätzlich kann er bestimmen, welche Nutzer überhaupt auf seine Arbeit zugreifen dürfen. Wenn er einem Dokument den Status „fertig“ verleiht, verliert auch er das Recht, weitere Änderungen vorzunehmen. Er kann dann nur noch lesend darauf zugreifen.
- **Lektor/Rezensent**: Der Lektor liest unfertige Dokumente eines Autors und verfasst Kritiken und Verbesserungsvorschläge. Der Rezensent beurteilt dagegen abschließend fertige Dokumente eines Autors. Die Aktionen dieser Nutzer sind daher eng verzahnt mit denen des Autors. Sie erhalten Zugriff auf die von ihnen betreuten Dokumente. Zwar können sie diese nicht ändern, aber Informationen zu den Dokumenten hinzufügen. Diese zusätzlichen Informationen können sich gegebenenfalls noch ändern oder sogar wieder entfernt werden.
- **Administrator**: Alle Funktionen zur Verwaltung des Servers übernimmt der Administrator. Er hat meistens alle möglichen Rechte im System. Er kann Nutzer anlegen oder löschen. Durch den Zugriff auf alle Dokumente kann er diese löschen

oder die Zugriffsrechte ändern. Natürlich muss zum Administrator immer ein spezielles Vertrauensverhältnis bestehen, da dieser durch seine Rechte alle Prozesse im Server beeinflussen kann.

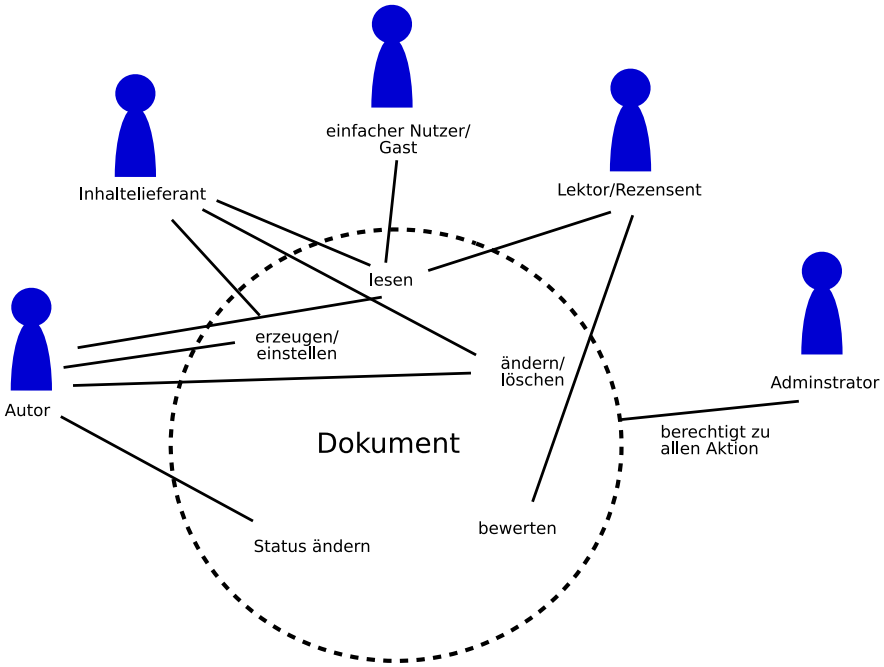

Abbildung 4.1: Nutzer und deren Aktionen im Rahmen des Dokumenten- und Publikationsservers

4.1.3 Komplexität

In Abbildung 4.1 ist ein Überblick über die einzelnen Nutzer bzw. Nutzergruppen aufgezeigt. Sie können verschiedene Aktionen in Bezug auf ein Dokument ausführen. Dies zeigt, dass ein solcher Server eine komplexe Funktionalität besitzt. Durch die verschiedenen Rollen der Nutzer und ihre dazugehörigen Aktionen ergibt sich ein große Vielfalt an Aktionen im Server. Abbildung 4.2 zeigt die Metadaten, die zu einem Dokument zur Verfügung stehen können. Sie können für bestimmte Zugriffsrechte eines Nutzers wichtig sein. Durch die Metadaten kann nach einem Dokument im Server gesucht werden.

Die Entwickler von Mycore haben es sich zum Ziel gesetzt, einen solchen Server zu realisieren. Dabei sind (noch) nicht alle oben genannten Aspekte in Mycore verfügbar. Dies zeigt aber, dass die Nutzerverwaltung ein wichtiger Aspekt in Mycore ist. Was nun Mycore genau ist und auf welchem Stand sich die aktuelle Entwicklung befindet, wird im nächsten Abschnitt erklärt.

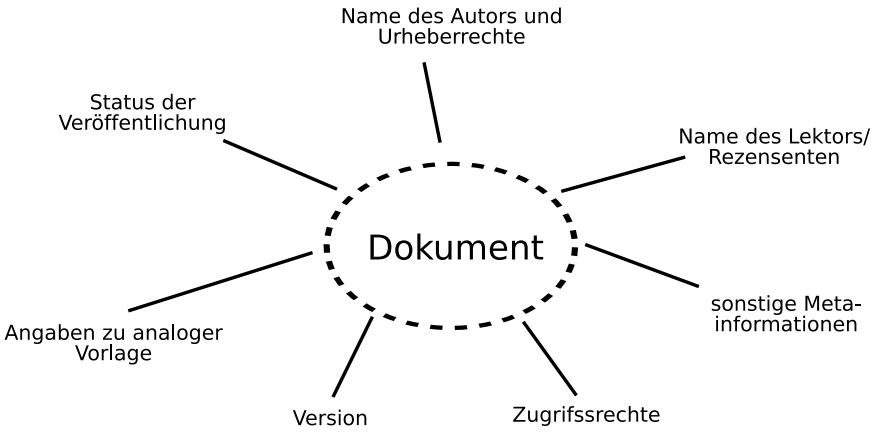

Abbildung 4.2: Verschiedene Metadaten zu einem Dokument

4.2 Mycore

4.2.1 Die Entstehung von Mycore

Im universitären Bereich ist der Bedarf eines Dokumenten- und Publikationsservers an vielen Stellen vorhanden. Angefangen von Arbeiten wie dieser Diplomarbeit bis hin zu eingescannte alten Zeitungen und Zeitschriften, gibt es viele Anwendungsmöglichkeiten.

Vor diesem Hintergrund entstand im Jahre 1997 das MILESS[1] Projekt, das schon einige Funktionalitäten eines Dokumentenservers bietet. Das Projekt ist Open Source und wurde im Laufe der Zeit auch von anderen Universitäten eingesetzt. Das Grundkonzept erwies sich aber als nicht flexibel genug, um MILESS ohne größeren Aufwand für ähnliche Aufgaben an anderen Universitäten anzupassen. Daher beschlossen im Jahre 2001 eine Reihe von Universitäten gemeinsam ein neues System zu entwickeln. Ziel war ein flexibles System, das einfach und schnell an verschiedene Aufgaben angepasst werden kann. Dies war der Startschuss zu Mycore.

Mycore ist eine Abkürzung für **M**ILESS Communit**y** **Co**ntent **Re**pository. Man kann aber auch das englische Wort „Core“, das übersetzt Kern bedeutet, als Deutung verwenden. Dadurch ergibt Mycore wörtlich übersetzt „mein Kern“, das als „mein eigenes Content Repository“ interpretiert werden kann.

Mycore wurde seitdem ständig weiterentwickelt. Vor kurzem (Stand August 2006) wurde von den Entwicklern die Version 1.3 freigegeben. Mycore erfüllt in dieser Version die meisten Anforderungen, die an einen Dokumten- und Publikationsserver gestellt werden. Allerdings sind nicht alle Funktionen für einen kompletten Publikationsprozess in Mycore vorhanden.

[1] Multimedialer Lehr- und Lernserver Essen, `www.miless.uni-essen.de`

4.2.2 Funktionalität von Mycore

Mycore ist zunächst im Kern nur ein Content Repository. Es bietet die Kernfunktionalität, die für die Verwaltung einer großen Zahl von Dokumenten benötigt wird. Dabei kümmert sich Mycore um die Speicherung der Daten in einer Datenbank. Auch die Suche nach Dokumenten oder die Nutzerverwaltung wird von Mycore übernommen. Dabei ist alles möglichst flexibel konfigurierbar.

Das Layout und die eingesetzte Funktionalität ist jedoch nicht festgelegt. Dies kann über eine XML-Konfiguration und über XSL-Stylesheets jeweils an die Bedürfnisse des konkreten Systems angepasst werden.

Damit ein neues System nicht immer komplett neu entworfen werden muss, gibt es vom Mycore Projekt eine Beispielanwendung. Diese nennt sich Docportal und ist ein klassischer Dokumentenserver. Eine neues System lässt sich somit am einfachsten durch eine Anpassungen von Docportal realisieren. Im nächsten Kapitel wird der Einsatz von Shibboleth mit Docportal getestet werden. Auf der Webseite von Mycore befinden sich weitere Informationen und eine detaillierte Dokumentation [5].

4.2.3 Technik von Mycore

Mycore basiert im Kern zunächst auf verschiedenen Java Klassen. Zur Kommunikation mit dem Nutzer über den Web Browser werden Servlets eingesetzt. Diese erfüllen verschiedene Aufgaben, wie die Authentifizierung oder die Auswahl des Layouts über Stylesheets. Das Layout wird über eine XSL Transformation erzeugt. Dabei wird XML dynamisch in HTML umgewandelt.

Zwischen den Java Klassen und den Daten befindet sich eine weitere Schicht. Diese sorgt dafür, dass unabhängig von der Art der Speicherung der Daten alle Zugriffe über ein Interface möglich sind. Dazu wird die Java Bibliothek Hibernate[2] eingesetzt. Sie erlaubt einheitliche Anfragen an die Datenbank. MySQL ist die empfohlene Open Source Datenbank. Es kann aber auch der proprietäre DB2 Content Manager von IBM[3] eingesetzt werden.

Mycore bietet auch eine Schnittstelle zu anderen Systemen. Es gibt die Möglichkeit einer Anbindung an ein Bibliothekssystem über OAI[4]. Andere Systeme können über allgemeinere Protokolle, wie beispielsweise XML oder SOAP, angesprochen werden.

Die Konfiguration wird durch den Einsatz von XML Dateien realisiert. Dadurch kann jede konkrete Anwendung auf ihre jeweiligen Bedürfnisse angepasst werden. Diese Einstellungen können zur Laufzeit von Mycore ausgelesen werden und stehen über eine Manager Klasse in allen anderen Klassen zur Verfügung.

[2] urlwww.hibernate.org

[3] `www.ibm.com/software/de/db2/cm/cm.html`

[4] Open Archive Initiative - `www.openarchives.org`

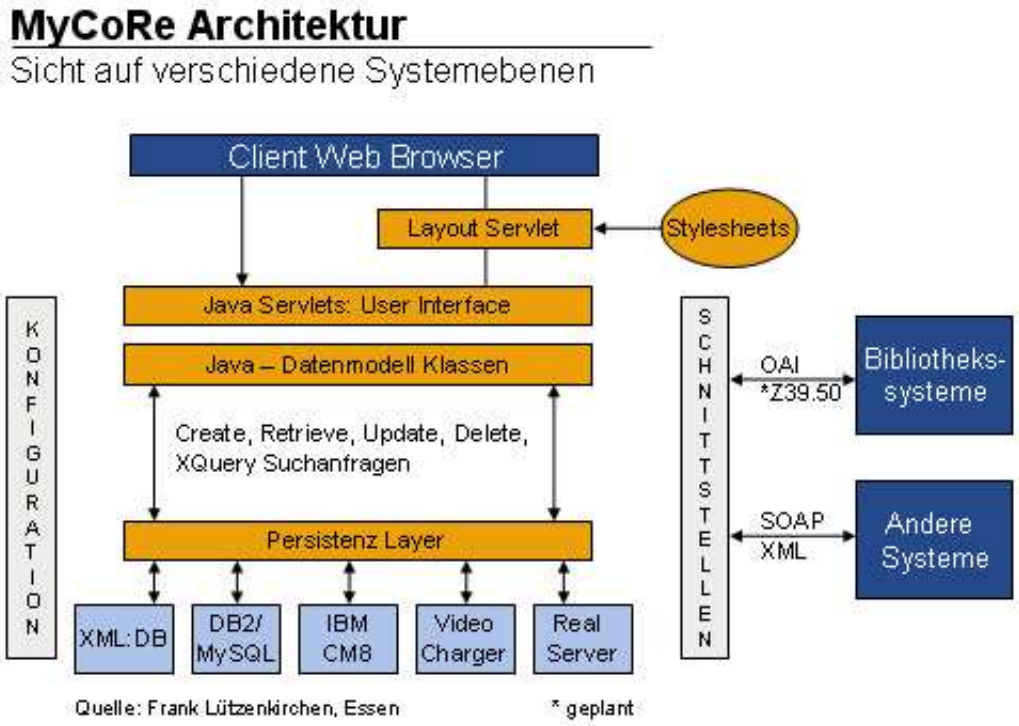

Abbildung 4.3: Die Architektur von Mycore

Abbildung 4.3 zeigt eine Übersicht der Architektur von Mycore. Wie man gut erkennen kann, verfügt Mycore über einen flexiblen und modularen Aufbau.

Eine sehr nützliche Funktion ist der Zusammenschluss von mehreren Mycore Systemen. Dadurch ist der Nutzer in der Lage eine Suche auf mehrere Instanzen von Mycore hinweg auszuführen. Somit kann ein Verbund von verschiedenen Bibliotheken entstehen. Jedes einzelne System wird dabei unabhängig von den anderen betrieben und gewartet.

4.3 Nutzerverwaltung

Mycore besitzt eine eigene Nutzerverwaltung, die jedem Nutzer eine UserID zuweist. Damit kann jeder Nutzer, der sich einloggt, identifiziert werden. Die Rechte, die ein Nutzer besitzt, werden aber nicht für jeden separat gespeichert. Jeder Nutzer ist Mitglied einer Gruppe, wobei diese einen gewissen Status besitzt. Beispielsweise befindet sich der Administrator in der Gruppe mit der Bezeichnung „Admin“ und hat dadurch bestimmte Rechte. Oder ein weiterer Nutzer ist Mitglied eine Gruppe „Autor“ und hat somit das

Recht, bestimmte Dokumente in das System einzustellen.

Beim Zugriff auf ein Mycore System ist der Nutzer zunächst nicht eingeloggt. Er ist automatisch Nutzer vom Typ Gast, der üblicherweise minimale Rechte besitzt. Dadurch können neue Nutzer, die noch keinen Account haben, Information über die Art und den Inhalt des konkreten Mycore Systems erhalten. Will der Gast Nutzer auf einen Bereich zugreifen, für den er zusätzliche Rechte benötigt, so wird er zum Einloggen aufgefordert. Hat sich der Nutzer mit seiner UserID eingeloggt, so darf er entsprechend seiner neuen Rechte auf zusätzliche Inhalte im System zugreifen.

Für den erfolgreichen Login im Mycore System wird ein persönliches Passwort benötigt, das beim Anlegen des Nutzers vergeben wird. Dieses Passwort wird verschlüsselt in der angeschlossenen Datenbank gespeichert. Will der Nutzer sich im System anmelden, so wird das eingegebene Passwort mit dem in der Datenbank verglichen. Der Administrator ist für das Anlegen neuer Accounts zuständig.

Der Nachteil der lokalen Nutzerverwaltung liegt in dem hohen administrativen Aufwand, der das Verwalten der einzelnen Nutzer erfordert. Bei einem System mit einer überschaubaren Anzahl von Nutzern ist dies meist noch einfach zu meistern. Sollen allerdings eine größere Anzahl von Nutzern, wie eine komplette Fakultät oder gar alle Mitglieder einer Universität, Zugriff mittels eigener UserIDs auf das System haben, ist der Aufwand erheblich größer.

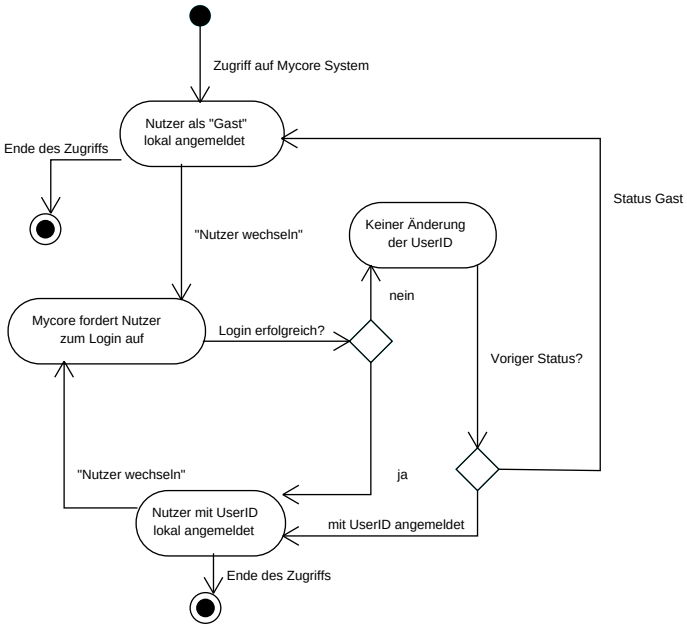

Abbildung 4.4: Ablaufdiagramm der Mycore Nutzerverwaltung

Im Diagramm (Abbildung 4.4) sind die einzelnen Schritte bis zum eingeloggten Nutzer schematisch dargestellt. Zunächst ist der Nutzer zu Beginn des Zugriffs auf Mycore Gast

Nutzer. Will er sich darauf mit seiner UserID anmelden, so wählt er „Benutzer wechseln“. Wird der Vorgang des Einloggens erfolgreich abgeschlossen, so ist er mit der eingegeben UserID im System angemeldet. Anderenfalls ist der Nutzer weiterhin als Gast im System geführt und es wird eine entsprechende Fehlermeldung angezeigt.

Der eingeloggte Nutzer erreicht im Schaubild einen Endzustand. Dies ist das Ziel dieses und der folgenden Diagramme. Der Nutzer kann natürlich erneut den Punkt „Nutzer wechseln“ wählen. Oder es sind Nutzer denkbar, die nur den Status „Gast“ in einer Sitzung haben. Das Ziel dieses und der folgenden Diagramme ist aber, die Aktionen eines Nutzers zu verfolgen, der sich mit seiner UserID im Mycore System anmeldet.

4.4 Mycore durch Shibboleth geschützt

Wie ist es nun möglich, Mycore durch Shibboleth sinnvoll zu schützen? Ziel ist es zunächst die interne Nutzerverwaltung von Mycore mit Shibboleth und den Nutzern des IdPs zu verknüpfen. Dadurch wäre es sehr einfach, beispielsweise allen Mitgliedern einer Universität den Zugriff auf ein Mycore System zu gewähren, falls ein IdP vorhanden ist, der alle diese Nutzer enthält. Damit wird der Aufwand für den Administrator sehr viel geringer, da neue Nutzer automatisch beim IdP eingetragen werden.

Technisch gesehen ist der Aufbau des Systems ähnlich zu dem Beispiel im vorigen Kapitel. Der Apache Server fungiert wiederum als Anlaufstelle für Anfragen des Webbrowsers des Nutzers. Auf den Tomcat Server, der die Mycore Anwendung enthält, kann wie zuvor nur vom lokalen Apache Server aus zugegriffen werden.

Im Folgenden geht es zunächst darum, die Nutzerverwaltung von Mycore durch den Einsatz von Shibboleth zu erweitern. Dabei gibt es immer mehrere Möglichkeiten der Realisierung, wobei jede üblicherweise einige Fragen und Probleme aufwirft. Zunächst wird versucht Shibboleth mit und ohne Lazy Session mit Mycore zu verknüpfen. Dabei soll die Realisierung noch so einfach wie möglich bleiben. Danach werden noch einige Variationen oder Erweiterungen der beiden Möglichkeiten besprochen.

4.4.1 Schutz durch Shibboleth ohne Lazy Session

Zunächst wird Mycore durch den Einsatz Shibboleths ohne Lazy Session geschützt. Der Zugriff wird ohne Einschränkung nur nach einer Authentifizierung beim IdP gewährt. Jede Anfrage eines Nutzers an das Mycore System wird automatisch zum IdP weitergeleitet, an dem sich der Nutzer einloggen muss. Es werden dem Nutzer somit zunächst keine Informationen über die konkrete Mycore Anwendung, die er gerade aufrufen möchte, übermittelt.

In Abbildung 4.5 ist der modifizierte Ablauf des Mycore Systems zu sehen. Dabei sind die Punkte mit dunklerem Hintergrund neu hinzugekommen. Wie man erkennen kann,

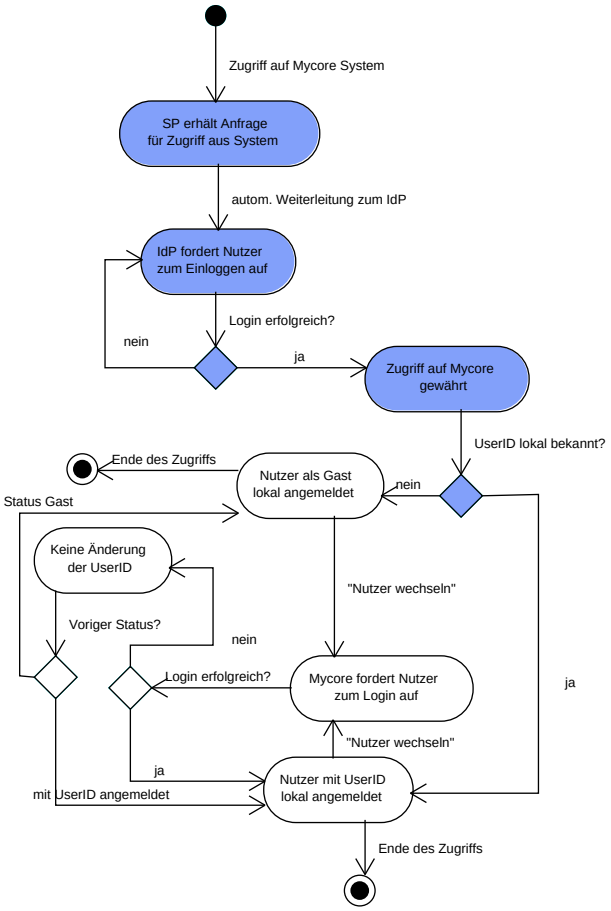

Abbildung 4.5: Mycore Nutzerverwaltung und Shibboleth ohne Lazy Session

hat der Nutzer keine Wahl und wird automatisch vom IdP zum Einloggen aufgefordert. Die Aufforderung zum Anmelden wird wiederholt, falls die Authentifizierung scheitern sollte. Der Zugriff auf das System bleibt dem Nutzer grundsätzlich verwehrt, falls er keinen Account beim IdP besitzt.

Nach einem Login beim IdP leitet dieser normalerweise die UserID an den SP weiter. Dieser fügt diese dann in den HTTP Header ein, der von Mycore aus ausgelesen werden kann. Im Mycore System kann also auf die UserID des Nutzers, unter der er beim IdP gespeichert ist, zugegriffen werden. Mit Hilfe dieser UserID kann nun überprüft werden, ob der Nutzer im lokalen System bekannt ist. Sollte dies der Fall sein, so wird er automatisch von der Nutzerverwaltung von Mycore als eingeloggt betrachtet. Sollte die UserID nicht in der lokalen Datenbank vorhanden sein, oder der IdP hat die UserID nicht übermittelt, so erhält der Nutzer den Status „Gast“.

Dabei hat der Gast Nutzer einen höheren Status gegenüber einem System ohne Shib-

boleth. Es haben nur derjenigen Nutzer Zugriff, die sich erfolgreich über den IdP angemeldet haben. Somit ist der Gast Nutzer nicht ein beliebiger Nutzer, sondern er kommt aus der Gruppe der Nutzer, die über einen Account beim IdP verfügen. Damit ist es eventuell sinnvoll, dem Gast Nutzer mehr Rechte zu verleihen, als es in einem System ohne Shibboleth der Fall ist.

Will der Nutzer sich mit einer anderen UserID anmelden, so hat er über die normale Funktionalität von Mycore dazu die Möglichkeit. Dies kann der Fall sein, falls die UserID beim IdP und im Mycore System nicht übereinstimmen. Der Nutzer muss sich in diesem Fall zweimal einloggen, solange nicht die gleiche UserID verwendet wird.

Wichtig ist auch, dass nicht fälschlicherweise ein Nutzer vom IdP in der lokalen Nutzerdatenbank gefunden wird, obwohl es sich nicht um den selben Nutzer handelt. Dies kann erreicht werden, indem alle lokalen Nutzer im Mycore System einen Namen erhalten, der i.d.R. nicht in einem IdP vorhanden ist. Ein Ansatz ist, jeden lokalen Nutzer mit dem Präfix „MCR“ beginnen zu lassen. Dadurch kann es zu keiner ungewollten Übereinstimmung von UserIDs kommen. Nur diejenigen Nutzer ohne den Präfix werden dann nach erfolgreicher Übertragung der UserID des IdPs automatisch als eingeloggt angesehen.

Der größte Vorteil von Shibboleth für Mycore in dieser Variante ist das grundsätzliche Blocken von jedem Zugriff auf das System. Dies ist im Mycore System bisher nicht möglich. Möchte ein neuer Nutzer sich über das System informieren, so benötigt er mindestens einen Account beim entsprechenden IdP. Den Gast Nutzer im ursprünglichen Sinne gibt er daher nicht mehr. Folglich gibt es ebenso keinen unbekannten bzw. anonymen Nutzer mehr. Jeder potentielle Nutzer muss sich mit seinem Account beim IdP als erstes erfolgreich anmelden. Dadurch erreicht der Schutz einer Mycore Anwendung eine neue Qualität.

4.4.2 Schutz durch Shibboleth mit Lazy Session

Eine weitere Möglichkeit des Schutzes bietet die Lazy Session bei Shibboleth. Dabei hält sich Shibboleth zunächst zurück und der Zugriff des Nutzers auf das Mycore System wird erst einmal zugelassen. Zu einem späteren Zeitpunkt kann der SP dann aufgefordert werden, den Nutzer beim IdP zu authentifizieren. Dies geschieht üblicherweise durch einen speziellen Link, der den Login über den angeschlossenen IdP aufruft.

Der Ablauf wird in dem Diagramm der Abbildung 4.6 skizziert. Wiederum sind neue Aspekte im Ablauf durch einen dunkleren Hintergrund hervorgehoben. Das System verhält sich zunächst identisch zu einem ungeschützten System. Der Nutzer erhält den Zugriff auf die Anwendung und ist Gast Nutzer. Somit kann er sich ohne Einschränkung über den möglichen Inhalt oder die Art der digitalen Bibliothek informieren. Ebenso ist das Wechseln des Nutzers wie gewohnt möglich. Ist der lokale Login erfolgreich, so ist der Nutzer mit der eingegebenen UserID angemeldet.

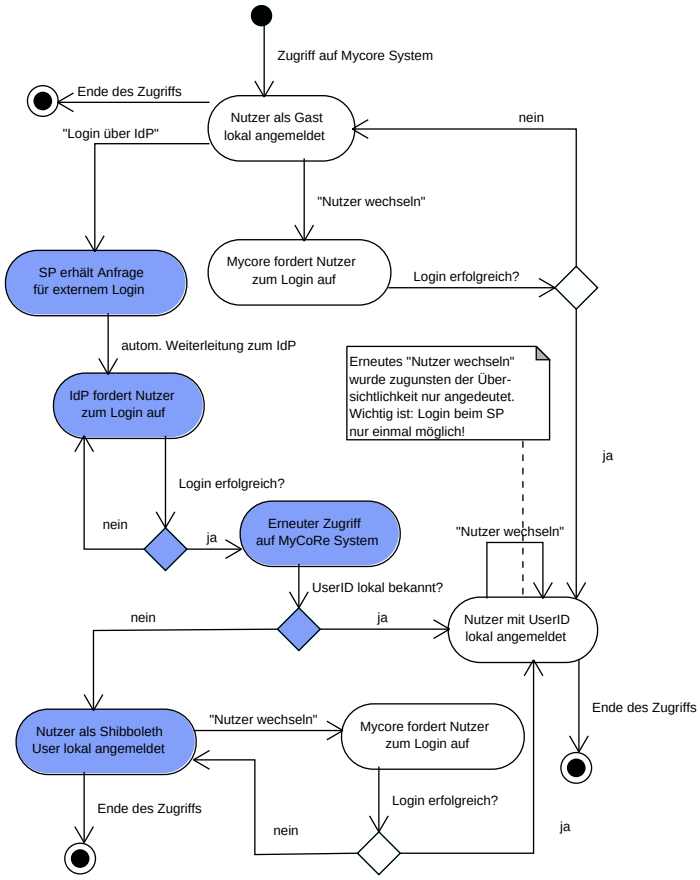

Abbildung 4.6: Mycore Nutzerverwaltung und Shibboleth mit Lazy Session

Shibboleth selbst wird nun über eine zusätzliche Möglichkeit zum Einloggen realisiert. Dazu muss in Mycore ein weiterer Login-Button platziert werden. Dieser sollte mit einem präzisen Namen versehen werden, und für den Nutzer ausreichend dokumentiert sein. Der Nutzer hat nun zwei Möglichkeiten sich einzuloggen. Einerseits über das interne System von Mycore, andererseits über den IdP mittels Shibboleth. Der Nutzer sollte den Unterschied erkennen können, um die für ihn geeignete Variante auswählen zu können.

Wählt der Nutzer den Login über den IdP, so ergibt sich ein ähnlicher Ablauf zum Shibboleth ohne Lazy Session. Der SP leitet die Anfrage an den IdP weiter. Beim IdP geht es nur nach einem erfolgreichen Anmelden weiter. Sollte der Login beim IdP nicht möglich sein, beispielsweise da der Nutzer keinen Account beim IdP oder sein Passwort vergessen hat, so muss er die URL der Mycore Anwendung neu in den Browser eingeben oder die entsprechenden Seiten zurückblättern.

Ist der Login über den IdP erfolgreich abgeschlossen, so übermittelt der IdP die User-ID an das Mycore System. Wiederum wird nach der UserID in der lokalen Datenbank

gesucht. Ist die übermittelte UserID vorhanden, wird der Nutzer automatisch als eingeloggt angesehen. Dieser Ablauf ist identisch zur Variante mit Shibboleth ohne Lazy Session.

Wird die UserID nicht lokal gefunden, ist es nicht ratsam, dem Nutzer den Status Gast zu verleihen. Damit würden zwei Gruppen von Nutzern nicht unterschieden werden. Ein Teil der Nutzer sind diejenigen, die auf die Seite zugreifen, ohne sich eingeloggt zu haben. Die zweite Gruppe enthält die Nutzer, die sich über den IdP angemeldet haben und keinen Account in Mycore besitzen. Daher sollten die zuletzt genannten Nutzer nicht den Status „Gast“ erhalten. Sie sind einen neuer Typ von Nutzer, der im folgenden Shibboleth Nutzer genannt wird. Der Shibboleth Nutzer ist also ein Nutzer, der sich über den IdP erfolgreich angemeldet hat, aber dessen UserID nicht in der lokalen Datenbank von Mycore gefunden wird. Der Shibboleth Nutzer kann nun je nach Wunsch mit anderen Rechten als der Gast Nutzer ausgestattet werden, um dem besonderen Status Rechnung zu tragen.

Der Shibboleth Nutzer kann wiederum seinen Status ändern, indem er sich lokal mit einer anderen UserID einloggt. Er behält natürlich seinen Status, falls der Versuch des lokalen Logins scheitern sollte. Aber man kann erkennen, dass kein weiterer Login beim IdP möglich ist. Dies ist nur möglich, falls der Nutzer sich in der aktuellen Sitzung noch nicht über den IdP authentifiziert hat.

Daraus ergibt sich ein wichtiger Punkt bei Shibboleth mit Lazy Session: Der Nutzer kann sich nur einmal über den IdP einloggen. Die Möglichkeit für den Nutzer, sich auszuloggen oder die UserID beim IdP zu wechseln, gibt es nicht. Diese Funktionalität bietet Shibboleth in der Version 1.3 nicht. Erst ab der zukünftigen Version 2.0 (siehe Abschnitt 4.12) wird dies möglich sein. Daher ergibt das Anlegen von mehrere Accounts für einen Nutzer beim IdP keinen Sinn. Werden für einen Nutzer zwei Accounts benötigt, beispielsweise für die Rollen als Administrator und „normaler“ Nutzer, so ist dies nur über das Anlegen von lokalen Nutzern möglich. Als einzige Möglichkeit sich wiederholt beim IdP einzuloggen, bleibt das manuelle Löschen der entsprechenden Cookies im Browser des Nutzers, was sicherlich nur für Testzwecke relevant ist.

4.5 Zwischenbilanz

Nach der Betrachtung beider Varianten von Shibboleth kann man erkennen, dass jeweils sowohl Vor- als auch Nachteile exitieren. Sicherlich gibt es verschiedene Anforderung an das Mycore System und somit Szenarien, indem jeweils eine der beiden Lösungen zu bevorzugen ist.

Beim Einsatz von Shibboleth ohne Lazy Session ergeben sich für den Nutzer weniger Änderungen zum gewohnten Mycore System. Der einzige Unterschied ist die Authentifizierung am IdP, bevor der Zugriff auf Mycore gewährt wird. Danach hat der Einsatz von Shibboleth keinen aktiven Einfluss mehr auf die Abläufe in Mycore. Allerdings entfällt

der klassische Gast Nutzer, was den Zugriff auf das System insgesamt restriktiver werden lässt.

Shibboleth ist mit Lazy Session etwas schwieriger in Mycore zu integrieren, da mehr Änderungen im System nötig sind. Da man sich beim IdP nur einmal einloggen kann, muss der Login über den IdP danach deaktiviert werden. Folglich muss für jeden Nutzer explizit gespeichert werden, ob er sich bereits über den IdP angemeldet hat. Sollte er es erneut probieren, muss er darüber informiert werden, dass dies nicht mehr möglich ist.

Nun werden noch einige Variationen der zuvor gezeigten Lösungen diskutiert.

4.6 Lazy Session mit einem Login Button

Ein Problem der Lazy Session von Shibboleth mit Mycore ist die Trennung des lokalen Logins von dem über Shibboleth. Für den Nutzer intuitiver und einfacher wäre, nur eine einzige Möglichkeit zum Einloggen über „Nutzer wechseln“ zu besitzen. Die Programmlogik entscheidet dann, welche Art von Login gewählt wird. Ein prinzipielles Problem dabei ist, dass Shibboleth die komplette Kontrolle über den Login-Prozess hat. Die einzige Möglichkeit zur Authentifizierung ist, die Anfrage an den SP zu stellen, dass der Nutzer sich über den IdP anmelden möchte. Eine Übermittlung von UserID und Passwort zur Authentifizierung ist nicht möglich.

Abbildung 4.7 zeigt den schematischen Ablauf für eine Realisierung durch einen einzigen Login Button. Wie man erkennen kann, meldet sich der Nutzer zunächst wie in Mycore ohne Shibboleth über „Nutzer wechseln“ an. Gibt der Nutzer eine lokal vorhandene UserID und ein entsprechendes Passwort an, so ist er angemeldet und es ergibt sich kein Unterschied zu der Version ohne Shibboleth.

Meldet das System einen erfolgreichen Login Versuch, bieten sich zwei Möglichkeiten. Ein erster Weg ist, den Nutzer direkt zum IdP weiterzuleiten. Dort kann er sich einloggen und wird anschließend vom System als Shibboleth Nutzer geführt. Zu Überprüfen, ob die lokale UserID übereinstimmt, ist nur sinnvoll, wenn der Nutzer beim IdP eine unterschiedliche UserID gewählt hat. Dies sollte aber nicht der Fall sein, da der Nutzer sich sofort mit der richtigen UserID im System einloggen kann.

Was ist der große Nachteil dieser Variante? Der Nutzer muss beim Login über den IdP seine UserID und das Passwort zweimal eingeben. Dies ist nicht sehr komfortabel und für den Nutzer auch eventuell nicht nachvollziehbar. Ein zusätzliches Problem tritt auf, wenn der Nutzer sich beim lokalen Login vertippt hat. Er wird zum IdP weitergeleitet, ohne dies zu wollen. Daher muss er die entsprechende Seite im Browser zurückblättern. Aufgrund dieses Problems ist diese Variante eher als nicht praktikabel einzustufen.

Eine Alternative zur vorigen Lösung wäre folgende: Um dem Nutzer die Möglichkeit zu geben, seine Eingaben zu korrigieren, kann man die automatische Weiterleitung zum IdP deaktivieren. Nun erhält der Nutzer einen Link zum externen Anmelden und gleichzeitig

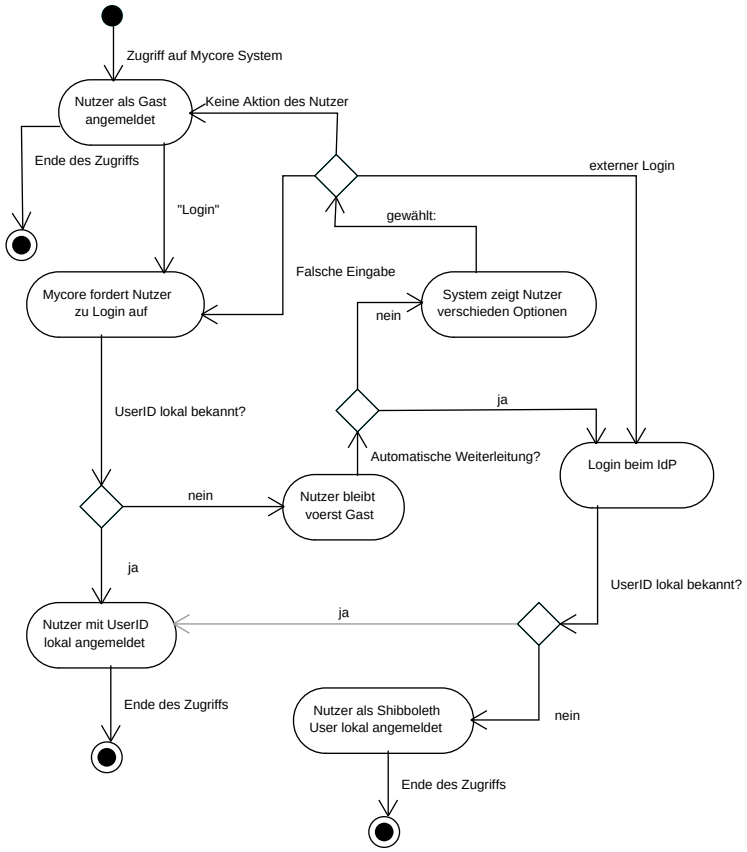

Abbildung 4.7: Shibboleth mit Lazy Session in Kombination mit einem Login Button

eine Eingabemaske für UserID und Passwort von Mycore, um sich lokal einzuloggen. Sollte der Nutzer sich vertippt haben, so kann er den lokalen Login erneut versuchen. Oder er wählt den externen Login über den IdP. Diese Variante ist aber streng genommen nicht ein Button für die Anmeldung, sondern eigentlich wiederum zwei Buttons, nur etwas anders logisch verknüpft.

Als Ergebnis lässt sich feststellen, dass der Reduzierung auf einen Button einige konzeptionelle Schwächen in das System bringt. Grundsätzlich wäre ein Button für den Nutzer verständlicher zu vermitteln, aber die Probleme, die sich damit ergeben, verhindern eine praktikable Lösung. Besser ist wahrscheinlich die Lösung mit grundsätzlich zwei verschieden Möglichkeiten zum Login - lokal und über den IdP.

4.7 Versteckter Lokaler Login

Um dem Nutzer mehr Klarheit bei der Auswahl des für ihn geeigneten Logins zu verschaffen, gibt es die Möglichkeit, den lokalen Login zu „verstecken“. Damit ist gemeint, dass der lokale Login nicht von jeder Seite über die Menüleiste zu erreichen ist. Er könnte unter einem in der Menüstruktur tiefer gelegenen Menüpunkt „Administration“ oder „Bereich für Experten“ platziert werden. Dadurch befindet sich in der Variante Lazy Session in der Menüleiste nur noch der Login-Button für das Anmelden über den IdP. Bei Shibboleth ohne Lazy Session ist ein Login-Button in der Menüleiste nicht nötig, da jeder Nutzer sich zunächst beim IdP erfolgreich einloggen muss.

Wie sehen nun die günstigen Rahmenbedingungen für dieses Szenario aus? Sicherlich sollte die Gruppe, die sich lokal anmelden muss, eher klein sein. Dies wären dann nur Nutzer, die administrative Aufgaben im System übernehmen. Davon gibt es in einem Mycore System üblicherweise nur sehr wenige. Diesen Administratoren muss dann gezeigt werden, an welcher Stelle sich der lokale Login befindet. Alle anderen Nutzer benötigen einen Account beim IdP. Werden zusätzlich bestimme Rechte benötigt, kann der entsprechende Nutzer lokal angelegt werden, und die UserID wird dann entsprechend verglichen. Nach dem Anmelden über den IdP ist der Nutzer somit mit seiner UserID im System angemeldet. Alle anderen Nutzer erhalten wie gewohnt den Status Shibboleth Nutzer, falls sie sich erfolgreich beim IdP angemeldet haben.

Sollten nicht alle Nutzer außer den Administratoren über einen Account beim IdP verfügen, ist diese Lösung nicht zu empfehlen. Einem einfachen Nutzer ist es sicherlich schwer zu vermitteln, dass er sich nicht über den offensichtlichen Login am System anmelden kann.

Diese Variante vereinfacht das System für den Nutzer. Er wählt nun automatisch den externen Login. Es müssen nur lokale Nutzer angelegt werden, die mit speziellen Rechten ausgestattet sind - wie beispielsweise Autoren oder Lektoren. Diese können dann gleichzeitig einen lokalen und einen Account beim IdP besitzen. Sollte die UserID übereinstimmen, führt der Login beim IdP dazu, dass der Nutzer lokal eingeloggt ist. Wichtig ist natürlich, dass der lokale Account ebenso ein Passwort benötigt, was unterschiedlich zu jenem beim IdP sein kann. Alle anderen Nutzer können mit dem IdP verwaltet werden. Dadurch ergibt sich insgesamt eine Reduzierung der administrativen Aufgaben für das Mycore System.

4.8 Die Schwäche des Shibboleth Nutzers

Wie bereits erwähnt ist, der Shibboleth Nutzer ein neuer Nutzer, der durch den Einsatz von Shibboleth in Mycore benötigt wird. Jeder Nutzer, der sich erfolgreich am IdP anmeldet und dessen User-ID nicht lokal gefunden wird, wird zum Shibboleth Nutzer.

Greift ein Nutzer auf ein Mycore System ohne Shibboleth zu, so ist er zunächst Gast Nutzer. Er ist also noch anonym und seine Aktionen können nicht einer Person zugeschrieben werden. Dies ist aber kein Problem, da der Gast sowieso nur eine Art passiven Status innehat. Er kann sich Inhalte ansehen, ohne jedoch selbst etwas an dem System zu verändern. Daher ist seine Anonymität für das Mycore System nicht problematisch.

Anders verhält sich dies nun bei einem Nutzer, der sich über den IdP angemeldet hat, aber keinen lokalen Account besitzt. Dieser ist bei Shibboleth ohne Lazy Session Gast Nutzer und bei Lazy Session Shibboleth Nutzer. Der Nutzer bleibt anonym, obwohl er dies de facto nicht ist. Die User-ID wird normalerweise vom IdP übertragen. Er kann also eindeutig identifiziert werden. Aber mit dem Status des Shibboleth Nutzers wird dies wieder verworfen.

Daher folgt eine signifikante Einschränkung des Shibboleth Nutzers. Er darf auch nur passive Funktionen ausführen, wie beispielsweise das Lesen von Informationen auf einer Seite oder der Zugriff auf Dokumente im Server. Führt der Nutzer aktive Aktionen aus, wirft dies einige Probleme auf. Was passiert, wenn zwei Shibboleth Nutzer ein Dokument einstellen? Man kann zum einen die Aktion nicht eindeutig zuweisen und zum anderen sind die Nutzer selbst möglicherweise verwirrt, weil der Shibboleth Nutzer weitere Aktionen neben den eigenen im System vornimmt.

Bietet das Mycore System die Möglichkeit, persönliche Einstellungen zu speichern, so ist dies als Shibboleth Nutzer nicht möglich. Wieder verhindert die Tatsache, dass der Nutzer anonym bleibt, die Funktion des Systems.

Durch den Shibboleth Nutzer wird der Schutz von Mycore durch Shibboleth zwar einfacher, aber es ergeben sich die oben genannten Schwächen. Daher sollte die Variante mit dem Shibboleth Nutzer nur dann verwendet werden, wenn die Anonymität der Nutzer kein Problem darstellt. Sollen die Aktionen der Nutzer entsprechend zugeordnet werden, müssten mit der bisherigen Lösung alle Nutzer zusätzlich lokal angelegt werden. Dies würde aber bedeuten, dass Shibboleth nicht mehr benötigt wird, da sich jeder Nutzer auch lokal einloggen kann.

4.9 Dynamisch generierter Nutzer

Wie kann man nun die Probleme, die sich durch den Einsatz des Shibboleth Nutzers ergeben, verhindern? Ein Lösung ist ihn einfach abzuschaffen. Damit stellt sich die Frage, was mit den Nutzern passiert, die sich beim IdP anmelden und nicht in der lokalen Datenbank bekannt sind? Diese müssen automatisch erzeugt werden. Dabei gibt es zwei unterschiedliche Möglichkeiten, diese Nutzer zu erzeugen.

4.9.1 Temporäre Nutzer

Der Nutzer, dessen UserID vom IdP lokal nicht vorhanden ist, wird nun temporär erzeugt. Dies bedeutet, dass die UserID vom IdP als lokale UserID verwendet wird. Diese muss für einen temporären Nutzer zwingend vom IdP übertragen werden, sonst ist das System auf diese Art nicht zu realisieren. Der Nutzer ist nicht mehr als Shibboleth Nutzer, sondern mit seiner UserID im Mycore System eingeloggt. Er ist damit nicht mehr anonym und seine Aktionen können eindeutig zugeordnet werden.

Der Nutzer hat dadurch die Möglichkeit, aktiv Dokumente einzustellen oder andere Aktionen auszuführen, die einer Person zugeordnet werden müssen. Alle temporär erzeugten Nutzer haben allerdings die gleichen Rechte. Benötigt ein Nutzer spezielle Rechte, so muss er über eine lokale UserID im Mycore System verfügen.

Der temporäre Nutzer hat einige gravierende Nachteile. Greift der Nutzer zu einem späteren Zeitpunkt erneut auf das System zu, so wird er nach den Einloggen über den IdP wieder erzeugt. Daher ist es nicht möglich, die persönlichen Einstellungen des Nutzers zu speichern.

Hat der Nutzer neu Dokumente erzeugt, sind diese zwar noch seiner UserID zuzuordnen, aber den Nutzer selbst gibt es nicht mehr. Will ein Autor beispielsweise seinem Werk einen Lektor zuweisen, so ist dies nicht möglich, da dieser nur temporär erzeugt wird. Dies führt dazu, dass Mycore nicht mehr einwandfrei funktionieren kann. Der temporäre Nutzer ist keine realistische Option beim Einsatz von Shibboleth. Er schafft insgesamt eher ein größeres Problem, als dass er einen Vorteil durch die Lösung des Problems des Shibboleth Nutzers anbietet.

4.9.2 Benutzer lokal speichern

Die zweite Möglichkeit besteht darin, die Nutzer nicht temporär zu erzeugen, sondern automatisch in der lokalen Datenbank anzulegen, falls diese nicht lokal gefunden werden. Dadurch werden unbekannte Nutzer automatisch angelegt, vorausgesetzt sie authentifizieren sich beim IdP.

Der Vorteil besteht darin, dass Informationen über den Nutzer lokal gespeichert werden können. Loggt er sich zu einem späteren Zeitpunkt wieder ein, ist er im System bereits bekannt. Somit ist es auch nicht mehr nötig, wiederholt den selben Nutzer zu erzeugen.

Diese Variante wirft aber einige neue Probleme auf. Zunächst einmal wird jeder Nutzer in der Datenbank angelegt. Dies kann bei einem IdP mit vielen Nutzern zu einer großen Anzahl von automatisch erzeugten Nutzern führen. Als Gegenmaßnahme könnte man Nutzer, die sich über einen längeren Zeitraum nicht eingeloggt haben, automatisch wieder aus der Datenbank entfernen. Dies führt aber zu einem quasi temporären Nutzer, was alle zuvor genannten Probleme aufwirft.

Ein weiteres Problem sind Nutzer, die ihren Account beim IdP verlieren. Diese sind weiterhin in der Datenbank des Mycore Systems vorhanden. Wie kann nun das Mycore System erfahren, dass diese Nutzer nicht weiter existieren? Dies ist nur mühsam per Hand abzugleichen, was nicht praktikabel erscheint. Also bleiben die automatisch erzeugten Nutzer im System erhalten.

Wichtig ist auch, dass eine UserID, die wieder frei wird, nicht an einen neuen Nutzer vergeben wird. Sonst kommt es zu einem Konflikt im Mycore System, da dieses davon keine Kenntnis haben kann. Daher muss dies explizit vom IdP gefordert werden. Üblicherweise werden UserIDs von gelöschten Accounts über einen gewissen Zeitraum nicht erneut verwendet.

Insgesamt ergeben sich einige Probleme bei dieser Variante. Die Nutzerverwaltung beim IdP und lokal in Mycore verläuft parallel. Dies erzeugt eine Redundanz, die nicht sinnvoll ist, da beide Systeme nicht automatisch synchronisiert werden können.

4.10 Einsatz von Attributen

Beide Arten von Shibboleth lassen sich durch den Einsatz von Attributen erweitern. Was genau Attribute sind und wie diese funktionieren, wird in den Kapiteln 2 und 3 ausführlich erklärt. Attribute sind zusätzliche Informationen über einen Nutzer, wie z.B. die UserID oder Email-Adresse. Diese werden vom IdP zur Verfügung gestellt und bei einer erfolgreichen Authentifizierung an den SP weitergeleitet. Dieser überprüft dann die Einstellungen für die Attribute. Sollten Attribute für die Anwendung in den Einstellungen genannt werden, so gibt Shibboleth diese über den HTTP-Header an Mycore weiter. In Mycore können diese dann als eine Art Umgebungsvariable ausgelesen werden.

Welche Attribute werden nun zur Verfügung gestellt? Dies ist vom konkreten Shibboleth System abhängig. Es ist grundsätzlich nicht garantiert, dass bestimmte Attribute vorhanden sind. Jeder IdP kann selbst bestimmen, welche Attribute er anbietet. Vielleicht ist er auch durch die Art der angeschlossenen Nutzerdatenbank an ein gewisses Schema gebunden. Man kann beim Zugriff auf Mycore über Shibboleth nur von einigen Standard Attributen ausgehen, die jeder IdP höchst wahrscheinlich anbietet. Diese sind sicherlich die UserID, der Name und eventuell die Rolle des Nutzers, wie beispielsweise „Student“. Über weitere Attribute kann keine verbindliche Aussage getroffen werden. Mann muss den Einsatz von Attributen daher immer vom jeweils eingesetzten Shibboleth System bzw. dessen Attributen abhängig machen.

Aufgrund dieser Tatsachen muss der Einsatz von Attributen variabel bleiben. Jedes Mycore System, das Attribute einsetzen möchte, muss auf die verfügbaren Informationen vom IdP angepasst werden. Damit können verschiedene Rechte eines Nutzers in Mycore durch den Einsatz von Attributen abgebildet werden.

Man könnte sich nun beispielsweise vorstellen, dass der IdP alle Studenten einer Univer-

sität enthält. Neben der User-ID sind noch folgende Attribute verfügbar: Die Fakultät und der Name eines Projekts, an dem der Student teilnimmt. Sollte das zuletzt genannte Attribut nicht verfügbar sein, wird davon ausgegangen, dass er momentan kein Projekt belegt.

In dem Mycore System können nun zwei Typen von Bereichen kreiert werden, die nur für bestimmte Nutzer zugänglich sind. Bei jeder Fakultät gibt es Dokumente, die nur für die eigenen Studenten zugänglich sind. Dies kann über das entsprechende Attribut zugesichert werden. Weiterhin gibt es noch Dokumente, auf die nur Mitglieder eines bestimmten Projekts Zugriff haben sollen. Wiederum wird das entsprechende Attribut überprüft und der Nutzer ist nur dann autorisiert, falls er sich im richtigen Projekt befindet.

Wie verhält es sich nun mit der Nutzerverwaltung von Mycore bei diesem Beispiel? Um die genannten Probleme mit dem dynamisch erzeugten Nutzer zu umgehen, erfährt der Einsatz von Shibboleth einen qualitativen Sprung. Shibboleth, erweitert durch Attribute, ist nur sinnvoll, wenn es die Nutzerverwaltung von Mycore komplett ersetzt. Es gibt also nur noch Nutzer, die sich beim IdP eingeloggt haben. Die Rechte der einzelnen Nutzer werden von System automatisch durch die Attribute generiert. Sie dürfen nicht gespeichert werden, denn falls die Attribute eines Nutzers sich ändern, ändern sich möglicherweise auch seine Rechte im Mycore System.

Diese vollständige Auslagerung der Nutzerverwaltung inklusive ihrer Rechte wirft jedoch einige neue Probleme auf. Was passiert mit komplexeren Abläufen in einem Mycore System? Solch ein Ablauf könnte beispielsweise einen Autor und den ihm zugewiesenen Lektor beinhalten. So eine Art von Struktur kann wohl kaum mit Attributen vom IdP abgebildet werden. Also müssen wiederum zusätzliche Informationen zum Nutzer gespeichert werden, was in die Sackgasse des dynamisch erzeugten Nutzers führt.

Da es keine lokale Nutzerverwaltung mehr gibt, müssen alle Accounts beim IdP angelegt werden. Diese ist für Accounts von Administratoren auch schwer realisierbar. Sie haben meist einen speziellen Status für administrative Aufgaben. Wie soll dieser dann von den Attributen des IdPs erzeugt werden? Das ist wohl nur über ein spezielles Attribut beim IdP zu erreichen.

Der Einsatz von Attributen für Mycore ist nur dann sinnvoll, wenn alle Nutzer beim IdP bekannt sind und es nur wenige Gruppen von Nutzern mit unterschiedlichen Rechten gibt. Die Nutzer können dann über die Attribute mit entsprechenden Rechten ausgestattet werden. Loggt ein Nutzer sich über den IdP ein und seine UserID ist lokal unbekannt, wird diese automatisch erzeugt und er ist lokal angemeldet. Andernfalls besteht die UserID bereits und er ist ebenso eingeloggt.

Die verschiedenen Rechte werden über die Gruppenzughörigkeit der Nutzer abgebildet. Ist ein bestimmtes Attribut vorhanden, wird der Nutzer einer Gruppe zugewiesen. Als Mitglied dieser Gruppe hat der Nutzer dann entsprechende Rechte im Mycore System. Ein Problem entsteht allerdings, wenn die Attribute des Nutzers sich ändern. Daher

müssen die Attribute nach dem Einloggen des Nutzers immer überprüft werden und gegebenenfalls die Gruppenzugehörigkeit geändert werden.

Die dynamische Zuweisung von Attributen kann möglicherweise auch auf das ACL System von Mycore angewendet werden. Dieses ermöglicht seit einiger Zeit auch die Zuweisung von Zugriffsrechten für Dokumente nach anderen Kriterien als UserID oder Gruppezugehörigkeit. Somit könnten die Attribute eines Nutzers auf gewisse Zugriffsrechte für die Dokumente abgebildet werden. Allerdings bleibt die Frage offen, ob die gewünschten Kategorien von Rechten durch die angebotenen Attribute des IdPs abgedeckt werden können. Zusätzlich sollte der Nutzer dennoch mit einer (automatisch erzeugten) UserID eingeloggt sein. Im Fall des Shibboleth Nutzers ergeben sich wiederum die zuvor (Unterkapitel 4.8) diskutierten Nachteile.

Für den automatisch erzeugten Nutzer, erweitert durch Attribute, gibt es sicherlich noch weiteren Bedarf für Untersuchungen. Vor allem ist zu klären, wie sich Attribute mit Hilfe des ACL Systems verbinden lassen. Allerdings bleibt als Vorgabe die dynamische Erzeugung der Rechte nach der Authentifizierung des Nutzers. Nur so kann sicher gestellt werden, dass der Nutzer nur Rechte entsprechend seiner Attribute besitzt.

4.11 Authentifizierung über die Kommandozeile

Mycore besitzt neben dem Zugang über einen Webbrowser auch noch andere Schnittstellen für den Zugriff auf das System. Diese Zugangswege bestehen über die Kommandozeile und zukünftig auch über Webservices. Speziell der Zugang über die Kommandozeile lässt sich ebenso mit der Nutzerverwaltung steuern. Der Nutzer loggt sich also im Mycore System über die Kommandozeile ein und kann dann bestimmte Aktionen ausführen, wie beispielsweise neue Dokumente in das System einstellen.

Es stellt sich die Frage, ob dieser Zugang auch mit Shibboleth geschützt werden kann. Dies ist nicht möglich. Shibboleth ist von seinem Konzept her nur für den Einsatz im Rahmen eines Webservers geeignet. Er kann also nur Webanwendungen schützen. Ein Schutz außerhalb dieses Bereichs ist aufgrund der Shibboleth eigenen Architektur nicht möglich.

Dieser Punkt schränkt den Vorteil von Shibboleth für Mycore weiter ein. Jeder Nutzer, der sich über die Kommandozeile in Mycore anmelden möchte, benötigt daher einen lokalen Account.

4.12 Shibboleth 2.0

Wie schon in den vorigen Abschnitten deutlich wurde, gibt es einige konzeptionelle Schwächen beim Einsatz von Shibboleth in Mycore. Die beiden kritischsten Punkte sind

der fehlende Logout und die Tatsache, dass Shibboleth nicht die Kontrolle über den Login Prozess abgeben kann. Daher bietet es sich an, ein wenig in die Zukunft zu schauen, um die Neuerungen von Shibboleth 2.0 unter die Lupe zu nehmen.

Shibboleth 2.0 ist noch nicht verabschiedet (Stand August 2006) und befindet sich momentan in der Implementierungsphase. Es basiert auf SAML 2.0, dass sich im Beta Status befindet. Ein spezieller Release Termin für Shibboleth 2.0 ist bisher noch nicht angekündigt.

Die Version 2.0 bringt eine ganze Reihe von Neuerungen mit sich. Der wichtigste Punkt ist eine neue Funktion namens Single Logout Service (SLO). Damit kann sich der Nutzer beim IdP abmelden bzw. ausloggen. Somit kann der Nutzer sich erneut beim IdP anmelden und ist in der Lage, sich nach seiner Sitzung am System abzumelden. Dies wäre für Mycore mit Shibboleth eine Verbesserung.

Für den SP ergeben sich einige Neuerungen. Er kann zwischen verschiedenen Arten der Authentifizierung wählen. Er kann dem IdP eine Liste aller akzeptierten Arten der Authentifizierung übergeben und dieser versucht dann eine vom Nutzer gewählte Methode anzuwenden. Dies lässt wohl aber weiterhin keine Eingabe der UserID und des Passworts auf Seiten des SPs zu.

4.13 Zusammenfassung

Was sind die Ergebnisse der Untersuchung des Einsatzes von Shibboleth? Es wird deutlich, dass der Einsatz von Shibboleth nur in dem Fall uneingeschränkt zu empfehlen ist, wenn die Ressource keine eigene Nutzerverwaltung besitzt oder diese „einfach“ abgeben kann. Im Falle von Mycore ist es jedoch recht schwierig Shibboleth einzusetzen. Sowohl Mycore als auch Shibboleth sind so konzipiert, dass sie jeweils die Kontrolle über die Nutzerverwaltung besitzen möchten. Dadurch kommt es zu einem Konflikt, da sich diese beiden Ansätze nicht zu einem homogenen System vereinbaren lassen.

Ein Beispiel daher wäre die beiden Login-Buttons, die sich nicht zu einem einzigen vereinigen lassen. Ebenfalls ist durch den Einsatz von Shibboleth mit Lazy Session für den Nutzer kein wiederholter Login während einer Sitzung über den IdP möglich.

Welche Szenarien sind am ehesten für den Einsatz von Shibboleth geeignet? Sicherlich sind dies einfachere Dokumentenserver ohne Publikationsprozesse. Bei dieser Art von System gibt es eine große Menge an Nutzern, die nur passiv agieren. Sie greifen nur auf Dokumente zu, ohne selbst welche einzustellen. Einige wenige Nutzer sind Lieferanten von neuen Dokumenten oder Administratoren für das System. In so einem Fall ist der Einsatz von Shibboleth sicherlich zu empfehlen. Der Shibboleth Nutzer dürfte für diesen Fall ausreichen. Alle Nutzer mit speziellen Rechten werden über einen lokalen Account abgedeckt. Diese können sich lokal anmelden, oder, falls sie einen entsprechenden Account beim IdP besitzen, sich über das Shibboleth System einloggen.

Handelt es sich jedoch bei dem Mycore System um einen Publikationsserver mit allen sich dadurch ergebenden Prozessen, so ist ein Einsatz doch eher kritisch zu bewerten. Jeder Nutzer muss mit einer UserID im Mycore System vorhanden sein, da jeder Nutzer spezielle Rechte benötigt. Dadurch bleibt nur eine Auslagerung der Authentifizierung an Shibboleth, ohne dadurch etwas einzusparen. Der Einsatz von Shibboleth bringt daher nur einen Mehraufwand, ohne eine Reduzierung des administrativen Aufwands zu erreichen.

Kapitel 5

Implementierung in Mycore

In diesem Kapitel geht es um die technische Realisierung von Shibboleth in Mycore. Wie kann man die vorher beschriebenen Konzepte in Mycore implementieren? Welche Klassen müssen verändert bzw. erstellt werden?

Mycore basiert, wie bereits im vorigen Kapitel erwähnt, auf Java Klassen mit einigen zusätzlichen Bibliotheken. Dabei werden für die Logik des Frontends hauptsächlich Servlets eingesetzt. Im Folgenden werden einige grundsätzliche Techniken im Rahmen von Java und der Servlet Technologie vorausgesetzt. Sollte man mit dem Einsatz von Servlets und der Technik dahinter nicht vertraut sein, ist auf das ausführliche Tutorial von SUN hinzuweisen [17].

Zunächst einmal werden einige Vorüberlegungen angestellt. Es gibt ein Konzept für Java, das sich mit Authentifizierung und Autorisierung beschäftigt. Dies wird zunächst kurz vorgestellt werden. Leider wird dieses Konzept, wie sich herausstellen wird, nicht zum Einsatz kommen werden. Des weiteren wird auf die Implementierung in Mycore eingegangen werden. Anschließend wird das Pressearchiv, ein Mycore System im prodiktiven Einsatz, vorgestellt werden. Dies wird zeigen, dass sich Shibboleth auch ohne Änderung am Mycore System unter gewissen Bedingungen sinnvoll einsetzen lässt.

5.1 JAAS

Das Wort JAAS ist die Abkürzung für Java Authentication und Authorization Service. Dabei handelt es sich um ein Klassenkonzept für Java, das sich mit der Autorisierung und Authentifizierung von Nutzern in einer beliebigen Java-Anwendung beschäftigt. Die Klassen sind alle in der JAVA Standard Edition Version 5[1] enthalten. Das Konzept ist daher schon ausgereift und kann als Standard für Java angesehen werden. Eine detaillierte Beschreibung zum Einsatz von JAAS ist in einem Tutorial zu finden [16]. Das

[1] java.sun.com/javase/

Prinzip, das hinter JAAS steckt und auch in anderen Bereichen zum Einsatz kommt, ist ausführlich in einem Artikel der Zeitschrift Linux-Magazin erklärt [13].

JAAS ist so aufgebaut, dass der Nutzer zunächst seine UserID und sein Passwort dem System mitteilt. Dann versucht JAAS den Nutzer zu authentifizieren, und zwar an verschieden angeschlossenen Systemen, die eine Authentifizierung des Nutzers anbieten. Für jedes System gibt es ein so genanntes Pluggable Authentification Module (PAM), das für die Authentifizierung zuständig ist. Die PAMs verfügen über eine Schnittstelle, über die sie von der Anwendung bzw. JAAS angesprochen werden können. Dies hat einen enormen Vorteil, da damit die Anwendung unabhängig von dem darunter liegenden Mechanismus der Authentifizierung bleibt. In der Anwendung gibt es nur einen Aufruf, der veranlasst, dass der Nutzer seine UserID und sein Passwort angibt. Nun greift eine Konfigurationsdatei, in der bestimmt wird, auf welche Art der Nutzer authentifiziert werden kann. Es folgen zwei Beispiele (Listing 5.1 und 5.2) einer solchen Konfiguration. In jedem gibt es zwei Module. Das eine Modul ermöglicht eine lokale Authentifizierung, das andere ist für einen Login über einen so genannten LDAP-Server zuständig. Auf LDAP wird genauer in Kapitel 6 eingegangen, und daher an dieser Stelle nicht näher erklärt.

```
1 MyLogin {
2
3   MyApplication.Jaas.LocalLogin Sufficient;
4   MyApplication.Jaas.LDAPLogin Required;
5
6 };
```

Listing 5.1: Beispielkonfiguration von JAAS (1)

```
1 MyLogin {
2
3   MyApplication.Jaas.LocalLogin Required;
4   MyApplication.Jaas.LDAPLogin Required;
5
6 };
```

Listing 5.2: Beispielkonfiguration von JAAS (2)

In einer Konfigurationsdatei können mehrere Möglichkeiten zur Authentifizierung erscheinen und auf verschiedene Arten verknüpft werden. In Listing 5.1 wird der Nutzer zuerst lokal in einer Datenbank gesucht. Stimmen UserID und Passwort überein, so meldet JAAS bereits eine erfolgreiche Authentifizierung. Das zweite Modul kommt in diesem Fall nicht zum Einsatz. Das Wort „Sufficient“ in Zeile 3 gibt an, dass der erfolgreiche Login in diesem Modul insgesamt ausreicht. Wird der Nutzer lokal nicht gefunden oder stimmt das Passwort nicht überein, gibt es eine weitere Alternative. Es wird nun versucht den Nutzer über einen LDAP Server zu authentifizieren. Nun sorgt ein „Required“ dafür, dass JAAS nur dann eine erfolgreiche Authentifizierung meldet, falls der Login über LDAP erfolgreich war. Andernfalls gilt der Versuch des Logins insgesamt als gescheitert. Man kann sich nun noch weitere Module vorstellen, um die der Login Prozess erweitert werden kann.

Die Konfiguration kann auch so gewählt werden (siehe Listing 5.2), dass der Nutzer an zwei verschiedenen Modulen authentifiziert werden muss. Man sieht nun, dass bei beiden Modulen das Wort „Required“ steht. Der Nutzer muss nun über die lokale Datenbank sowie über LDAP Server erfolgreich authentifiziert werden. Nur dann meldet JAAS ein positives Ergebnis.

Dies kann alles über die externe Konfigurationsdatei gewählt werden, ohne dass am Quellcode eine Zeile geändert werden muss. JAAS ist somit ein sehr flexibles Modell für ein System, dass die Funktionalität bietet, dass Nutzer, die an verschiedenen Orten einen Account haben, sich im System anmelden können. Die Entwicklung der eigentlichen Anwendung ist völlig losgelöst von den konkreten Klassen von JAAS. Soll eine neue Art von Authentifizierung realisiert werden, muss an dem Code der Anwendung nichts geändert werden, da alles modular konzipiert ist.

Ist das Modell von JAAS nun auf Mycore zu übertragen? Die Möglichkeit, den Prozess der Authentifizierung auszulagern und damit die Konfiguration einfacher zu gestalten, erscheint auf den ersten Blick als sinnvoll. Man könnte eine Modul für den lokalen Login sowie ein Modul für Shibboleth kreieren. Dann kann das System je nach Wunsch konfiguriert werden, ohne dass etwas an der Anwendung geändert werden muss.

Es gibt jedoch zwei zentrale Punkte, die gegen eine solche Lösung sprechen. Erstens ist es nicht möglich, bei Shibboleth die Kontrolle über den Prozess der Authentifizierung zu erhalten. Es ist auch nicht möglich, dem IdP eine UserID und ein Passwort zu senden, damit dieser die Nachricht über den Erfolg oder Misserfolg des Versuchs der Authentifizierung zurück gibt. Es gibt nur die Möglichkeit, den SP aufzufordern den Nutzer zu authentifizieren. Der zweite Grund liegt bei Mycore selbst. Die interne Nutzerverwaltung sowie die Klassen für die Anmeldung der Nutzer unterliegen einem eigenen Konzept. Wird das Modell von JAAS angewendet, so müssen einige Klassen von Mycore diesem Konzept angepasst werden. Dadurch wären grundlegende Änderungen nötig, die den Rahmen dieser Arbeit hier sprengen würde.

5.2 Shibboleth mit Mycore Konzept

Da sich das Konzept von JAAS aus oben genannten Gründen nicht übertragen lässt, muss nun eine alternative Lösung gefunden werden. Es bietet sich an, Shibboleth durch möglichst wenig Änderungen im Code von Mycore zu integrieren. Dabei wird versucht, das Konzept der bisherigen Nutzerverwaltung von Mycore zu übernehmen. Dieses basiert auf den Einsatz eines Servlets zur Interaktion mit dem Nutzer für die Authentifizierung. Die eigentliche Funktionalität der Nutzerverwaltung ist in eine Manager Klasse ausgelagert.

Für Shibboleth wird nun ein ähnlicher Ansatz gewählt. Es gibt ein Servlet für die Interaktion mit dem Shibboleth System. Zusätzlich gibt es eine Manager-Klasse, die weitere Funktionalität zu Verfügung stellt.

Folgende Klassen müssen neu erzeugt werden bzw. erhalten Änderungen:

- **MCRShibLoginServlet**: neu - dient zur Kommunikation mit dem Shibboleth System
- **MCRShibMgr**: neu - erweitertet Funktionalität für Shibboleth in Mycore
- **MCRStaticXMLServlet**: geändert - benötigt eine Abfrage für Shibboleth ohne Lazy Session
- **MCRSession**: geändert - erhält zusätzlich Speicherung eines erfolgreichen Shibboleth Logins
- **mycore.properties.private** geändert - zusätzliche Optionen für Shibboleth

Dies ist eine erste Übersicht, um den Aufwand der Realisierung abzuschätzen. Nun folgt eine ausführlichere Erklärung über die internen Abläufe in Mycore.

Der folgende Abschnitt beschäftigt sich mit den wichtigen Klassen des Mycore Systems. Es wird nicht auf alle Klassen von Mycore eingegangen. Es werden nur diejenigen Klassen gezeigt, die durch den Einsatz von Shibboleth verändert werden müssen oder mit diesen in einer engen Beziehung stehen. Als Einstieg in Mycore empfiehlt sich zunächst die Dokumentation des Projekts [5]. Speziell für diesen Abschnitt ist der „Programmers Guide“ eine erste Möglichkeit, Mycore von der technischen Seite kennen zu lernen.

5.2.1 Das MCRServlet

Die zentrale Klasse für den Frontend Bereich im System ist `MCRServlet`. In Abbildung 5.1 sind die Klasse `MCRServlet` und die wichtigsten Klassen, die mit ihr in irgend einer Art in Verbindung stehen, in einem Klassendiagramm dargestellt. `MCRServlet` ist abgeleitet von der Klasse `HTTPServlet`. Die Klasse `MCRServletJob` dient nur zur Kapselung der beiden wichtigen Parameter `HTTPServletRequest` und `HTTPServletResponse`, die ein Servlet nach dem Aufruf übergeben bekommt.

Über die Klasse `MCRConfiguration` wird die Konfiguration von Mycore gesteuert. Alle Optionen können in einer Datei über einzelne Parameter gewählt werden. Die Parameter sind aus unterschiedlichen Bereichen wie z.B. die Art der Datenbank oder bestimmte Einstellungen zu den Nutzern. Hier bietet sich ebenso an, die Einstellungen zu Shibboleth zu platzieren. Es folgt die Liste der Parameter und ihre Bedeutung:

- **MCR_Shib_Active**: Wird auf `true` gesetzt, falls Shibboleth in Mycore aktiviert wird - andernfalls auf `false`.
- **MCR_Shib_LazySession**: Bei `true` wird Shibboleth mit Lazy Session betrieben - ansonsten ohne Lazy Session.

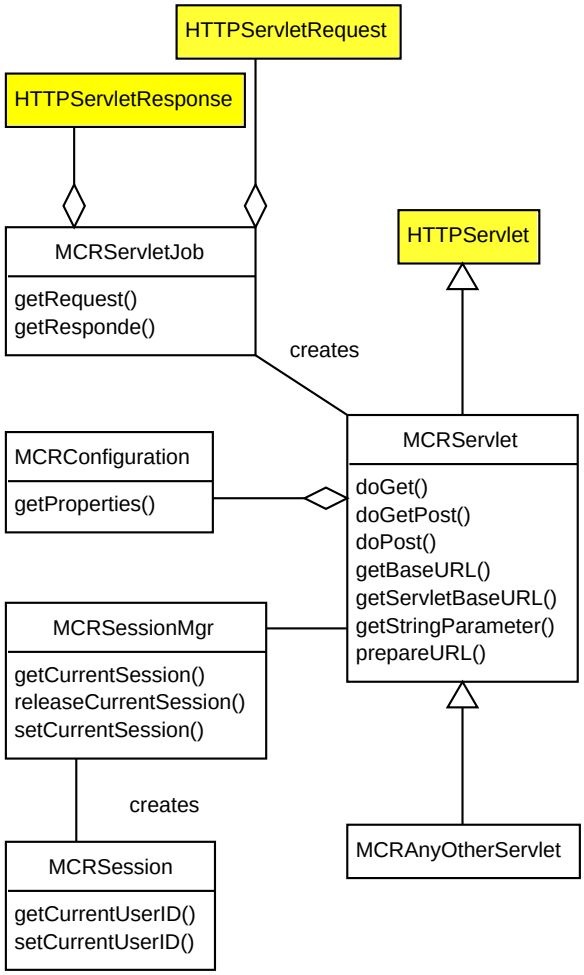

Abbildung 5.1: Übersicht zur Mycore Klasse MCRServlet

- **MCR_Shib_Header_UserName**: Name des HTTP-Headers, in dem die UserID vom Shibboleth System übergeben wird.
- **MCR_Shib_MatchLocalUser**: Bei `true` wird ein Nutzer, der sich am IdP angemeldet hat, in der lokalen Datenbank gesucht, und gegebenenfalls als lokal eingeloggt eingestuft - bei `false` ist die Funktion entsprechend deaktiviert.
- **MCR_Shib_UserName**: Sollte ein Nutzer sich über den IdP erfolgreich authetifizieren und nicht in der lokalen Datenbank gefunden werden, oder diese Funktion deaktiviert sein, dann wird er im Mycore System unter dieser UserID geführt.
- **MCR_Shib_GroupName**: Ist analog zur vorigen Einstellung und bestimmt den Namen der Gruppe.

Die Einstellungen befinden sich in der Datei `mycore.properties.private`. Eine Bei-

spielkonfiguration für ein aktiviertes Shibboleth mit Lazy Session, bei dem die Nutzer gemappt werden sollen und der Shibboleth Nutzer „ShibUser“ genannt wird, ist in Listing 5.3 vorhanden. Dabei wird die UserID über den HTTP-Header mit dem Namen „Shibboleth-UserID“ übergeben.

```
1  ...
2  ************************************************************
3  * Shibboleth                                               *
4  ************************************************************
5  MCR_Shib_Active=true
6  MCR_Shib_LazySession=true
7  MCR_Shib_Header_UserName=Shibboleth-UserID
8  MCR_Shib_MatchLocalUser=true
9  MCR_Shib_UserName=ShibUser
10 MCR_Shib_GroupName=ShibGroup
11 ...
```

Listing 5.3: Beispielkonfiguration von Shibboleth - mycore.properties.private

Auf die Parameter in Listing 5.3 kann über die Methode `getProperties` der Klasse `MCRConfiguration` zugegriffen werden. Somit stehen alle Einstellungen in der Klasse `MCRServlet` zur Verfügung.

Jeder Nutzer, der auf das Mycore System zugreift, bekommt eine Session vom System zugewiesen. Sie dient dazu, einen Zugriff auf eine Webanwendung eindeutig zuordnen zu können. Auch Mycore übernimmt dieses Konzept durch eine Klasse namens `MCRSession`. Jede Session hat eine eindeutige Nummer und jeder Nutzer ist daher einer bestimmten Session zugeordnet. Die Klasse `MCRSession` enthält einige Informationen über den Nutzer. Wichtig in Bezug auf Shibboleth ist vor allem die in der Session gespeicherte UserID.

Das `MCRServlet` erzeugt bei einem neuen Nutzer ein Objekt der Klasse `MCRSession` allerdings nicht selbst. Die Logik der Verwaltung der Sessions ist in der Klasse `MCRSessionMgr` vorhanden. Sie kümmert sich darum, neue Sessions zu erzeugen oder auf bestehende zuzugreifen. Die Klasse `MCRSessionMgr` ist durch das Design Pattern Singleton[2] realisiert.

Ist bei Shibboleth die Lazy Session aktiviert, so kann der Nutzer sich nur einmal über den IdP anmelden. Daher sollte in der Session vermerkt werden, ob der Nutzer sich bereits angemeldet hat. Ein weiteres Attribut vom Typ `boolean` und eine Get- sowie eine Set-Funktion sollten diese Anforderung erfüllen. Die Änderungen sehen dann folgendermaßen aus:

```
1  ...
2    //* Shibboleth Lazy Session Flag */
3    private boolean shibLoggedIn = false;
4  ...
5
6  ...
7    public void setShibLoggedIn() {
8      shibLoggedIn = true;
```

[2]Singleton bedeutet, dass von der Klasse nur ein Objekt gleichzeitig existieren kann.

```
 9     }
10
11     public boolean hasShibLoggedIn() {
12       return shibLoggedIn;
13     }
14   ...
```

Listing 5.4: Änderungen an MCRSession

Dabei ist zu beachten, dass die Variable sich nur einmalig auf `true` setzen lässt. Dass der Nutzer sich während einer Sitzung wiederholt einloggt, ist nicht möglich.

5.2.2 Die Servlets in Mycore

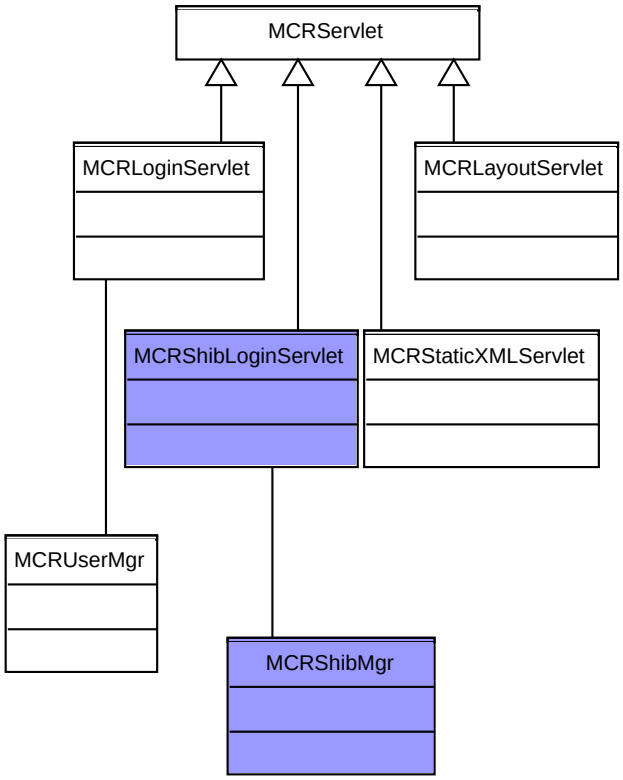

Abbildung 5.2: Servlets in Mycore

Von der Klasse `MCRServlet` gibt es keine konkreten Objekte, sondern von ihr leiten sich alle Servlet Klassen in Mycore ab. Ein Klassendiagramm in Abbildung 5.2 zeigt verschiedene Servlets von Mycore.

Beim Zugriff auf ein Mycore System wird als erstes das Servlet `MCRStaticXMLServlet` aufgerufen. Es initialisiert einige Daten und sorgt dafür, dass der Nutzer zunächst den

Status Gast erhält. Dabei wird auch vom Session Manager `MCRSessionMgr` eine neue Session erzeugt, indem die Daten des Nutzers gespeichert werden. Danach wird an das Servlet `MCRLayoutServlet` übergeben. Dieses Servlet ist, wie der Name schon andeutet, für das Layout zuständig.

Möchte der Nutzer sich lokal einloggen, so ist dafür das Servlet `MCRLoginServlet` zuständig. Es erledigt mehrere Aufgaben. Beim ersten Aufruf zeigt es dem Nutzer eine Login Maske für UserID und Passwort. Der Nutzer gibt seine Daten ein und klickt auf den Button „Anmelden“. Danach wird wiederum `MCRLoginServlet` aufgerufen. Es enthält die eingegebenen Daten des Nutzers. Nun kommt noch eine weitere Klasse namens `MCRUserMgr` ins Spiel. Sie ist wie die Klasse `MCRSessionMgr` mit Hilfe des Design Patterns Singleton entworfen worden. `MCRUserMgr` ist eine Klasse, die ein Großteil der Funktionalität des Bereichs Nutzerverwaltung kapselt. `MCRLoginServlet` versucht den Nutzer mit der Instanz von `MCRUserMgr` zu authentifizieren. Dazu vergleicht eine Funktion in `MCRUserMgr` die eingegebenen Daten des Nutzers mit den Einträgen in der Datenbank. Gibt es eine Übereinstimmung, gilt der Nutzer als eingeloggt. Andernfalls wird vom Servlet eine Fehlermeldung angezeigt.

Für den Einsatz von Shibboleth bietet es sich an, zwei neue Klassen zu erstellen. In Anlehnung an die interne Nutzerverwaltung von Mycore sind dies `MCRShibLoginServlet` und `MCRShibUserMgr`. Das Servlet `MCRShibLoginServlet` verhält sich jedoch anders als `MCRLoginServlet`. Es fordert den Nutzer nicht aktiv auf Daten einzugeben, sondern es überprüft, ob der Nutzer sich bereits über den IdP angemeldet hat.

5.2.3 StaticXMLServlet

Zunächst wird nur der Fall betrachtet, dass Mycore durch Shibboleth ohne Lazy Session geschützt wird. Dies ist der einfachere Fall. Der Nutzer muss sich über den IdP anmeldem, bevor er Zugriff auf Mycore erhält. In Mycore wird das Servlet `StaticXMLServlet` aufgerufen. Hier wird überprüft, ob Shibboleth aktiviert, der Nutzer nicht über Shibboleth angemeldet und die Lazy Session deaktiviert ist. Sollten alle drei Bedingungen erfüllt sein, wird an das Servlet `MCRShibLoginServlet` übergeben. Listing 5.5 enthält in den Zeilen 5-14 den hinzugefügten Code.

```
1  ...
2  job.getRequest().setAttribute("XSL.FilePath", file.getPath());
3  job.getRequest().setAttribute("MCRLayoutServlet.Input.FILE", file);
4
5  MCRSession session = MCRSessionMgr.getCurrentSession();
6
7  if ((session.hasShibLoggedIn() == false)
8      && (CONFIG.getString("MCR.shib_active").equals("true"))
9      && (CONFIG.getString("MCR.shib_lazy_session").equals("false")) {
10   RequestDispatcher rd = getServletContext()
11       .getNamedDispatcher("MCRShibLoginServlet");
12   rd.forward(job.getRequest(), job.getResponse());
13   return;
```

```
14 }
15
16 RequestDispatcher rd = getServletContext()
17     .getNamedDispatcher("MCRLayoutServlet");
18 rd.forward(job.getRequest(), job.getResponse());
19 ...
```

Listing 5.5: Änderungen an MCRStaticXMLServlet

Damit wird nicht mehr automatisch an `MCRLayoutServlet` übergeben. Falls die Bedingungen erfüllt sind, wird an das Servlet `MCRShibLoginServlet` übergeben. Die Klasse `RequestDispatcher` in Zeile 10 ist dafür zuständig, an ein gewünschtes Servlet zu übergeben.

5.2.4 MCRShibLoginServlet

Die Klasse `MCRShibLoginServlet` wurde neu erzeugt. Das Servlet wird für den Einsatz von Shibboleth benötigt. Es wir für beide Arten von Shibboleth eingesetzt, entweder mit oder ohne Lazy Session. Bei Shibboleth ohne Lazy Session wird das Servlet automatisch vom dem zuvor vorgestellten `MCRStaticXMLServlet` aufgerufen. Wird Shibboleth mit Lazy Session betrieben, so erfolgt der Aufruf unmittelbar nach der Anmeldung des Nutzers beim IdP.

```
1  ...
2  public void doGetPost(MCRServletJob job) throws Exception {
3
4    MCRSession session = MCRSessionMgr.getCurrentSession();
5
6    if (CONFIG.getString("MCR.shib_active").equals("true")) {
7      session.userShibLoggedIn();
8      String shibUser = job.getRequest().getHeader(
9          CONFIG.getString("MCR.shib_header_username"));
10     if (shibUser != null) {
11       if (MCRShibMgr.instance().login(shibUser)) {
12         StringBuffer groups = new StringBuffer();
13         session = MCRSessionMgr.getCurrentSession();
14         String uid = session.getCurrentUserID();
15         ArrayList groupList = MCRUserMgr.instance()
16             .retrieveUser(uid).getGroupIDs();
17
18         for (int i = 0; i < groupList.size(); i++) {
19           groups.append((String) groupList.get(i)).append(" ");
20         }
21         RequestDispatcher rd = getServletContext()
22             .getNamedDispatcher("MCRUserServlet");
23         job.getRequest().getSession()
24             .setAttribute("XSL.CurrentGroups", groups.toString());
25         job.getRequest().removeAttribute("lang");
26         job.getRequest().setAttribute("lang", session.getCurrentLanguage());
27         job.getRequest().removeAttribute("mode");
```

```
28          job.getRequest().setAttribute("mode", "Select");
29          rd.forward(job.getRequest(), job.getResponse());
30
31          return;
32        }
33      }
34    }
35    String backto_url = MCRServlet.getBaseURL();
36    job.getResponse().sendRedirect(backto_url);
37
38    return;
39  }
40  ...
```

Listing 5.6: Auszug aus der Klasse MCRShibLoginServlet

Die wichtigste Funktion des Servlets zeigt Listing 5.6. Als erstes wird überprüft, ob Shibboleth aktiviert ist. Dann wird in der Session gespeichert, dass der Nutzer sich eingeloggt hat. Dies ist nötig, da es bei Shibboleth keine Logout-Funktion gibt und daher ein wiederholtes Anmelden nicht möglich ist.

In Zeile 8 wird die UserID des Nutzers ausgelesen. Diese wird von Shibboleth als Umgebungsvariable über den HTTP-Header übergeben. Sollte diese einen Wert enthalten, wird der Nutzer über die Klasse `MCRShibMgr` im lokalen System angemeldet. Diese Klasse wird im folgenden Abschnitt vorgestellt. Ist das Einloggen erfolgreich, wird der Nutzer vom System als angemeldet betrachtet. Ab Zeile 12 werden einige Einstellungen bezüglich des Nutzers in Mycore vorgenommen.

5.2.5 MCRShibMgr

```
1  ...
2  public final synchronized boolean login(String userID) {
3    if (userID == null) {
4      return false;
5    } else {
6      if (CONFIG.getString("MCR.shib_match_lokal_user").equals("true")) {
7        if (MCRUserMgr.instance().existUser(userID)) {
8          MCRSession session = MCRSessionMgr.getCurrentSession();
9          session.setCurrentUserID(userID);
10         return true;
11       }
12     }
13     MCRSession session = MCRSessionMgr.getCurrentSession();
14     session.setCurrentUserID(CONFIG.getString("MCR.shib_username"));
15     return true;
16   }
17 }
18 ...
```

Listing 5.7: Auszug aus der Klasse MCRShibMgr

Listing 5.7 zeigt die Login-Funktion. Falls die UserID keinen Wert enthält wird `false` zurück gegeben. Sonst wird überprüft (Zeile 6), ob die Suche nach der passenden UserID in der lokalen Datenbank aktiviert ist. Ist dies der Fall, und die UserID existiert, wird der Nutzer mit dieser UserID von Mycore als eingeloggt geführt. Andernfalls wird der Nutzer im System als Shibboleth Nutzer geführt. Soll der Nutzer nicht lokal gesucht werden, wird er automatisch zum Shibboleth Nutzer, ungeachtet dessen, ob seine UserID in der lokalen Datenbank vorhanden ist oder nicht.

Diese Klasse enthält noch nicht viel mehr an Funktionalität, daher könnte man diese auch in `MCRSHibLoginServlet` realisieren. Aber in Hinblick auf den eventuellen Einsatz von zusätzlicher Funktionalität von Shibboleth, wie beispielsweise Attribute, wurde diese Lösung gewählt.

5.2.6 Ergebnis des Entwurfs

Der Entwurf der Klassen muss sicherlich als prototypisch angesehen werden. Er soll den praktischen Einsatz von Shibboleth in Mycore aufzeigen. Wie genau und zu welchem Zeitpunkt Shibboleth letztendlich in das Mycore Projekt integriert wird, kann noch nicht eindeutig beantwortet werden. Die Gründe dafür sind vielschichtig. Die aufgezeigten Probleme im vorigen Kapitel verdeutlichen die Schwierigkeiten. Die Hoffnung zu Beginn der Arbeit, Shibboleth einfach und effektiv in Mycore integrieren zu können, hat sich leider nicht bewahrheitet. Dies liegt sicherlich auch an der Architektur von Shibboleth. Infolge dessen wurde zusätzlich der Einsatz eines weiteren Systems zur Authentifizierung für Mycore untersucht, was im nächsten Kapitel gezeigt werden wird.

5.3 Der Shibboleth Login

Wie sieht nun der genaue Ablauf eines Logins in Mycore System mit Shibboleth aus? Es gibt für den Nutzer zwei Buttons um sich anzumelden. Einmal durch den lokalen Login, der keine Änderung erfahren hat, und somit wie in einem Mycore ohne Shibboleth funktioniert. Zum anderen kann sich der Nutzer über den Button „Login via Shibboleth“ extern über Shibboleth anmelden. Dieser Link kann nicht dynamisch in der Menüleiste angezeigt werden, da dies Mycore nicht anbietet. Der Nutzer kann sich über Shibboleth nur einmalig pro Sitzung anmelden. Sollte Shibboleth ohne Lazy Session betrieben werden, ist der Login über Shibboleth nicht vom Nutzer aktiv wählbar - er wird automatisch beim ersten Zugriff zur Authentifizierung am IdP aufgefordert. Das Problem ist daher, wie der Nutzer davon abgehalten werden kann, sich über den Link einzuloggen, obwohl dies nicht mehr möglich ist.

Eine Lösung für das Problem ist eine extra Seite für den externen Login. Klickt der Nutzer also auf den Button „Login via Shibboleth“, wird er nicht sofort zum IdP weitergeleitet. Es wird zuerst überprüft, ob er sich überhaupt einloggen kann. Dies ist nur der Fall, falls Shibboleth mit Lazy Session betrieben wird, und er sich bisher in der Sitzung

noch nicht über Shibboleth authentifiziert hat. Sollte dies zutreffen, wird der Link zum Anmelden mit Shibboleth angezeigt. Andernfalls erhält er eine Fehlermeldung, die ihm vermittelt, dass diese Aktion nicht mehr möglich ist.

Dieses Problem wird durch Shibboleth 2.0 wahrscheinlich gelöst werden, da sich der Nutzer dann abmelden kann. Daher ist ein wiederholtes Anmelden möglich und der direkte Link in der Menüleiste ohne weitere Abfrage einsetzbar. Die Anzeige der zusätzlichen Seite entfällt somit.

5.4 Pressearchiv

Neben der allgemeinen Implementierung der entsprechenden Klassen für den Einsatz von Shibboleth in Mycore wird nun noch ein konkreter Fall betrachtet. Es geht um den Einsatz von Shibboleth in einem praktischen Fall des Rechenzentrum Freiburg. Dies soll zeigen, dass neben der bisher gezeigten allgemeinen Herangehensweise, bei der immer alle Eventualitäten in Betracht gezogen werden, der Entwurf von einzelnen konkreten Lösungen ebenso sinnvoll sein kann.

Das Pressearchiv ist eine Anwendung, die auf Mycore basiert. Sie bietet bestimmten Mitarbeitern eine Plattform, um Presseberichte im Rahmen der Universität zu archivieren. Sie ist bereits beim Rechenzentrum der Universität Freiburg im produktiven Einsatz. Jedoch gibt es einige Forderungen an das System, die bisher nicht realisiert werden konnten. Nun stellt sich die Frage, ob dies mit Shibboleth zu erreichen ist. Dabei gibt es die Forderung, dass an dem laufenden System so wenig wie möglich verändert wird.

5.4.1 Anforderungen

Die Nutzer, die das Pressearchiv nutzen, bilden eine kleine Gruppe von ca. 20 Personen. Diese Personen haben bereits einen Account im internen System von Mycore. Sie können sich somit bereits einloggen und neue Artikel in das Archiv einstellen. Dies funktioniert so, wie es für ein normales Mycore vorgesehen ist. Daher ist auch jeder Nutzer, der auf die Seite zugreift, erst einmal Nutzer mit dem Status „Gast“. Genau dies soll mit Shibboleth verhindert werden. Außer den Nutzern, die einen Account im Pressearchiv haben, soll niemand einen Zugriff auf das System haben.

Shibboleth bzw. der SP greift auf den IdP der Bibliothek der Universität Freiburg zu. Dieser kennt alle Nutzer, die einen Account beim Rechenzentrum haben. Somit erfasst er auch alle Nutzer, die Zugriff auf das Pressearchiv haben sollen.

5.4.2 Einsatz von Attributen

Wie kann man nun am besten die Anforderungen des Pressearchivs erfüllen? Die einfachste Möglichkeit ist der Einsatz von Attributen. Jeder Nutzer des Presserachivs erhält eine spezielles Attribut beim IdP. Dabei ist der Name des Attributs beliebig, da es nur für diese spezielle Anwendung benötigt wird. Das Mycore System kann nicht dafür sorgen, dass keine unberechtigten Nutzer Zugriff haben. Das System zeigt ohne Überprüfung sofort die Startseite des Pressearchivs an. Die Selektion der Nutzer muss also früher geschehen. Hier kommt der SP ins Spiel. Er leitet erst jeden Nutzer, der auf das Pressearchiv zugreift, zum IdP weiter. Hat der Nutzer sich erfolgreich angemeldet, erhält der SP die entsprechende Session mit den verfügbaren Attributen des Nutzers. Nun prüft der SP zusätzlich, ob das entsprechende Attribut beim Nutzer vorhanden ist. Nur dann erhält der Nutzer den Zugriff auf das Pressearchiv.

Der Nachteil dieser Lösung ist, dass sich ein Nutzer zweimal anmelden muss. Dies ist aber mit der Forderung, möglichst wenig am laufenden System zu ändern, nicht anders möglich. Wird ein neuer Nutzer angelegt, so erhält er beim IdP einen Account mit dem Attribut des Pressearchivs und einen entsprechenden Account in Mycore. Beim Löschen eines Nutzers ist es wichtig darauf zu achten, dass das entsprechende Attribut beim IdP gelöscht wird. Wird der Account beim IdP gelöscht und im Mycore System nicht, hat dies keine Konsequenzen, da der Zugriff nicht mehr möglich ist. Der SP verweigert ohne das Pressearchiv-Attribut den Zugriff.

5.4.3 Realisierung

Als erstes wird beim IdP ein neues Attribut für die Nutzer gewählt - beispielsweise das Attribut `urn:mace:unifr:pressearchiv:pressearchiv`. Dies wird bei jedem Nutzer eingetragen, der zum Zugriff auf das Pressearchiv berechtigt ist. Hat der entsprechende Nutzer keinen Account beim IdP, so muss dieser noch angelegt werden. Jeder Nutzer sollte bereits einen Account im Mycore System haben.

Der SP erhält nun eine spezielle Konfiguration, die von den bisher gezeigten abweicht.

```
1 ...
2 <Location /pressearchiv>
3     AuthType Shibboleth
4     ShibRequireSession On
5     Require urn:mace:unifr:pressearchiv:pressearchiv
6 </Location>
7 ...
```

Listing 5.8: Einstellung für den SP des Pressearchivs - mod_shib.conf

Die Einstellung `Require` enthält das spezielle Attribut für das Pressearchiv. Diese bedeutet, dass ein Zugriff auf das Verzeichnis `pressearchiv` nur möglich ist, falls das Attribut vorhanden ist.

Die Anwendung selbst hat nun nichts mehr mit der Autorisierung zu tun. Da die Zugriffskontrolle schon zuvor im Apache Webserver erfolgt, können nur autorisierte Nutzer Zugriff erhalten. Diese sind im Pressearchiv zunächst Gast Nutzer und können sich dann mit ihrem lokalen Account anmelden.

5.4.4 Ergebnis des Pressearchivs

Der Einsatz von Shibboleth im Rahmen des Pressearchivs zeigt, dass eine Anpassung von Code im Mycore System nicht immer unbedingt nötig ist. Es erfordert zwar eine doppelte Anmeldung des Nutzers, aber das Pressearchiv erhält durch diese Lösung rasch den benötigten Schutz.

Grundsätzlich zeigt dieses Beispiel auch, dass eine Anpassung auf die Ansprüche eines speziellen Mycore Systems durchaus Sinn macht. Wenn der Einsatz von Shibboleth allgemein noch einige Fragen aufwirft, gibt es sicherlich in einigen Fällen Lösungen für ein konkretes Mycore System, die durchaus als praktikabel eingestuft werden können.

Kapitel 6

Mycore und LDAP

Die Ergebnisse der Untersuchungen der Integration von Shibboleth in Mycore lassen einige Wünsche offen. Da es nicht möglich ist, die komplette Nutzerverwaltung von Shibboleth übernehmen zu lassen, bleibt die Frage nach einer Alternative zu Shibboleth. Dabei sollte Mycore wünschenswerterweise die Kontrolle über den Prozess der Authentifizierung behalten. Damit kann in einer einfachen Version nur die Authentifizierung ausgelagert werden, wobei an der restlichen Nutzerverwaltung nichts geändert werden muss. Eine solche Lösung könnte sich durch den Einsatz von LDAP ergeben.

Allerdings muss dabei beachtet werden, dass man LDAP und Shibboleth nicht direkt vergleichen kann. Es handelt sich um zwei grundlegend verschiedene Ansätze. LDAP kann daher ein Shibboleth System niemals ersetzen. Aber es kann vielleicht einige Probleme, die sich beim Einsatz von Shibboleth aufgetan haben, lösen.

Als erstes wird LDAP kurz vorgestellt und auf den Aufbau eines Testsystems eingegangen werden. Anschließend wird untersucht werden, wie sich eine Authentifizierung in Mycore realisieren lässt. Dann wird ein Vergleich von LDAP und Shibboleth, soweit dies möglich ist, aufgezeigt werden. Abschließend wird der Einsatz von LDAP in einer Zusammenfassung diskutiert werden.

6.1 LDAP

LDAP ist die Abkürzung für Lightweight Directory Access Protocol. Es handelt sich um ein Netzwerkprotokoll, das die Abfrage und Modifikation von Informationen eines Verzeichnisdienstes erlaubt. Das Protokoll dient zur Kommunikation zwischen einem LDAP-Client und einem LDAP-Server. Ein Client kann z.B. ein Email-Programm oder eine Ressource mit ausgelagerter Authentifizierung sein. Ganz allgemein ist ein LDAP Server eine Datenbank zur Speicherung beliebiger Daten. Der Server dient typischerweise zur Speicherung der Daten von allen Mitgliedern einer Organisation oder einem

Unternehmen. Er kann aber auch Informationen enthalten, die nicht personenbezogen sind, wie beispielsweise Informationen über alle Rechner einer bestimmten Einrichtung.

Eine typische Anwendung eines LDAP Servers findet sich in einem Unternehmen. Dort sind beispielsweise unter anderem die Namen der Mitarbeiter und ihre Email-Adressen gespeichert. Möchte nun ein Mitarbeiter einem anderen eine Nachricht per Email schicken, so muss er nur dessen Namen wissen. Den gibt er in seinem Email Client ein, und dieser erhält die Email Adresse vom LDAP Server. Von jedem Nutzer befinden sich meist auch die UserID und ein persönliches Passwort auf dem Server. Somit kann sich ein Nutzer über den LDAP Server authentifizieren und erhält Zugriff auf bestimmte angeschlossene Ressourcen.

LDAP wurde an der Universität Michigan entwickelt und basiert auf dem X.500-Standard. X-500 ist ein sehr umfangreicher Standard von einem Verzeichnisdienst mit vielen Anforderungen und ist daher schwer zu implementieren. Folglich hat er sich in der Praxis nicht durchsetzen können. LDAP ist in dieser Hinsicht flexibler und setzte sich auch durch die Beschränkung auf das TCP/IP-Protokoll als Standard durch.

6.1.1 LDAP Verzeichnis

Die Daten eines LDAP Servers werden in dem LDAP Verzeichnis gespeichert. Dieses hat eine hierarchische Baumstruktur. Die Wurzel *root* ist das oberste Datenobjekt. Von ihr aus verzweigen sich alle abgelegten Informationen.

Hier erkennt man den Unterschied zu einer relationalen Datenbank, die Daten in beliebigen Tabellen speichern kann, die untereinander in keiner Beziehung stehen müssen. Ein LDAP Verzeichnis wird gerne für Daten eingesetzt, die keinen großen Veränderungen unterliegen. Dies ist bei den Daten von Mitgliedern einer Organisation beispielsweise der Fall.

Jedes Objekt, das sich im LDAP Verzeichnis befindet, hat einen eindeutigen Namen - Distinguished Name (DN) genannt. So kann es in der Struktur des Verzeichnisses gefunden werden. Dieser Name setzte sich aus den *Relative Distinguished Names* (RDN) zusammen. Sie sind die Bezeichnungen der Äste von der Wurzel bis zu dem Objekt. Gibt es einen Nutzer mit der UserID *jjohn*, und befindet er sich in der Gruppe *people* in dem LDAP Verzeichnis von *example.de*, so sieht sein DN folgendermaßen aus: *uid=jjohn, ou=people, dc=example, dc=de* .

Damit nicht alle Informationen willkürlich in dem LDAP Verzeichnis gespeichert sind, gibt es für die häufigsten Anforderungen eine bestimmte vorgegebene Struktur. Diese ist in einem so genannten Schema definiert. In einem solchen Schema sind Klassen von Objekten beschrieben, die eine Anzahl von Attributen besitzen. Dabei kann ein Objekt zu mehreren Klassen gehören. Das Objekt enthält dann alle Attribute dieser Klassen, wobei nicht alle Attribute zwingend einen Wert enthalten müssen.

LDAP bietet noch eine Fülle von Möglichkeiten, um bestimmte Informationen zu speichern und sie somit flexibel zugänglich zu machen. Es enthält auch eine komplexe Zugriffsverwaltung. So kann über die Konfiguration erreicht werden, dass jeder eingetragene Nutzer nur auf seine Daten zugreifen kann - und dies nur mit seinem persönlichen Passwort.

6.2 Aufbau eines LDAP Servers mit Testnutzern

Für den Einsatz von LDAP im Rahmen von Mycore wird primär eine Anzahl von Testnutzern benötigt. Diese haben einige persönliche Daten und befinden sich ein einer Gruppe. Dabei ist die UserID und das persönliche Kennwort am wichtigsten. Als LDAP Server kommt OpenLDAP[1] zu Einsatz. OpenLDAP ist Open Source und in jeder gängigen Linux Distribution schon als fertiges Paket vorhanden.

Bei der Testanwendung befinden sich OpenLDAP und Mycore auf dem selben Rechner. Das erleichtert die Konfiguration und den Zugriff auf den LDAP Server. Für einen produktiven Einsatz sind diese Vorraussetzungen sicherlich nicht erfüllt. Dies erfordert dann noch einige Änderungen an der Konfiguration.

6.2.1 Konfiguration

```
1  include         /etc/ldap/schema/cosine.schema
2  include         /etc/ldap/schema/inetorgperson.schema
3  include         /etc/ldap/schema/nis.schema
4  pidfile         /var/run/slapd/slapd.pid
5  argsfile        /var/run/slapd.args
6  modulepath      /usr/lib/ldap
7  moduleload      back_bdb
8  backend         bdb
9  checkpoint 512 30
10
11 access to *
12   by * read
13
14 database        bdb
15 suffix          "dc=mycore_test,dc=de"
16 rootdn          "cn=Manager,dc=mycore_test,dc=de"
17 rootpw          secret
18 directory       /var/lib/ldap
19 index objectClass                          eq,pres
20 index ou,cn,mail,surname,givenname        eq,pres,sub
21 index uidNumber,gidNumber,loginShell      eq,pres
22 index uid,memberUid                       eq,pres,sub
23 index nisMapName,nisMapEntry              eq,pres,sub
```

Listing 6.1: Einstellung für den LDAP Test-Sever - slapd.conf

[1] www.openldap.org

OpenLDAP wird über die Datei `slapd.conf` konfiguriert. Listing 6.1 zeigt die Konfiguration des LDAP Servers für die Testumgebung. Zeile 1-3 bestimmt die Schemata, die geladen werden sollen. In den Zeilen 11 und 12 wird definiert, dass der Zugriff auf die Daten für jeden möglich ist. Dies sollte in einem produktiven Einsatz entsprechend angepasst werden.

Die Daten der Nutzer werden nicht direkt an die Wurzel angehängt, sondern unter den beiden RDNs *dc=mycore_test,dc=de* abgelegt. Dies wird durch die Einstellung `suffix` in Zeile 15 festgelegt. Die Zeilen 16 und 17 legen die Daten des Administrators bzw. Root Nutzers fest. Er wird Manager genannt und ist unter *cn=Manager, dc=mycore_test, dc=de* abgelegt. Vom Speichern des Passworts im Klartext in der Konfigurationsdatei ist im produktiven Einsatz ebenso abzusehen. Die restlichen Zeilen beschäftigen sich mit der Indizierung der Attribute für die Suche, was für die Authentifizierung nicht relevant ist.

```
1 dn: dc=mycore_test,dc=de
2 objectclass: dcObject
3 objectclass: organization
4 dc: mycore_test
5 o: LDAP Server zum Test von Mycore
6
7 dn:cn=Manager,dc=mycore_test,dc=de
8 objectclass: organizationalRole
9 cn: Manager
```

Listing 6.2: Daten zu Manager und Name des Verzeichnis für den LDAP Test-Server - admin.ldif

Wird der OpenLDAP Server gestartet, so enthält er noch keine Daten oder Objekte. Dazu müssen zunächst einige Daten für die Grundstruktur geladen werden. Dies wird in der Testanwendung über die Kommandozeile vorgenommen. Listing 6.2 zeigt die Datei `admin.ldif`, in der zwei Objekte für einen LDAP Server definiert sind. Diese können mit dem Befehl `ldapadd` zu den Daten des Servers hinzugefügt werden. Der komplette Befehl lautet:

```
ldapadd -h localhost -D cn=Manager,dc=mycore_test,dc=de -w secret -f admin.ldif
```

Danach existiert im LDAP Server eine Wurzel, von der aus ein Ast mit dem Namen *dc=de* abgeht. Von dort gibt es einen weiteren Ast mit dem Namen *dc=mycore_test*, worin sich das Objekt des Managers (*cn=Manager*) befindet. Man kann sich so einen Ast auch als Ordner in einem Verzeichnis vorstellen. Ein Ordner kann wiederum weitere Ordner enthalten - diese stellen bei der Baumstruktur eine weitere Verzweigung dar.

6.2.2 Anlegen der Nutzer

In diesem Abschnitt werden die Daten der Nutzer hinzugefügt. Die Daten der Nutzer befinden sich in einer weitere Datei namens `users.ldif`. Diese wird wie zuvor über den Befehl `ldapadd` zum Server übertragen.

```
1  dn: ou=people,dc=mycore_test,dc=de
2  objectclass: organizationalUnit
3  ou: people
4
5  dn: uid=shendry,ou=people,dc=mycore_test,dc=de
6  cn: Stephen Hendry
7  sn: Hendry
8  givenname: Stephen
9  objectclass: top
10 objectclass: person
11 objectclass: organizationalPerson
12 objectclass: inetOrgPerson
13 ou: Accounting
14 ou: People
15 l: Scotland
16 uid: shendry
17 mail: shendry@mycore_testd.e
18 telephonenumber: +44 408 555 4798
19 facsimiletelephonenumber: +44 408 555 9751
20 roomnumber: 0001
21 userpassword: scotland
22
23 dn: uid=kdoherty,ou=people,dc=mycore_test,dc=de
24 cn: Ken Doherty
25 sn: Doherty
26 givenname: Ken
27 objectclass: top
28 objectclass: person
29 objectclass: organizationalPerson
30 objectclass: inetOrgPerson
31 ou: Accounting
32 ou: People
33 l: Irland
34 uid: kdoherty
35 mail: kdoherty@mycore_test.de
36 telephonenumber: +44 408 555 9187
37 facsimiletelephonenumber: +44 408 555 8473
38 roomnumber: 0002
39 userpassword: irland
40
41 dn: uid=smurphy,ou=people,dc=mycore_test,dc=de
42 cn: Shaun Murphy
43 sn: Murphy
44 givenname: Shaun
45 objectclass: top
46 objectclass: person
47 objectclass: organizationalPerson
48 objectclass: inetOrgPerson
49 ou: Accounting
50 ou: People
51 l: Enland
52 uid: smurphy
53 mail: smurohy@mycore_test.de
```

```
54 telephonenumber: +44 30124314-21412
55 facsimiletelephonenumber: +44 23123 2312
56 roomnumber: 0005
57 userpassword: england
```

Listing 6.3: Nutzer und ihre Daten für den LDAP Test-Server - users.ldif

Listing 6.3 zeigt den Inhalt der Datei. In den ersten drei Zeilen wird eine Gruppe mit dem Namen *ou=people* erzeugt. In ihr befinden sich alle Nutzer, die in den weiteren Zeilen definiert werden. Wichtig bei jedem Nutzer ist die UserID und das Passwort. Alle weiteren Daten sind für den Test nicht zwingend notwendig, machen das Beispiel aber anschaulicher.

Nun kann man verschiedene Anfragen an den Server stellen. Dies kann z. B. die Suche nach einem Nutzer oder die Ausgabe aller Nutzer sein. Für den Einsatz im Rahmen von Mycore ist zunächst allerdings nur die Authentifizierung eines Nutzers wichtig.

6.2.3 Zugriff auf LDAP aus einer Java-Umgebung mittels JNDI

Wie kann auf einen LDAP Server aus einer Java-Umgebung heraus zugegriffen werden? Dies kann am einfachsten mit dem *Java Naming und Directory Interface* (JNDI) erreicht werden. JNDI ist Bestandteil der Java Standard Edition und somit problemlos in Mycore einsetzbar.

Mittels JNDI kann man auf verschiedene Arten von Namens- und Verzeichnisdiensten zugreifen. Dabei bietet JNDI eine allgemeine Schnittstelle. Neben einem LDAP Server kann ebenso auf Dateisysteme oder DNS-Server zugegriffen werden, um nur zwei Beispiele zu nennen. JNDI ist modular aufgebaut und kann durch das allgemeine Schnittstellenkonzept einfach auf neue Arten von Verzeichnisdiensten erweitert werden.

Die Klassen von JNDI befinden sich hauptsächlich im Paket `javax.naming`. Für den Zugriff auf Verzeichnisse eines Dateisystems stehen die Klassen unter `javax.naming.directory` zu Verfügung - für LDAP sind sie unter `javax.naming.ldap` zu finden. Eine zentrale Bedeutung hat die Klasse `javax.naming.Context`. Sämtliche JNDI-Operationen werden auf Objekten ausgeführt, deren Klassen diese Schnittstelle implementieren.

Die Beispiele und Überlegungen zum Einsatz von LDAP im Rahmen von Java stammen hauptsächlich aus dem Buch „LDAP für Java-Entwickler" von Stefan Zörner und Jörg Wegener [12]. Dies ist als Einstieg in die Materie sehr zu empfehlen. Es bietet gleichzeitig einen Überblick über das theoretische Konzept hinter LDAP und den praktischen Einsatz in Java.

6.2.4 Authentifizierung mit JNDI

Wie kann man solch eine Authentifizierung erreichen? Dazu wird zunächst ein einfaches Java Programm betrachtet, das über die Kommandozeile gesteuert wird. Es ermöglicht den Zugriff auf den OpenLDAP Server mittels des zuvor besprochenen JNDIs.

```
1  import java.util.Hashtable;
2  import javax.naming.Context;
3  import javax.naming.InitialContext;
4  import javax.naming.NamingEnumeration;
5  import javax.naming.NamingException;
6
7  public class ldapTest {
8
9    public static void main(String[] args)
10     throws NamingException {
11
12     if (args.length < 2) {
13         \\ Keine zwei Argumente
14     } else {
15
16       Hashtable env= new Hashtable();
17       env.put(Context.INITIAL_CONTEXT_FACTORY,
18               "com.sun.jndi.ldap.LdapCtxFactory");
19       env.put(Context.PROVIDER_URL,"ldap://localhost:389");
20       env.put(Context.SECURITY_AUTHENTICATION, "simple");
21       env.put(Context.SECURITY_PRINCIPAL, args[0]);
22       env.put(Context.SECURITY_CREDENTIALS, args[1]);
23
24       try {
25         Context ctx=new InitialContext(env);
26
27         ctx.close();
28         System.out.println("Authentifizierung erfolgreich!");
29       } catch (javax.naming.NamingException e) {
30         System.out.println(e.toString());
31
32       }
33     }
34   }
35 }
```

Listing 6.4: Testklasse zur Authentifizierung eines Nutzers über die Kommandozeile - ldapTest.java

In dem Listing 6.4 wird die Klasse `ldapTest` vorgestellt. Dabei werden die UserID und das Passwort über Parameter beim Aufruf des Programms übergeben. Dabei reicht die UserID alleine nicht aus. Der LDAP Server benötigt den kompletten DN - beim Nutzer Stephen Hendry wäre dies *uid=shendry, ou=people, dc=mycore_test, dc=de* . Ein Aufruf einer erfolgreichen Authentifizierung sieht folgendermaßen aus:

```
javac ldapTest uid=shendry,ou=people,dc=mycore_test,dc=de scotland
```

Stimmen UserID und Passwort nicht mit den gespeicherten Daten im LDAP Server überein, so führt dies zu einem Fehler vom Typ `javax.naming.NamingException`. Dieser enthält eine Fehlernummer, die der LDAP Server übergibt. Bei falscher Kombination von UserID und Passwort ist dies der Fehler mit der Nummer 49. Sollte keine Verbindung zum LDAP Server zustande kommen, wird ein andere Fehler ausgegeben. Daher ist eine Überprüfung der Fehlernummer wichtig, um dem Nutzer nicht fälschlicherweise eine falsche Kombination von UserID und Passwort zu melden.

In Zeile 19 wird die URL des LDAP Servers angegeben. Der LDAP Server befindet sich auf dem selben Rechner wie Mycore und ist somit über `localhost` zu erreichen. Der übliche Port für eine Kommunikation mittels LDAP ist wie angegeben 389. In der nächsten Zeile wird die Art der Authentifizierung festgelegt. Hier bedeutet *simple* eine einfache Übertragung der Daten. Dass Passwort und die UserID wird bei diese Methode unverschlüsselt übertragen. Dies ist für den produktiven Einsatz nicht zu empfehlen, weitere Anmerkungen dazu folgen in Abschnitt 5 dieses Kapitels.

6.3 Authentifizierung in Mycore durch LDAP

Eine Authentifizierung eines Nutzers aus Mycore an einem LDAP Server benötigt einige Vorüberlegungen. Zunächst geht es um den Unterschied zu Shibboleth bei der technischen Realisierung. Durch den Einsatz von LDAP behält Mycore die Kontrolle über den Prozess der Authentifizierung. Es geht im Folgenden ausschließlich um die Authentifizierung eines Nutzers, der einen Account in der lokalen Nutzerdatenbank von Mycore hat. Alle weiteren Möglichkeiten im Einsatz von LDAP bleiben eventuell zukünftigen Arbeiten zu diesem Thema vorbehalten.

Die Änderung an Mycore betrifft daher nur den direkten Prozess der Authentifizierung. Anstatt zu überprüfen, ob die Kombination von UserID und Passwort in der Datenbank von Mycore vorhanden ist, wird diese Anfrage gegebenenfalls an den LDAP Server weitergeleitet. Gibt dieser keine Fehlermeldung zurück, ist der Nutzer erfolgreich angemeldet. Anderenfalls wird eine Fehlermeldung an den Nutzer ausgegeben.

Abbildung 6.1 zeigt den Ablauf des Login Prozesses in einem Mycore System, der durch den Einsatz von LDAP erweitert ist. Der Nutzer wird vom System aufgefordert, seine UserID und sein persönliches Passwort einzugeben. Nicht jede Anfrage zur Authentifizierung wird automatisch an den LDAP Server weitergeleitet. Es gibt Fälle, in denen dieses nicht sinnvoll ist. Dies wäre beispielsweise der Fall, wenn ein Administrator in Mycore keinen entsprechenden Account beim LDAP Server hat.

Als erstes muss sichergestellt werden, dass der Nutzer sich wieder abmelden kann - also wieder den Status „Gast“ erhält. Da der Gast Nutzer ebenso in der lokalen Datenbank vorhanden ist, darf dieser Nutzer nicht über den LDAP Server angemeldet werden.

Der nächste Fall betrifft die Nutzer, die keinen Account beim LDAP Server besitzen,

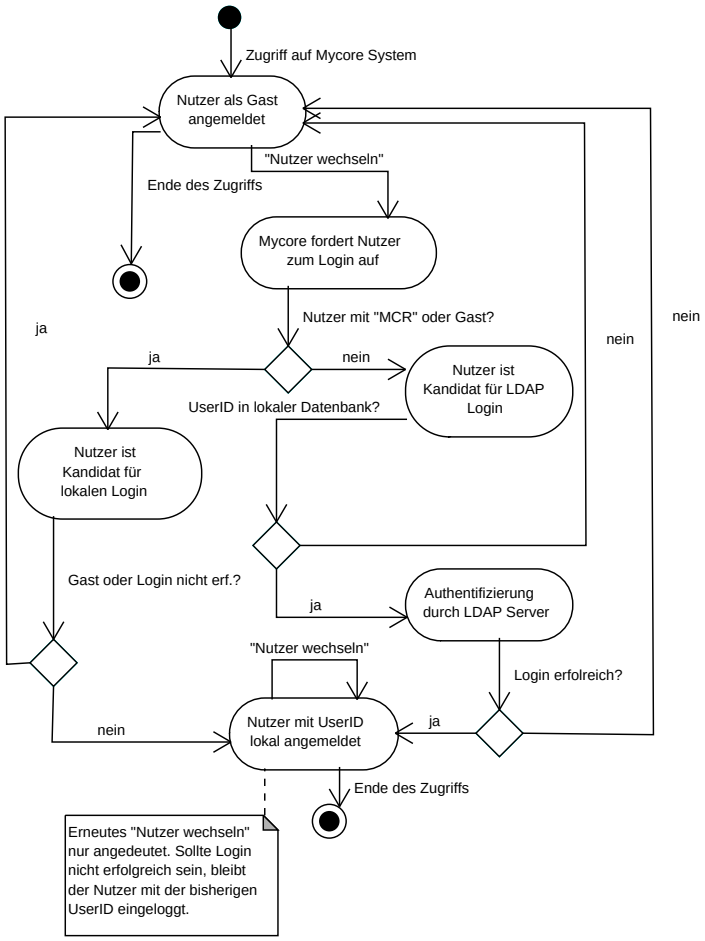

Abbildung 6.1: Ablauf eines Logins mit angeschlossenem LDAP Server. Gast Nutzer und mit Präfix „MCR“ beginnend werden lokal authentifiziert, die restlichen über einen LDAP Server.

aber sich dennoch im Mycore System anmelden dürfen. Sie benutzen dazu ihren lokalen Account und das entsprechende Passwort. In die Gruppe dieser Nutzer fallen beispielsweise Administratoren, die normalerweise keinen entsprechenden Account beim LDAP Server haben. Wie kann man nun unterscheiden, ob ein Nutzer sich lokal oder beim LDAP Server authentifizieren darf? Dies ist am einfachsten über ein Präfix in der UserID zu realisieren. In dem Testsystem ist dieser Präfix „MCR“. Jede UserID, die damit beginnt, wird also nicht am LDAP Server authentifiziert.

Sollte die UserID nicht in die beiden zuvor genannten Kategorien fallen, ist der Nutzer ein Kandidat für die Authentifizierung am LDAP Server. Nicht jeder Nutzer wird aber automatisch weitergeleitet. Zuerst wird überprüft, ob die UserID in der lokalen Datenbank bekannt ist, d.h. ob der Nutzer einen Account hat. Sollte dies nicht zutreffen, wird die Authentifizierung am LDAP Server vom System verweigert. Dies ist notwendig, um nicht die Problematik der dynamisch generierten Nutzer zu erhalten (siehe Kapitel 2.9).

Würde sich ein Nutzer am LDAP Server anmelden ohne einen lokalen Account zu besitzen, so müsste er vom System beispielsweise als LDAP Nutzer (vgl. Shibboleth Nutzer) geführt werden.

Ist die UserID lokal vorhanden, wird diese inklusive Passwort an den LDAP Server weitergeleitet. Der Nutzer ist bei erfolgreicher Authentifizierung in Mycore angemeldet, andernfalls bleibt er Gast Nutzer.

Beim Einsatz von LDAP ist eine erneute Anmeldung unbegrenzt möglich. Sollte der Versuch der Authentifizierung nicht erfolgreich sein, so bleibt der Nutzer mit der zuvor benutzten UserID im System angemeldet. Die Abbildung 6.1 zeigt nur die erste erfolgreiche Anmeldung in einem Mycore System. Jeder weitere Versuch einer Authentifizierung muss dann immer in Bezug auf die zuvor verwendete UserID gesehen werden.

Ein Szenario für die Anwendung von LDAP mit Mycore gibt es sicherlich im universitären Umfeld. Dort gibt es meist für jeden Studenten und Mitarbeiter einen Account. Dieser befindet sich oft auf einem LDAP Server. Bei der Mycore Anwendung erhalten alle Nutzer die UserID des zentralen Accounts. Informationen zu den einzelnen Nutzern müssen nicht mehr auf der Seite von Mycore gespeichert werden, da diese der LDAP Server ebenso bereitstellt. Der administrative Aufwand reduziert sich daher auf das Einstellen bzw. Löschen von UserIDs sowie das Setzen der Rollen und Rechte für die Nutzer.

Der grundlegende Unterschied zum Einsatz von Shibboleth besteht darin, dass der Nutzer den Einsatz von LDAP nicht mitbekommt. Für ihn bleibt der Login wie gehabt. Die Nutzerverwaltung von Mycore bleibt bestehen.

6.4 Implementierung in Mycore

Wie kann man den Zugriff auf den LDAP Server zur Authentifizierung in Mycore realisieren? Dies ist prototypisch relativ einfach zu erreichen. Es geht, wie schon besprochen, nur um die Authentifizierung eines Nutzers am LDAP Server.

```
1  ...
2  ##################################################################
3  # LDAP                                                           #
4  ##################################################################
5
6  # The configuration for LDAP Login
7  MCR.LDAP.active=true
8  MCR.LDAP.provider_URL=ldap://localhost:389
9  MCR.LDAP.domain=ou=people,dc=mycore_test,dc=de
10 MCR.LDAP.authentification=simple
11 ...
```

Listing 6.5: Auszug aus der Mycore Konfigurationsdatei - mycore.properties.private

Um den Einsatz von LDAP so flexibel wie möglich zu gestalten, wird wie zuvor bei Shibboleth die Konfiguration in die Datei `mycore.properties.private` ausgelagert.

Das Listing 6.5 zeigt die Änderungen in der entsprechenden Datei. Es sind vier Einstellungen zu sehen. Mit der Einstellung in Zeile 7 lässt sich der Einsatz von LDAP an- bzw. ausschalten. In den nächsten beiden Zeilen befinden sich die URL des LDAP Servers und der RDN, unter dem sich die Nutzer befinden. Die letzte Einstellung beschäftigt sich mit der Art der Kommunikation zum LDAP Server.

Die Authentifizierung in Mycore wird durch das Servlet `MCRLoginServlet` eingeleitet. Es ist für das Einloggen des Nutzers zuständig und benutzt zur Überprüfung von UserID und Passwort die Klasse `MCRUserMgr`. Diese meldet nach erfolgreicher Authentifizierung `true` zurück - andernfalls wird `false` zurück gegeben. Diese Anfrage muss nun zum LDAP Server weitergeleitet werden. Dazu wird eine neue Klasse namens `MCRLDAPMgr` erzeugt, die die Kommunikation zum LDAP Server herstellt.

```
1  ...
2  public final synchronized boolean login(String userID, String passwd) {
3
4    boolean login_success = false;
5
6    Hashtable env = new Hashtable();
7    env.put(Context.INITIAL_CONTEXT_FACTORY,
8            "com.sun.jndi.ldap.LdapCtxFactory");
9    env.put(Context.PROVIDER_URL,
10           CONFIG.getString("MCR.LDAP.provider_URL"));
11   env.put(Context.SECURITY_AUTHENTICATION,
12           CONFIG.getString("MCR.LDAP.authentification"));
13   String userLDAP = "uid="+userID+"," +
14                     CONFIG.getString("MCR.LDAP.domain");
15   env.put(Context.SECURITY_PRINCIPAL, userLDAP);
16   env.put(Context.SECURITY_CREDENTIALS, passwd);
17
18   try {
19     Context ctx = new InitialContext(env);
20     ctx.close();
21     login_success = true;
22   } catch (javax.naming.NamingException e) {
23     System.out.println(e.toString());
24   }
25   return login_success;
26 }
27 ...
```

Listing 6.6: MCRLDAPMgr.java

Im Listing 6.6 ist die Funktion `login` zu sehen. Sie leitet die beiden übergebenen Parameter UserID und Passwort an den LDAP Server weiter. Gibt dieser keine Fehlermeldung zurück, ist der Nutzer authentifiziert und die Funktion liefert `true` zurück. Andernfalls wird ein Fehler ausgegeben und die Funktion liefert `false` zurück. In den Zeilen 13+14 wird der komplette DN des Nutzers zusammengesetzt.

```
1  ...
2    if (CONFIG.getString("MCR.LDAP.active").equals("true")) {
3      if (uid != null) {
```

```
4       if ((uid.equals(CONFIG.getString("MCR.users_guestuser_username")))
5           || (uid.startsWith("MCR"))) {
6         loginOk = ((uid != null) && (pwd != null)
7                    && MCRUserMgr.instance().login(uid, pwd));
8       } else {
9         if (MCRUserMgr.instance().existUser(uid)) {
10          loginOk = ((uid != null) && (pwd != null)
11                     && MCRLDAPMgr.instance().login(uid, pwd));
12        } else {
13          root.setAttribute("unknown_user", "true");
14        }
15      }
16    }
17  } else {
18    loginOk = ((uid != null) && (pwd != null)
19               && MCRUserMgr.instance().login(uid, pwd));
20  }
21 ...
```

Listing 6.7: MCRLoginServlet.java

Das Listing 6.7 zeigt die entscheidende Stelle in der Klasse `MCRLoginServlet`. Die Variable `loginOK` vom Typ `boolean` wird bei einem erfolgreichen Login auf `true` gesetzt. Bisher waren nur die beiden Zeile 18 und 19 im Servlet vorhanden - also nur der lokale Login.

Nun ergibt sich mittels der Erweiterung durch die restlichen Zeilen, dass bei einer aktivierten LDAP Authentifizierung die Überprüfung von UserID und Passwort unter bestimmten Bedingungen an die zuvor vorgestellte Klasse `MCRLDAPMgr` weitergegeben wird. In Zeile 2 wird zuerst überprüft, ob der Einsatz eines LDAP Servers aktiviert ist. Dann wird in Zeile 4 geprüft, ob der Nutzer Gast ist oder die UserID mit „MCR“ beginnt. Der Gast Nutzer besitzt keinen Account beim LDAP Server und muss somit lokal angemeldet werden. Ebenso kann es eventuell noch Nutzer geben, die keinen Account beim LDAP Server haben, aber trotzdem einen Zugang zum Mycore System erhalten sollen. Diese Nutzer benötigen eine UserID, die mit dem Präfix „MCR“ beginnt.

Sollte der Nutzer nicht in diese Gruppe fallen, wird in Zeile 9 überprüft, ob der Nutzer in der lokalen Datenbank vorhanden ist. Nur dann wird eine Anfrage an den LDAP Server gestartet.

6.5 Hinweise zum produktiven Einsatz

Für den produktiven Einsatz gibt es einige Punkte, die beachtet werden müssen. Zunächst einmal sollte der LDAP Server besser geschützt werden, damit nicht jeder auf alle Daten zugreifen kann. Diese Probleme sind bei einem existierenden LDAP Server wohl meist gelöst. Es ist kaum davon auszugehen, dass speziell für den Einsatz im Rahmen von Mycore ein LDAP Server aufgesetzt wird. Vielmehr wird Mycore mit einem bereits

bestehenden Server eingesetzt, der i.d.R. über eine ausreichende Sicherheit der gespeicherten Daten verfügt.

Ein weiterer Punkt ist die Kommunikation zwischen Mycore und dem LDAP Server. Der LDAP Server befindet sich sicherlich nicht auf dem selben Rechner bzw. Server wie die Mycore Anwendung. Somit sollte die Kommunikation ausreichend und effizient verschlüsselt werden. Eine Möglichkeit ist, das Passwort mit Hilfe von DIGEST-MD5 zu verschlüsseln. MD5 ist eine kryptographische Hash-Funktion, um Daten in einen 128-bit Hash-Wert umzuwandeln. Von diesem Wert kann dann nicht mehr auf die ursprünglichen Daten geschlossen werden. Daher wird nicht das Passwort im Klartext, sondern durch MD5 verschlüsselt übertragen. Allerdings muss das Passwort auf dem LDAP Server dann im Klartext vorliegen, was nicht immer der Fall ist. Der LDAP Server verschlüsselt normalerweise das Passwort eines Nutzers automatisch.

Ein grundsätzlicher Punkt, der gegen den Einsatz von MD5 spricht, ist die nicht gewährte Sicherheit dieses Verfahrens. Es wurde vor einigen Jahren von Experten als nicht mehr sicher eingestuft. Die Schlüssellänge von 128 Bit ist inzwischen für eine praktische Sicherheit nicht mehr ausreichend .

Als weitere Lösung für die Sicherheit der Kommunikation mit dem LDAP Server bleibt dann die Verschlüsselung der gesamten Kommunikation. Dies kann durch SSL erreicht werden. Sowohl OpenLDAP als auch Java unterstützen diese Technik. Für den Einsatz müssen daher entsprechende Schlüssel und Zertifikate erstellt werden. Damit ist die Vertraulichkeit der Kommunikation sichergestellt.

Für die Nutzer mit dem ausschließlich lokalen Account gibt es eine weitere Variante. Anstatt eine UserID, die mit „MCR“ beginnt, zu fordern, kann eine neue Variable für jeden Nutzer eingeführt werden. Diese Variable bestimmt, ob der Nutzer sich über den LDAP Server oder lokal authentifiziert. Somit ist es nicht mehr nötig, die lokalen UserIDs mit „MCR“ beginnen zu lassen.

Die gezeigte Implementierung bietet nur eine Authentifizierung über das Servlet bzw. den Webbrowser. Wie bereits erwähnt besitzt Mycore noch einen weiteren Zugang über die Kommandozeile. Durch einige Änderungen ließe sich die Anmeldung mittels LDAP ebenso über diesen Zugang realisieren. Dazu müssten entsprechende Änderungen in den Klassen, die für die Anmeldung über die Kommandozeile zuständig sind, geändert werden.

6.6 Vergleich von LDAP und Shibboleth

Nach der Besprechung von LDAP und Shibboleth im Rahmen von Mycore bietet sich für die abschließende Betrachtung von LDAP ein Vergleich beider Systeme an. LDAP und Shibboleth besitzen jedoch einige konzeptionelle Unterschiede, die einen Vergleich nicht in jedem Punkt möglich machen. Man muss sicherlich auch zwischen verschiedenen

Szenarien für den Einsatz im Rahmen von Mycore unterscheiden.

6.6.1 Installationsaufwand

Geht man davon aus, dass jeweils ein IdP bei Shibboleth und ein LDAP Server bei LDAP vorhanden ist, bleibt der Aufwand der Installation auf der Seite von Mycore. Hier hat LDAP sicherlich einen Vorteil, da lediglich die Daten des LDAP Servers in die Konfiguration eingetragen werden müssen. Dann müssen noch Schlüssel für die Kommunikation mittels SSL erzeugt werden. Dieser Aufwand kann insgesamt sicherlich als gering eingestuft werden.

Bei Shibboleth ist der Aufwand der Installation um einiges umfangreicher. Das Erzeugen und Installieren des SP Moduls, die Einstellungen am Apache Server und die Konfiguration bei Mycore wird mehr Zeit in Anspruch nehmen. Für Shibboleth müssen ebenfalls die entsprechenden Schlüssel für die Kommunikation zwischen IdP und SP erzeugt werden.

6.6.2 Funktionalität

Bei der Frage der Funktionalität hat jedes System seine Vor- und Nachteile. Zunächst folgt eine stichwortartige Übersicht beider Systeme. Für LDAP ergeben sich folgende Punkte:

- + eine Login-Maske für den Nutzer
- + einfache Kombination von lokalen und externen Nutzern
- + schnelle Integration in Mycore Anwendung
- + für Authentifizierung über Kommandozeile einsetzbar
- – kein grundsätzlicher Schutz der Mycore Anwendung
- – kein SSO-Prinzip

Die Vor- und Nachteile bei Shibboleth:

- + Mycore kann durch Shibboleth ohne Lazy Session grundsätzlich geschützt werden
- + SSO-Prinzip wird eingehalten
- + durch Anschluss an mehrere IdPs grundsätzlich größere Anzahl von Nutzern verfügbar
- – zwei getrennte Buttons zum Einloggen für den Nutzer
- – kein Logout bzw. erneuter Login via Shibboleth in einer Sitzung möglich

6.7 Zusammenfassung

An dieser Stelle muss zunächst noch einmal betont werden, dass der Vergleich von LDAP und Shibboleth nur bedingt Sinn macht. Beide Systeme haben eine unterschiedliche Architektur und sind unter anderem auch für völlig verschieden Aufgaben geeignet.

Der Gewinn für Mycore durch den Einsatz von Shibboleth ist ein grundsätzlicher Schutz des Systems. Durch Shibboleth ohne Lazy Session erhält ein Nutzer nur Zugriff auf das System, falls er sich zuvor am IdP authentifiziert hat. Dies ist mit dem bisherigen Mycore System nicht möglich.

Ein weiterer wichtiger Punkt ist das SSO-Prinzip von Shibboleth. Dies ermöglicht dem Nutzer mit seinem Account den Zugriff auf alle Systeme, die an das Shibboleth System angeschlossen sind. Dies ist ein Ansatz, der bei der ständig wachsenden Anzahl von eingesetzten Systemen in einer Organisation, unbedingt verfolgt werden sollte.

Der wichtigste Vorteil von LDAP ist sicherlich die „interne“ Integration in Mycore. Damit ist die Tatsache gemeint, dass der Nutzer nichts von dem Einsatz von LDAP mitbekommt. Für ihn gibt es nur einen Login Button und dort gibt er seine UserID und sein Passwort ein. Ob die Authentifizierung lokal oder über den LDAP Server erfolgt, ist für ihn nicht ersichtlich, aber auch völlig unerheblich. Die Funktionalität der bisherigen Nutzerverwaltung in Mycore bleibt komplett erhalten. LDAP erweitert ohne Einschränkungen für den Nutzer die Authentifizierung für das Mycore System. Zusätzlich lässt es sich auch für den Zugang über die Kommandozeile verwenden.

Kapitel 7

Zusammenfassung und Ausblick

Die Ergebnisse der Arbeit zeigen, dass der Einsatz von Shibboleth in Mycore nicht ohne weiteres möglich ist. Shibboleth ist ein System, dass die Authentifizierung, Autorisierung und Rechteverwaltung für ein angeschlossenes System übernehmen kann. Dies sorgt allerdings für einige Einschränkungen auf Seiten der Anwendung. Diese muss einige Aufgaben aus der Hand geben.

Mycore ist aber ein flexibles System, das viele Möglichkeiten zur Konfiguration bietet. Gleichzeitig ergeben sich durch die internen Prozesse komplexe Szenarien betreffend der Zugriffsbeschränkungen. Daraus folgt, dass die reibungslose Zusammenarbeit beider Systeme nicht ohne Kompromisse zu erreichen ist. Bei einem Mycore System, das die komplette Funktionalität in Anspruch nimmt, ist der Einsatz von Shibboleth sicherlich nur bedingt sinnvoll. Mit dem gewünschten Gast-Zugang bleibt nur Shibboleth mit Lazy Session. Dies bringt die zuvor gezeigte Einschränkungen für das System mit sich. Ob dies in einem produktiven Mycore System akzeptiert wird, bleibt zumindest fraglich.

Shibboleth ist ein sehr mächtiges System, das mit der Einhaltung des SSO-Prinzips ein zentrales Ziel besitzt. Es übernimmt den kompletten Schutz für eine angeschlossene Ressource. Solange für diese ein grundsätzlicher Schutz durch Shibboleth sinnvoll ist, bietet sich dessen Einsatz uneingeschränkt an. Dadurch können viele Webanwendungen effektiv und einfach geschützt werden. Der Nutzer hat durch das SSO-Prinzip auch einen zentralen Vorteil. Überschreitet die Webanwendung aber eine gewisse Stufe der Komplexität, wie dies in Mycore der Fall ist durch eine eigene Nutzerverwaltung mit verschiedenen Rechten und Rollen der Nutzer, mit einen Zugang über die Kommandozeile, usw. , so kommt es zu einem Konflikt. Shibboleth ist dann zu restriktiv für eine offenes und flexibles System, wie Mycore es ist. Daher bleibt letztendlich die Frage, ob die Vorteile durch den Einsatz von Shibboleth in Mycore die daraus resultierenden Nachteile überwiegen.

Ein Ansatz, um die Kompatibilität der beiden Systeme zu erhöhen, ist die Konzentration auf den Einsatz von Shibboleth ohne Lazy Session im Rahmen von Mycore. Mycore

fungiert hierbei lediglich als Dokumentenserver und die Aktionen der meisten Nutzer sind von passiver Natur, sie greifen auf die Dokumente nur lesend zu. Diese können in Mycore anonym bleiben, und Shibboleth kümmert sich um die Beschränkung des Zugriffs auf die gewünschten Nutzer. Die wenigen Nutzer, die Dokumente zum Server hinzufügen, werden über die interne Nutzerverwaltung von Mycore abgedeckt.

Der Einsatz von LDAP ist dagegen einfacher und unkomplizierter im Vergleich zu Shibboleth. Mycore hat durch LDAP keinerlei Einschränkungen in der Funktionalität. Der Zugriff auf den LDAP Server läuft im „Hintergrund“ und der Nutzer bekommt dies nicht mit. Die Integration der gewünschten Funktionalität in den Code von Mycore ist schneller als bei Shibboleth zu realisieren. Gleichzeitig haben einige Einrichtungen, die ein Mycore System einsetzen, einen LDAP Server zur Verwaltung ihrer Nutzer. Zusätzlich lässt sich auch eine Authentifizierung über die Kommandozeile mit LDAP realisieren.

LDAP erfüllt offenbar genau die Anforderungen, die sich durch das flexible und offene System von Mycore ergeben. Die Kontrolle über den Prozess der Authentifizierung bleibt in der Hand von Mycore. Daher kommt es auch zu keinen Konflikt, wie dies bei Shibboleth der Fall ist. Zusätzlich ergeben sich noch mehr Möglichkeiten, wenn auf die weiteren Daten der Nutzer, die auf dem LDAP Server gespeichert sind, zugegriffen wird. Mycore kann also durch den Einsatz von LDAP ohne Einschränkungen oder Nachteile erweitert werden.

Richtet man den Blick von einem Mycore System mit LDAP allerdings auf eine Organisation, die ein Shibboleth System einsetzt, so erkennt man schnell den Verlust. Das SSO-Prinzip wird dadurch nicht eingehalten. Durch den Zugriff auf LDAP wird zwar eine zentrale Nutzerdatenbank eingesetzt, aber der Nutzer muss sich in einem Mycore System jeweils neu anmelden.

Zusätzlich bleibt zu erwähnen, dass der Zugriff auf einen LDAP Server nicht von jeder Ressource aus so einfach möglich ist, vor allem wenn diese nur aus statischen Inhalten besteht. Shibboleth hingegen kann alle Schutzfunktion für so eine Art von Ressource übernehmen. Es bietet daher für ein größeres Spektrum an web-basierten Diensten eine Einsatzmöglichkeit. Shibboleth ist flexibler und folglich für das Management des Zugriffs vieler Ressourcen einer Organisation die bessere Alternative.

Letztendlich bleibt festzustellen, dass die Integration einer LDAP basierten Authentifizierung in Mycore sicherlich sinnvoll ist. Der Einsatz von LDAP wäre optional und würde in jedem Fall eine Bereicherung für Mycore darstellen. Dabei gibt es einige Szenarien im universitären Umfeld, bei denen sich ein Zugriff auf einen bestehenden LDAP Server anbieten würde.

Im Sinne des SSO-Prinzips, das Shibboleth durch seine Architektur verinnerlicht hat, sollte der Blick nicht von Shibboleth abgewendet werden. Für Mycore Systeme, die eher als Dokumentenserver fungieren, bleibt der Einsatz von Shibboleth eine Alternative. Auch bietet die neue Funktionalität in Shibboleth 2.0 wahrscheinlich eine Verbesserung für Mycore.

Abbildungsverzeichnis

Listings

Literaturverzeichnis

[1] Shibboleth Webseite im Rahmen des Internet2-Projekts

`http://shibboleth.internet2.edu/`

[2] Wiki-Seite des Shibboleth Projekts.

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

[3] Tom Scavo, Scott Cantor. *Shibboleth Architecture.*

`shibboleth.internet2.edu/shibboleth-documents/`

`shibboleth.internet2.edu/docs/draft-mace-shibboleth-tech-overview-latest.pdf`

[4] Valery Tschopp, Lukas Haemmerle, Patrik Schnellmann. *Install Shibboleth Service Provider 1.3 on Debian 3.1.*

`www.switch.ch/aai/docs/shibboleth/SWITCH/1.3/sp/insatll-sp-1.3-debian.html`

[5] Mycore - Dokumentation: *Overview, User Guide, Programmer Guide, Quick Installation Guide - DocPortal.*

`http://www.mycore.de/content/main/documentation.xml`

[6] OASIS - Organization for the Advancement of Structured Information Standards (Webseite)

`http://www.oasis-open.org/committees/security/`

[7] nmi-edit - Identity and Access Management for Higher Education and Research *EduPerson Object Class Specification*

`http://www.nmi-edit.org/eduPerson/internet2-mace-dir-eduperson-200604.html`

[8] Englische Wikipedia Seite über SAML

`http://en.wikipedia.org/wiki/SAML`

[9] AAR - Verteilte Authentifizierung, Autorisierung und Rechteverwaltung (Webseite)

`http://aar.vascoda.de/`

[10] Dr. Jochen Lienhard, Franck Borel *Installation und Konfiguration des Shibboleth-Service-Provider*

aar.vascoda.de/docs/install_sp_1.3.pdf

[11] Lajos Moczar. *Tomcat 5 - Einsatz in Unternehmensanwedungen mit JSP und Servlets.* Addison-Wesley, Martin-Kollar-Straße 10-12, 81829 München, Germany.

[12] Stefan Zörner (Hrsg.), Jörg Wegener. *LDAP für Java-Entwickler* Software & Support Verlag GmbH, Frankfurt, Germany.

[13] Martin Walser, Dirk von Suchodoletz. *Hereinspaziert - Die Pforten zur Linux-Maschine: Pluggable Authentication Modules.* Linux-Magazin 05/04.

[14] C. Troy Popplewell. *Configuring Tomcat5 and Apache2 with Virtual Hosts using mod_jk*

http://www.howtoforge.com/apache2_tomcat5_mod_jk

[15] Scott Oaks. *Java Security.* O'Reilly & Associates, Inc., 101 Morris Street, Sebastopol, CA95472.

[16] Qusay H. Mahmoud. *Java Authentication and Authorization Service (JAAS)in Java 2, Standard Edition (J2SE) 1.4*

http://java.sun.com/developer/technicalArticles/Security/jaasv2/index.html

[17] Sun Mircosystems. *The J2EE(TM) 1.4 Tutorial.*

http://java.sun.com/j2ee/1.4/docs/tutorial/doc/index.html

[18] Thomas Scheffler. *Analyse und Vergleich des Mycore „Content Repository" Rahmenwerks mit dem Java Content Repository Standard JSR-170 im Kontext einer offenen und erweiterbaren Repository-Schnittstelle (Diplomarbeit).* Institut für Informatik, Lehrstuhl für Datenbanken und Informationssysteme, Jena, Germany.

[19] Thomas Bender. *Benchmark von ECM Systemen am Beispiel einer ePublishing Anwendung auf Basis von Mycore (Diplomarbeit).* Institut für Informatik, Lehrstuhl für Kommunikationssysteme, Freiburg, Germany.