INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Masterarbeit

Identity Management für dynamische virtuelle Föderationen unter Verwendung einer Trusted Third Party

Michael Grabatin

INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Masterarbeit

Identity Management für dynamische virtuelle Föderationen unter Verwendung einer Trusted Third Party

Michael Grabatin

Aufgabensteller:	Priv.-Doz. Dr. Wolfgang Hommel
Betreuer:	Stefan Metzger Daniela Pöhn
Abgabetermin:	31. Oktober 2014

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

München, den 31. Oktober 2014

.....

(Unterschrift des Kandidaten)

Abstract

Diese Arbeit betrachtet Föderiertes Identity Management (FIM) unter Verwendung eines neuen Konzeptes für dynamische virtuelle Föderationen. Dieses Konzept ermöglicht es, anstelle der üblicherweise von Föderationen verwendeten geographischen Strukturen, mit geringem Aufwand, bedarfsorientiert dynamische Föderationen zu erstellen. Diese Föderationen werden schnell und flexibel, durch den Benutzer initiiert, erstellt und nach der Verwendung wieder aufgelöst.

In dieser Arbeit werden zur Entwicklung des Konzeptes dynamischer virtueller Föderationen zunächst die herkömmlichen Föderationskonzepte von Projekten, Communities, geographisch strukturierten Föderationen und Inter-Föderationen betrachtet. Dazu werden die an ein FIM-System gestellten Anforderungen, sowohl aus Sicht des Konzeptes zu dynamischen virtuellen Föderationen als auch aus Sicht herkömmlicher FIM-Konzepte betrachtet, abgeleitet und deren Gemeinsamkeiten und Unterschiede analysiert.

Diese Anforderungen werden daraufhin mit den beiden im Universitätsumfeld bekanntesten FIM-Implementierungen Shibboleth und simpleSAMLphp sowie einigen für bestimmte Teilaspekte der Anforderungen zuständigen Erweiterungen verglichen. Der Vergleich zeigt, dass keine Lösung oder Kombination von Lösungen geeignet ist, alle Anforderungen zu erfüllen. Als Grundlage für eine Erweiterung eignet sich jedoch Shibboleth am besten.

Daher wird, aufbauend auf Shibboleth, ein Konzept entwickelt, das die für den Aufbau dynamischer virtueller Föderationen nötigen Erweiterungen umsetzt. Für das dynamische Management der virtuellen Föderationen ist in dem entwickelten Konzept eine Trusted Third Party (TTP) zuständig. Diese koordiniert den für einen Föderationsaufbau nötigen Metadatenaustausch und die Konvertierung von eventuell unterschiedlichen bzw. die Ableitung unbekannter Attribute. Zur Demonstration dieses Konzeptes wird dieses als Prototyp implementiert.

Dieser Prototyp wird daraufhin in einer aus acht virtuellen Maschinen bestehenden Testumgebung analysiert. Das hier betrachtete Szenario besteht aus zwei Föderationen und zwei externen Providern. Hier zeigt sich, dass der Einsatz einer TTP einen Vorteil gegenüber herkömmlichen Lösungen bietet.

Inhaltsverzeichnis

1. Einleitung

Das Anmelden bei einem Webdienst ist für die meisten Menschen inzwischen ein alltäglicher Vorgang. Dazu wird im Regelfall von dem Benutzer eine Kombination aus Benutzername und Passwort abgefragt. Falls diese Kombination korrekt ist, also bei der Gegenstelle in dieser Form bekannt ist, kann der Benutzer über seinen Benutzernamen identifiziert und die eindeutige Kombination aus Benutzername und Passwort authentifiziert werden. Nach der Authentifizierung wird der Benutzer üblicherweise von der Gegenstelle autorisiert, eine Menge von Berechtigungen und Funktionen zu verwenden. Authentifizierung und Autorisierung sind allerdings nicht direkt voneinander Abhängig, auch ein nicht authentifizierter Benutzer kann autorisiert bzw. ein authentifiziert Benutzer nicht autorisiert sein eine Funktion zu verwenden.

Diese Art der Authentifizierung und Autorisierung wird heute an vielen Stellen genutzt. Zum Beispiel am privat genutzten Computer, E-Mail-Accounts, sozialen Netzwerken, Online-Shops und beim Online-Banking. Sowohl für den Dienstleister als auch für den Benutzer ist es wichtig, dass die Authentifizierung und Autorisierung einfach und schnell möglich ist, trotzdem aber ein möglichst hohes Maß an Sicherheit bietet. Ein schneller, einfacher Login-Vorgang ist relevant, da dadurch die Bereitschaft der Benutzer steigt, einen Account zu registrieren und auch regelmäßig zu verwenden. Die Sicherheit des Logins ist sowohl für Benutzer als auch Dienstleister wichtig, um private Daten zu schützen und Missbrauch zu verhindern.

Der Begriff Informationssicherheit kann durch die drei Ziele Vertraulichkeit, Integrität und Verfügbarkeit definiert werde [Bre+11]. Das Ziel der Vertraulichkeit beschreibt, dass sichergestellt sein soll, dass nur berechtigte Personen Zugang zu geschützten Daten haben dürfen. Dieses Ziel kann zum Beispiel durch eine geeignete Verschlüsselung erreicht werden. Das zweite Ziel, die Sicherstellung der Integrität von Daten, soll verhindern, dass Daten unautorisiert und eventuell auch unbemerkt modifiziert werden können. Eine Methode zur Überprüfung der Integrität sind kryptographische Prüfsummen. Die Verfügbarkeit, die das dritte Ziel der IT-Sicherheit ist, beschreibt, dass zugangsberechtigte Personen ohne Störung auf den von ihnen genutzten Dienst zugreifen können. Dieses Ziel kann durch entsprechende Redundanz erfüllt werden.

Gerade die Sicherheit von Benutzerkennungen steht regelmäßig im Fokus der Allgemeinheit mit Nachrichten über im Internet angebotene Datensätzen mit großen Mengen von Zugangsdaten zu E-Mail-Accounts [Bee14] sowie immer wieder auftauchenden Sicherheitslücken in

diversen Betriebssystemen [Eik14] und häufig verwendeten Programmbibliotheken [Sch14]. Die Schwachstellen bei der Sicherheit werden durch unterschiedliche Fehler verursacht. Zur Betrachtung der Sicherheit eines Login-Vorgangs kann man daher zunächst zwischen der technischen und der organisatorischen Sichtweise unterscheiden.

Empfehlungen für die Verbesserung der Sicherheit aus technischer Sicht enthalten Hinweise auf die Verwendung von Verschlüsselungstechnologie wie SSL/TLS, welche im Internet zum Beispiel durch die Verwendung von HTTPS gekennzeichnet ist. Zudem wird Dienstleistern empfohlen, Passwörter nur als gesalzenen Hash zu speichern [BSIKV14]. Das bedeutet, dass aus dem in der Datenbank eines Dienstleisters gespeicherten „Passwort“ nicht das eigentliche Passwort des Benutzers abgeleitet werden kann. Damit kann verhindert werden, dass falls Angreifer die Datenbank auslesen können, keine weiteren Accounts kompromittiert werden können, bei denen der Benutzer das gleiche Passwort verwendet hat.

Aus organisatorischer Sicht sorgen Passwort-Richtlinien, wie sie von nahezu jedem Dienstleister oder von IT-Sicherheits-Organisationen herausgegeben werden, dafür, dass Benutzer möglichst sichere Passwörter wählen und mit ihnen verantwortungsvoll umgehen. Betont wird dabei darauf zu achten, dass eine HTTPS-Verbindung zu der Seite hergestellt, keinesfalls das selbe Passwort bei mehreren Diensten und generell keine einfach zu erratenden Passwörter verwendet werden.

Da bei der Vielzahl an Accounts, die inzwischen jeder verwalten muss, häufig nicht für jeden Account ein eigenes Passwort verwendet wird, bieten alternative Authentifizierungs- und Autorisierungs-Konzepte hier Abhilfe. Große Unternehmen wie Google, Microsoft und PayPal unterstützen zum Beispiel Projekte wie OpenID und OAuth. Diese Projekte erlauben es zur Anmeldung bei einem neuen Dienst zum Beispiel seinen Google-Account zu verwenden. Bei der Authentifizierung wird der Benutzer zu einer speziellen Loginseite von Google weitergeleitet. Dort wird dem Benutzer gezeigt, bei welchem Dienst er sich mit seinem Google-Account einloggen möchte und welche Informationen der neue Dienst von Google erhält. Ist der Benutzer damit einverstanden, wird er mit den von Google beglaubigten Informationen zurück zur Seite des neuen Dienstes geleitet. Dadurch reicht der Zugriff auf den Google-Account, um den neuen Dienst zu verwenden und es wird kein neues Passwort benötigt.

Dieses Prinzip wird Federated Identity Management oder Föderiertes Identity Management (FIM) genannt. Angewendet wird es zum Beispiel auch bei der Anmeldung einiger Smartphone-Apps, bei denen ein Facebook- oder Twitter-Account als Login verwendet werden kann.

Ein aus Benutzersicht ähnliches System wird auch an Hochschulen verwendet. FIM beschreibt die Verwendung einer Identität in einer organisationsübergreifend verteilten Umgebung. Zum Beispiel können LMU-Studenten mit der gleichen Benutzername/Passwort-Kombination ihre E-Mails abrufen, den WLAN-Zugang verwenden, Vorlesungen belegen und Hausaufgaben zu Übungen abgeben, an e-Learning-Kursen anderer Universitäten teilnehmen sowie auf die Leistungsnachweise zugreifen. Diese Dienste werden von unterschiedlichen Organisationen erbracht. Das Identity Management sowie der Betrieb der E-Mail-Accounts und des WLANs wird zum Beispiel für die Ludwig-Maximilians-Universität (LMU) vom Leibniz-Rechenzentrum (LRZ) übernommen. Andere Dienste wie die Verwaltung der persönlichen Daten oder belegten Kurse werden sowohl von der Universität als auch direkt von

der Fachschaft oder den Instituten betrieben.

LMU und LRZ sind sehr eng miteinander verbundene Organisationen. FIM ist allerdings nicht auf solche Situationen beschränkt. So können Studenten sich mit ihren LMU-Accounts bei der Online-Bibliothek SpringerLink anmelden und dort kostenlos Fachbücher herunterladen oder sich gegenüber dem Microsoft-Store als für Studentenrabatte berechtigt ausweisen. Dieser Dienst steht natürlich nicht nur LMU-Studenten zur Verfügung, sondern Studenten aus Universitäten in ganz Deutschland beziehungsweise weltweit.

Das Einrichten und Aufbauen der Vertrauensbeziehung zwischen SpringerLink, einem Service Provider (SP) und den Universitäten, die in diesem Beispiel als Identity Provider (IDP) auftreten, ist ein komplizierter Prozess. Damit nicht jede Universität einzeln Informationen mit jedem Service Provider austauschen muss, werden Föderationen gegründet. Diese, für das Einrichten von Vertrauensbeziehungen bei FIM benötigten Informationen werden Metadaten genannt. Diese Metadaten müssen unter anderem Informationen enthalten, die den Dienst bzw. eine FIM spezifische Komponente identifizieren, angeben, welche Zertifikate zum Verifizieren von signierten Nachrichten verwendet werden sollen und welche Kommunikationsendpunkte, also URLs, unter denen ein Dienst erreichbar ist, existieren. Innerhalb dieser Föderationen gelten bestimmte Voraussetzungen und vertraglich festgehaltene Bestimmungen, die es erleichtern, eine Vertrauensbeziehung zwischen SP und IDP herzustellen.

In vielen Ländern gibt es nationale Föderationen, die sich um die Zusammenarbeit der Hochschulen und anderer Organisationen eines Landes kümmern. In Deutschland wird diese nationale Föderation von dem Deutschen Forschungsnetz (DFN) verwaltet. Da diese Föderation eine Authentifizierungs- und Autorisierungs-Infrastruktur (AAI) darstellt, wird die von dem DFN betriebene Föderation DFN-AAI abgekürzt.

Da nicht nur nationale Hochschulen und Forschungseinrichtungen kooperieren, sondern auch häufig gemeinsam zwischen internationalen Organisationen Projekte durchgeführt werden, hat sich herausgestellt, dass nationale Föderationen meist nicht ausreichen. Daher gibt es Bestrebungen auch auf internationaler Ebene Föderationen zu etablieren. Ein Ansatz besteht darin, dass diese Föderationen dann nicht einzelne Organisationen als Mitglieder haben, sondern (nationale) Föderationen. Dieses Prinzip wird Inter Federation Identity Management (IFIM) genannt.

Eines dieser Projekte ist eduGAIN, dass von den GÉANT-Partnern als eine europäische Inter-Föderation geschaffen wurde. Die Teilnahme an diesem Zusammenschluss von Föderationen ist aber nicht nur auf europäische Länder beschränkt. Abbildungen 1.1 und 1.2 zeigen die Adaptierung von eduGAIN in Europa bzw. der ganzen Welt. Dabei sind hellgrün markierte Länder Mitglied von eduGAIN, gelbe im Prozess des Beitretens und dunkelgraue Kandidaten, die an einem Beitritt interessiert sind.

Europaweit ist die Akzeptanz und Implementierung von eduGAIN schon sehr weit vorangeschritten und beinahe abgeschlossen. Weltweit gesehen fehlen aber noch einige Länder wie die Vereinigten Staaten von Amerika und mehrere asiatische Länder.

Der Zusammenschluss mehrerer Organisationen in Föderationen und Inter-Föderationen erleichtert die Einrichtung von Vertrauensbeziehungen zwischen Föderationsmitgliedern deutlich. Innerhalb von Föderationen können durch eine zentrale Instanz Metadaten automatisiert an die Mitglieder verteilt und aktualisiert werden, wodurch die Skalierbarkeit des

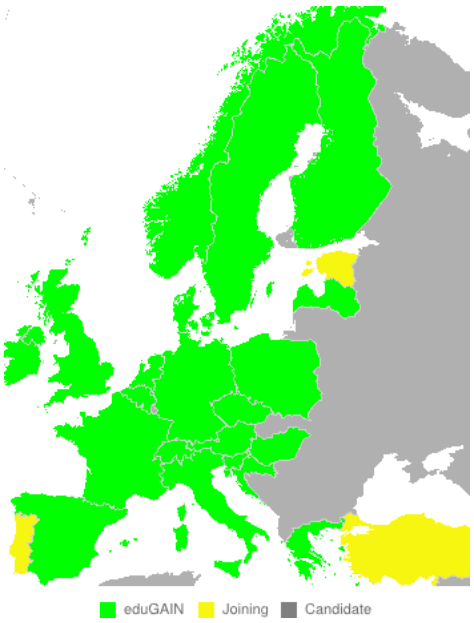

Abbildung 1.1.: Fortschritt eduGAIN in Europa [edu14a]

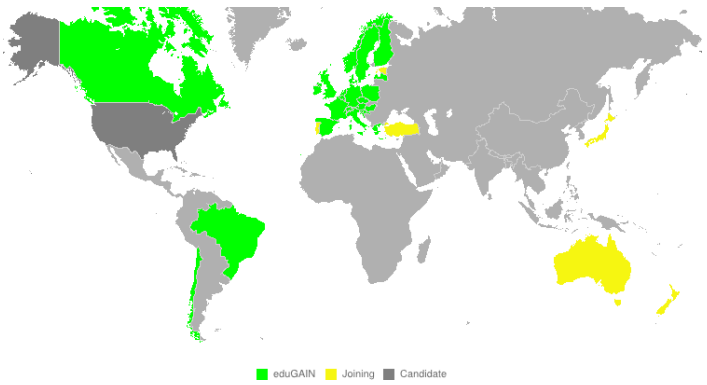

Abbildung 1.2.: Fortschritt eduGAIN weltweit [edu14a]

Systems erhöht wird.

Die beiden Ansätze FIM und IFIM weisen jedoch einige Schwachstellen auf, welche im nächsten Abschnitt erläutert werden.

1.1. Motivation

Damit Benutzer nicht für jeden Dienst, den sie bei einer anderen Organisation verwenden möchten, einen eigenen Account anlegen und dessen Informationen verifizieren müssen, kann ihre Heimatorganisation (zum Beispiel eine Universität) durch Föderiertes Identity Management erlauben, die dort vorhandenen Identitäten auch bei Diensten anderer Organisation zu nutzen.

Der Ansatz des Föderierten Identity Management hat sowohl für die Dienstbetreiber als auch für die Nutzer einige Vorteile gegenüber der klassischen Authentifizierung mit dienstspezifischen Benutzername/Passwort Kombinationen:

- Durch Single-Sign-On (SSO) muss der Benutzer sein Passwort nicht für jeden Dienst, den er benutzen will, neu eingeben. Es reicht sich innerhalb definierter Intervalle – zum Beispiel einmal täglich – zu authentifizieren.
- Ein Nutzer benötigt nur ein gutes Passwort, anstatt sich für jeden Dienst ein neues überlegen zu müssen. Dadurch wird die Gefahr reduziert, dass das gleiche Passwort bei verschiedenen Diensten verwendet und so die Angriffsfläche über mehrere Dienste verteilt wird. Die Stärke dieses einen Passworts ist aber um so wichtiger, da es Zugang zu vielen Diensten in mehreren Organisationen bietet.
- Die Daten der Benutzer werden besser geschützt, da viele Dienste nicht unbedingt den Namen, Adresse und ähnliche Informationen benötigen. Stattdessen reicht es ihnen zu wissen, dass sie den Benutzer eindeutig identifizieren können. Dafür reicht ein von einem IDP generiertes Token, das dem SP keinen Rückschluss auf die Identität des Benutzers gibt.
- Die Anbieter eines Dienstes müssen selber keine Passwörter verwalten und können sich dadurch Support-Aufwand für das Zurücksetzten vergessener Passwörter sparen. Zusätzlich wird die Angriffsfläche eines Dienstes reduziert, wenn dort gar keine Zugangsdaten gestohlen werden können und die sichere Authentifizierung auf eine darauf spezialisierte Organisation übertragen wird.
- Der Dienst eines Service Providers (SP) kann bei dem Identity Provider (IDP) beim Login immer die aktuellen Attribute des Nutzers abfragen. So müssen Änderungen an dem Studiengang, der Fachsemesterzahl oder Prüfungsleistungen nicht manuell für fremde Dienste eingetragen werden. Der SP hat außerdem die Bestätigung, dass der entsprechende Student noch eingeschrieben ist und nicht zwischenzeitlich das Studium beendet hat.

Neben diesen Vorteilen ist ein Nachteil des Systems allerdings der große Einrichtungs- und Betriebsaufwand. Für die technische Verbindung von SP und IDP müssen die jeweiligen Metadaten ausgetauscht werden. Bei der Verwendung von Föderationen müssen dazu die einzel-

nen Dienste und Provider ihre Metadaten bei einer zentralen Registrierungsstelle hinterlegen. Diese fasst die Metadaten zusammen und veröffentlicht sie regelmäßig an ihre Mitglieder, welche die Metadaten dann in ihren Systemen konfigurieren und stetig aktualisieren müssen. Das Aktualisieren der Metadaten kann in den meisten Fällen allerdings automatisiert werden. Durch dieses Vorgehen werden bei einem Dienst bzw. Provider allerdings die Metadaten aller anderen Provider bzw. Dienste installiert. In der Praxis bleibt der Großteil dieser Metadaten ungenutzt, da die Benutzer eines Providers kein Interesse bzw. Nutzen davon haben, die häufig sehr spezialisierten Dienste anderer Dienstanbieter zu verwenden.

Die Aggregation von Metadaten für Föderationen ist bei einem Mischbetrieb, wie er in der Praxis häufig vorkommt, ebenfalls ein Problem. Zum Beispiel wenn eine nationale Föderation in einer Inter-Föderation vertreten ist, müssen die Metadaten für die Dienste gefiltert werden, um Inkonsistenzen und Duplikate zu vermeiden. Diese Situationen können auftreten, wenn ein Teil der Mitglieder der nationalen Föderation sowohl in den aggregierten Metadaten der Inter-Föderation als auch der nationalen Föderation auftauchen, sodass ein Provider, der beide Metadatensätze integriert, durch unterschiedliche Aktualisierungsintervalle, unter Umständen zwei unterschidliche Metadaten zu einem Dienste oder Provider haben könnte.

Die Situation wird noch verschärft, wenn ein Dienst oder Provider zusätzlich zu der Föderation externe, nicht in einer Föderation vertretene, Partner hat. Diese Partner können zwar konfiguriert werden, profitieren dann aber nicht von einer automatischen Metadatenaktualisierung, sodass jede Änderung manuell kommuniziert und implementiert werden muss. Zudem besteht die Möglichkeit, dass über die komplexen Beziehungen von Föderationen und Inter-Föderationen der Partner zu einem Zeitpunkt Teil eines Föderations-Metadatensatzes wird und wieder ein Duplikat auftritt.

Für den Benutzer, der einen neuen Dienst mit seinem Provider benutzen möchte, bedeutet der manuelle Austausch von Metadaten und anderen organisatorischen Informationen zwischen den Administratoren und Verantwortlichen der Organisationen, in der Regel lange Wartezeiten bis ein Dienst genutzt werden kann.

Für den reibungslosen Austausch zwischen SP und IDP müssen die verwendeten Formate für Daten eines Benutzer, die auch Attribute genannt werden, kompatibel sein. Benötigt der SP bestimmte Informationen von dem IDP, wie zum Beispiel den Namen, das Geburtsdatum oder das Studienfach des Benutzers, gibt es zur Speicherung dieser drei simplen Attribute schon viele mögliche Datentypen und Standards zur Darstellung. Einige IDPs speichern in ihrer Datenbasis die Attribute `firstName` und `lastName` andere einen `fullName`. Zur Speicherung eines Datums wie beispielsweise eines Geburtstags gibt es weltweit viele verschiedene Konventionen. Zum Beispiel das in den USA übliche Format `01/31/2014` (MDY), das in Europa verwendete `31.01.2014` (DMY) oder die Speicherung als Unix-Timestamp `1391122800`. Bei dem Austausch von Attributen muss daher definiert sein, welches Attribut wie dargestellt wird und ein Dienst muss eventuell Konvertierungen vornehmen, um der Definition zu entsprechen bzw. das Attribut in seinem System zu verwenden. Außerdem müssen Zertifikate und Kommunikationsendpunkte konfiguriert werden.

Um den Vorgang der Anpassung von IDPs und SPs zu vereinfachen, schließen sich Organisationen häufig in einer Föderation zusammen. Für eine Föderation wird typischerweise ein bestimmtes Datenschema definiert, dem alle Föderationsmitglieder entsprechen müssen. Die

Zusammenfassung und Standardisierung mehrerer Attribute wird auch Attributs-Schema genannt. Innerhalb von Föderationen wird durch diese Attributs-Schemata das Einrichten von IDP-SP-Beziehungen vereinfacht. Ein SP, der nicht Teil einer Föderation ist, braucht für alle IDPs einer Föderation theoretisch nur einen Regelsatz, um die Attribute zu konvertieren.

Dies hat in der Praxis aber zur Folge, dass mit steigender Mitgliederanzahl einer Föderation die Anzahl der Attribute, auf die sich alle einigen können, sinkt. Für eduGAIN sind zum Beispiel nur acht Attribute „recommended“, das heißt, in den meisten Fällen verfügbar [Lin11]. Weitere Attribute können durch Vereinbarung zwischen Mitgliedsorganisationen definiert werden. In der nationalen DFN-AAI werden hingegen schon 18 Attribute definiert, die für die Verwendung der meisten Dienste nötig sind. Von diesen Attributen sind sechs als „grundlegend“ und der Rest als „ergänzend“ klassifiziert [Gie+08]. Zusätzlich werden zwei weitere Schemata angegeben, die Attribute für den Bereich E-Learning enthalten.

Der Zusammenschluss zu einer Föderation ist nur dann nützlich, wenn die gemeinsam definierten Attribute für einen SP ausreichen seinen Dienst sinnvoll zu betreiben. Wenn er zusätzliche Attribute benötigt, müssen wieder die IDPs, die mit diesem SP zusammenarbeiten wollen, eine eigene Konfiguration erstellen. Die FIM- und besonders IFIM-Systeme sind dadurch in ihrer Skalierbarkeit begrenzt.

1.2. Fragestellung

Diese Arbeit beschäftigt sich mit der Fragestellung, wie das aktuelle Föderationskonzept im Föderierten Identity Management erweitert und dahingehend verbessert werden kann, um anstelle von geographisch strukturierten Föderationen bedarfsorientierte, sogenannte virtuelle Föderationen zu bilden. Diese virtuellen Föderationen sollen es Diensten, Nutzern und Projektpartner erlauben, innerhalb ihrer virtuellen Föderation genau die von ihnen benötigten Attribute und Rahmenbedingungen zu spezifizieren. Hierzu soll hauptsächlich aus technischer, aber auch organisatorischer Sicht untersucht werden, welche Unterschiede zwischen etablierten FIM-Verfahren und dynamischen virtuellen Föderationen existieren.

Diese virtuellen Föderationen sollen sehr dynamisch sein, das heißt das Erstellen, Beitreten und Auflösen der Föderation soll schnell durchgeführt werden können und möglichst wenig Aufwand erzeugen. Ein wichtiger Punkt für die Dynamik ist es daher, automatisch die nötigen Metadaten zwischen den beteiligten Organisationen austauschen zu können und aktuelle Datensätze ohne große Verzögerung in die lokalen Systeme bei IDP und SP zu integrieren. Hierzu wird ein Konzept für ein geeignetes Metadatensynchronisationsverfahren benötigt, das eventuell auf den von Föderationen oder Inter-Föderationen verwendeten Systemen basiert.

Außerdem soll untersucht werden, wie die von den verschiedenen Organisationen verwendeten unterschiedlichen Attribute und Schemata in einem dynamischen System automatisiert durch Konvertierungsregeln umgewandelt werden können.

Aus diesen Betrachtungen soll ein Konzept für einen Metadaten- und Konvertierungsregel-Verwaltungsdienst für dynamische virtuelle Föderationen erstellt werden. Dieses Konzept soll durch technische Beschreibungen wie eine Systemarchitektur, Datenmodellierung und

Schnittstellenbeschreibung erweitert und als Prototyp implementiert werden. Dabei soll auf die Kompatibilität mit der auf der Security Assertion Markup Language (SAML) basierenden und im Hochschulumfeld häufig eingesetzten Software Shibboleth geachtet, aber auch alternative Protokolle wie OpenID betrachtet werden. Zur Demonstration des Konzepts sollen Schnittstellen für Shibboleth Identity Provider und Service Provider implementiert werden und die Lauffähigkeit anhand eines Shibboleth Testbed demonstriert werden.

1.3. Vorgehen

Im nächsten Kapitel werden zuerst grundlegende Techniken und Standards des Föderierten Identitymanagements vorgestellt und beschrieben. Dabei wird auch auf den Ablauf zum Aufbau einer Vertrauensbeziehung, die im Rahmen dieser Arbeit nur aus technischer Sicht betrachtet wird, sowie die für eine Authentifizierung und Autorisierung eines Benutzers wichtigen Funktionen eingegangen. Diese Betrachtung ist notwendig, um in Kapitel 3 Szenarien und Probleme an den aktuellen Konzepten darzustellen und daraus Anforderungen an eine Erweiterung abzuleiten.

Den Anforderungen werden daraufhin in Kapitel 4 aktuell verwendete Implementierungen gegenübergestellt. Aus dieser Analyse werden ein oder mehrere Systeme ausgewählt, die entsprechend der Aufgabenstellung geeignet sind, zu einem Metadaten-Verwaltungsdienst erweitert zu werden.

Das dabei zu entwickelnde Konzept wird in Kapitel 5 beschrieben. Die Implementierung des entwickelten Konzeptes wird dann in Kapitel 6 beschrieben. Die Implementierung wird in einer Testumgebung auf virtuellen Maschinen getestet. Die Ergebnisse dieser Tests werden in Kapitel 7 dargestellt. In Kapitel 8 werden die Ergebnisse dieser Arbeit abschließend zusammengefasst.

2. Grundlagen des Föderierten Identity Managements

In diesem Kapitel werden einige grundlegende Begriffe und Technologien beschrieben, die für die weitere Arbeit relevant sind. Zunächst wird in Abschnitt 2.1 beschrieben was eine „Digitale Identität“ ist und welche Eigenschaften bzw. Attribute sie haben kann. Daraufhin werden verschiedene Standards zum föderierten Identitymanagement vorgestellt. In Abschnitt 2.3 wird das in Forschung und Wissenschaft häufig verwendete SAML und in Abschnitt 2.4 das von großen Internet-Unternehmen unterstützte OpenID vorgestellt.

2.1. Digitale Identität und Attribute

Digitale Identitäten oder in dem Kontext dieser Arbeit einfach nur Identitäten, werden in ISO/IEC 29115 „Information technology – Security techniques – Entity authentication assurance framework“ definiert als „Set of attributes related to an entity“ [ISO/IEC29115]. Eine Entität ist nach demselben Standard etwas einzeln und unterscheidbar Identifizierbares.

Also enthält eine Identität bestimmte Attribute bzw. Eigenschaften eines bestimmbaren Objekts. Im Falle von Benutzern enthalten diese Attribute möglicherweise einen Benutzernamen, Realnamen, Kontaktadresse und E-Mail-Adresse. Je nach Anwendungsfall, können auch speziellere Attribute gespeichert werden. Bei Universitäten sind das zum Beispiel Studiengang, Studienfach, Semesterzahl und Matrikelnummer.

Attribute können verschiedene Typen haben:

- **Einfache Attribute** bestehen nur aus einem atomaren Wert eines bestimmten Datentyps. Beispiel hierfür sind eine `ID` wie eine Matrikelnummer oder das `Alter` eines Benutzers.
- **Zusammengesetzte Attribute** sind Attribute, die aus einfachen Attributen zusammengesetzt sind. Zum Beispiel das Attribut `Name`, welches aus den Attributen `Vorname` und `Nachname`, zusammengesetzt ist.
- **Komplexe Attribute** können mehrere Werte enthalten und werden auch Multi-Value-Attribute genannt. Zum Beispiel kann ein Benutzer eine oder mehrere E-Mail-Adressen angeben.
- **Abgeleitete Attribute** können aus anderen Attributen abgeleitet werden. Als Beispiel kann aus dem Attribut `Geburtsdatum` das `Alter` oder aus der `Postleitzahl` der `Ort` abgeleitet werden.

2.2. Föderationen

In diesem Abschnitt werden Föderationen, die teilweise schon in der Einleitung vorgestellt wurden, mit einem Fokus auf den organisatorischen Prozessen näher betrachtet. Dazu werden zunächst in Abschnitt 2.2.1 klassische nationale Föderationen betrachtet. Darauf werden in Abschnitt 2.2.2 Inter-Föderationen und in Abschnitt 2.2.3 Föderationen für Communities, welche Zusammenschlüsse von Interessensgemeinschaften sind, sowie Projekte beschrieben.

2.2.1. Nationale Föderationen

Klassische nationale Föderationen entstehen aus Vereinbarungen, die zwischen mehreren Organisationen getroffen werden, mit dem Ziel die Authentifizierung und den Zugriff auf geschützte Ressourcen zwischen den beteiligten Organisationen zu vereinfachen. Die Föderation bietet den Benutzern der Organisationen dadurch einen bequemeren Zugang zu deren

Diensten. Dadurch können die Dienstbetreiber einer großen Zahl an Benutzern ihre Dienste zur Verfügung stellen, sowie zur Effizienssteigerung Dienste bei einer Organisation aggregiert werden, ohne das Benutzer neue Zugangsdaten benötigen.

Im Folgenden wird untersucht, welche Voraussetzungen zum Beitritt zu einer Föderationen nötig sind und wie die Verwendung, Weitergabe und Qualität von persönlichen Daten aus organisatorischer Sicht geregelt wird. Zunächst wird dazu die schon in der Einleitung erwähnte Föderation des Deutschen Forschungsnetz DFN-AAI betrachtet.

Die DFN-AAI beschreibt unter `https://www.aai.dfn.de/teilnahme/` die folgenden zur Teilnahme nötigen Schritte:

- **Anmeldung:** Um an der Föderation teilnehmen zu können, muss sich eine interessierte Organisation mit Service Provider oder Identity Provider zunächst bei der DFN-AAI registrieren. Dieser Schritt kann wenig bürokratisch per signierter Mail oder telefonisch erfolgen. Danach kann die Organisation der Testföderation DFN-AAI-Test teilnehmen. Für die Aufnahme in die Produktivumgebung, muss ein Vertrag beantragt und unterzeichnet werden.
- **Online-Verwaltung:** Nach der erfolgreichen Registrierung und Installation der Provider-Software können über ein Online-Portal die Metadaten der Organisation den anderen Teilnehmern zur Verfügung gestellt werden.
- **Föderations-Metadaten:** Nach dem Hochladen der providerspezifischen Metadaten müssen die Metadaten der Föderation in den lokalen Dienst integriert werden. Dabei wird zwischen den beiden Produktivumgebungen „Basic“ und „Advanced“ mit unterschiedlichen Verlässlichkeitsklassen unterschieden. Die Verlässlichkeitsklassen stellen unterschiedliche Voraussetzungen an die Aktualität und Genauigkeit der durch einen IDP herausgegebenen Daten. Die zusammengefassten Metadatensätze werden stündlich aktualisiert und sollen so konfiguriert werden, dass sie automatisiert in den von den Providern eingesetzten Diensten integriert werden.
- **Konfiguration:** Nach dem Austausch der Metadaten, muss der Dienst für die Test-Föderation konfiguriert werden. Dazu bietet die DFN-AAI Installationsanleitungen sowohl für IDPs als auch SPs unter Verwendung der Software Shibboleth. In der Test-Föderation können anhand von Systemen, die durch die DFN-AAI betriebenen werden, IDPs und SPs mit dem eigenen System getestet werden.
- **Produktivbetrieb:** Nachdem die Organisation in der Test-Föderation gezeigt hat, dass ihre Systeme korrekt konfiguriert sind und der Vertrag für Produktivumgebung unterzeichnet ist, kann von der Test- auf die Produktiv-Föderation umgestellt werden. Dazu werden die Metadaten der Organisation in die Metadaten der Produktiv-Föderation übernommen.

Die Voraussetzungen und der Prozess zum Beitritt ist bei den meisten Föderationen ähnlich. Unterschiede existieren jedoch bei der Aufnahme von kommerziellen Diensten in die Föderation sowie den Gebühren für die Aufnahme. Die finnische Föderation HAKA erhebt zum Beispiel für nicht staatliche oder universitäre bzw. wissenschaftliche Teilnehmer bzw. von diesen Organisationen beauftragten. Dienstleistern, sowohl für die Anmeldung als auch als jährlichen Beitrag eine Gebühr von 1000 Euro [Laa13]. Die meisten Föderationen verlangen allerdings keine Gebühren [Hae14].

2.2.2. Inter-Föderationen

Um die Einschränkungen, die bei der internationalen Kooperation von Organisationen mit dem System der klassischen nationalen Föderationen auftreten, zu umgehen, wurde das Konzept der Inter-Föderation entwickelt. Hier schließen sich mehrere Föderationen zusammen, so dass es theoretisch einen Metadatensatz mit den Teilnehmern aus allen Föderationen gibt.

Auch die, zuvor vorgestellte, DFN-AAI nimmt an der Inter-Föderation eduGAIN teil. Damit können deren Mitglieder theoretisch an eduGAIN teilnehmen. Um allerdings in den eduGAIN Datensatz aufgenommen zu werden, muss die jeweilige Organisation dies extra aktivieren („opt-in“) und zusätzlich zu den DFN-AAI-Metadaten die eduGAIN-Metadaten integrieren. In den Föderationen aus England und Schweden wird der Beitritt zu Inter-Föderationen wie eduGAIN nicht als „opt-in“ betrieben, sondern ist standardmäßig aktiviert mit der Option eines „opt-out“ [Hä14]. Da viele Organisationen einfach die Standardeinstellungen verwenden, sind daher aus Föderationen mit „opt-out“ viel mehr Mitglieder als denen mit „opt-in“ vertreten. Die genauen Zahlen hierzu werden in Abschnitt 3.2.2.2 am Beispiel der schon in Kapitel 1 erwähnten Inter-Föderation eduGAIN verglichen.

Die eduGAIN Initiative entstand aus dem GÉANT-Projekt, welches ein Zusammenschluss europäischer Wissenschafts- und höheren Bildungs-Einrichtungen ist. Innerhalb dieser Föderation schließen sich nationale Föderationen zusammen, um länderübergreifend Authentifizierung durchführen zu können. Durch den Zusammenschluss in eduGAIN und die Teilnahme vieler nationaler Föderationen ist es in vielen Fällen möglich geworden, Dienste und Projekte international zu betreiben und zu verwenden. Eine Liste der aktuell verfügbaren Dienste wird unter `http://monitor.edugain.org/coco/?show=list_sps` geführt.

Ein weiteres Inter-Föderationsprojekt ist Kalmar2 (`https://www.kalmar2.org`), welches die skandinavischen Länder (Finnland, Norwegen, Dänemark, Schweden, Island) in einer Föderation vereint.

Für die Aufnahme in eine Inter-Föderation muss ein IDP oder SP zunächst einer klassischen Föderation beitreten um über diese in die Metadaten der Inter-Föderation aufgenommen zu werden. Ein direkter Beitritt der Inter-Föderation ist nicht vorgesehen [edu14b] und würde zudem die Komplexität des Systems deutlich erhöhen. Dienstanbieter, deren nationale Föderation nicht Teil einer Inter-Föderation ist, stellt dies vor das Problem eine nationale Föderation auszuwählen, über die sie der Inter-Föderation beitreten können. Für eine nationale Föderation ist der Beitritt hingegen relativ einfach möglich, die Bedingungen der Teilnahme für eduGAIN regelt zum Beispiel die nur etwa eineinhalb Seiten lange Vereinbarung [Lin+13], welche vor dem Beitritt unterzeichnet werden muss.

2.2.3. Communities und Projekt Föderationen

Community- und Projekt-Föderationen werden von Gruppen von Wissenschaftlern gegründet, die auf einem bestimmte Gebiet forschen und sich dazu zusammen schließen möchten. Communities und Projekte basieren daher nicht, wie die meisten klassischen Föderationen, auf nationalen Grenzen, sondern den Themen, mit denen sich die Wissenschaftler beschäftigen. In manchen Fällen werden für die Forschung in diesen Communities sehr spezielle

Versuchsanlagen benötigt, die nur an wenigen Instituten verfügbar und daher von Wissenschaftlern aus der ganzen Welt genutzt werden. Communities, die solche speziellen Anlagen benötigen, sind zum Beispiel die Community für Hochenergiephysik (HEP), die sich um den Large Hadron Collider (LHC) am CERN entwickelt hat. Ein Ziel war es die Verarbeitung und Analyse der großen Datenmenge von mehreren dutzend Petabytes zu verteilen [Bro+13].

Ein anderes Beispiel ist die auf Biowissenschaften spezialisierte Community ELIXIR (`http://www.elixir-europe.org`). Diese Community muss ebenfalls große Mengen an Daten sicher verwalten und diese trotzdem für Wissenschaftler in ganz Europa zugänglich machen. Da Communities auf ein Themengebiet spezialisiert sind, haben sie im Allgemeinen speziellere Anforderungen an eine Authentifizierungs- und Authorisierungs-Infrastruktur als Föderationen, die mehrere Themen- und Interessengebiete vereinen.

Eine weitere Community, deren Ziel es ist, Service Provider mit Identity Providern aus den nationalen Föderationen zu verbinden, um den Benutzern aus dem Bereich Geistes- und Sozialwissenschaften Zugang zu Sprachressourcen und Verarbeitungsprogrammen zu bieten, ist die Common Language Resources and Technology Infrastructur (CLARIN, `http://clarin.eu`). Die Föderation wird daher auch als „Service Provider Föderation“ bezeichnet. CLARIN ist dabei in mehrere Center aufgeteilt, diese Center sind meistens Universitäten oder ähnliche wissenschaftliche Einrichtungen aus ganz Europa, die der Community Zugang zu ihren Diensten geben.

CLARIN legt dabei auf die Stabilität und nachhaltige Verlässlichkeit der Center großen Wert. Die Center werden daher vor der Aufnahme durch das CLARIN European Research Infrastructure Consortium (ERIC) überprüft und zertifiziert. CLARIN ERIC besteht momentan aus den Nationen Österreich, Bulgarien, Tschechien, Deutschland, Dänemark, Estland, Niederlande und Polen [CLARINAbout].

Um der CLARIN Community beizutreten, muss ein CLARIN Center eine Vereinbarung unterzeichnen, die der CLARIN ERIC eine Vollmacht gibt mit den nationalen Föderationen für die CLARIN Center Verträge auszuhandeln. Dadurch müssen die einzelnen Center nicht selbst mit allen teilnehmenden nationalen Föderationen Verträge aushandeln. Nach dem Beitritt, können die SPs in CLARIN die Metadaten der IDPs aus den nationalen Föderationen über eine von CLARIN aggregierte Metadatendatei in ihre Systeme integrieren. Zusätzlich gibt es eine Metadatendatei für den CLARIN IDP, der es Benutzern ohne eigene Heimat-IDP unter gewissen Bedingungen erlaubt einen Account zu erstellen. Die entgegengesetzte Richtung, die Metadaten der SPs in den jeweiligen nationalen Föderationen bekannt zu machen, ist etwas komplizierter, da es keinen einheitlichen Standard hierfür gibt. Daher müssen verschiedene Methoden, wie das Versenden per E-Mail oder das Aktualisieren der Daten über die jeweiligen Web-Interfaces der Föderationen verwendet werden [CLARINSPF].

Mit der Entwicklung von Inter-Föderationen wie eduGAIN, sind Communities wie CLARIN theoretisch nicht mehr relevant, da eduGAIN den europaweiten Zugriff auf die nationalen Föderationen als Ziel hat. Allerdings hat die aktuelle Implementierung von eduGAIN, wie schon im vorhergehenden Abschnitt beschrieben, gerade bei der Anzahl der IDPs und SPs die sich zum „opt-in“ zu eduGAIN entschlossen haben, Schwächen, wodurch eine Community wie CLARIN durchaus wichtig bleibt. Dies wird auch durch eine von CLARIN durchgeführte Untersuchung bestätigt [Uyt13].

Projekte sind üblicherweise noch enger auf ein spezielles Thema fokussiert als Communities. Sie haben meistens eine begrenzte Projektdauer und eine höhere Fluktuation der teilnehmenden Organisationen bzw. Personen. Daher sind sie im Vergleich zu den klassischen Föderationen und Communities sehr dynamisch. In den meisten Fällen gibt es hier keine dokumentierten Verfahren oder Bedingungen für den Beitritt. Die aktuellen Anforderungen an die AAI eines Projektes ergeben sich daher direkt aus der Art des Projektes und werden informell zwischen den Projektpartnern ausgehandelt.

Da Projekte aber auch größere Dimensionen annehmen können, zeigt das GÉANT-Projekt, welches inzwischen 41 Partner verbindet, um Dienste für verschiedene wissenschaftliche Einrichtungen und Projekte zu betreiben. Die Teilnehmer umfassen Organisationen aus 38 europäischen Ländern sowie die Organisationen DANTE (`http://www.dante.net`), welche das europäische 500Gbps Netzwerk für GÉANT betreibt und dem Forschungsnetzwerk TERENA (`http://www.terena.org`) [GEA14].

2.3. Security Assertion Markup Language (SAML)

Die Security Assertion Markup Language (SAML) ist ein durch das Security Services Technical Committee der Organization for the Advancement of Structured Information Standards (OASIS) herausgegebener XML-basierter Standard für den Austausch von Authentifizierungs-, Authorisierungs- und Attributsinformationen [Wis+05]. SAML kann im föderierten Identity Management dazu genutzt werden, zwischen Benutzern, Identity Providern und Service Providern Informationen auszutauschen.

Die erste Version SAML 1.0 wurde im November 2002 veröffentlicht und im darauffolgenden Jahr zu SAML 1.1 erweitert. Nachdem der Standard im Rahmen von Wissenschafts- und Forschungseinrichtungen weiträumig verwendet wurde, erschien 2005 mit SAML 2.0 eine neue Version, die noch immer aktuell ist. In dieser Arbeit bezieht sich der Begriff SAML immer auf SAML 2.0, da SAML 1.1 veraltet ist und daher nicht mehr eingesetzt werden sollte.

SAML baut auf anderen bewährten Technologien wie XML, XML Schema, XML Signature, XML Encryption, HTTP und SOAP auf. Es ist dadurch plattformunabhängig und setzt auf lose gekoppelte Systeme. Die meisten XML-Elemente innerhalb von SAML basieren auf einer Vererbungshierarchie, d.h. es gibt allgemeine abstrakte Elemente, von denen die tatsächlich verwendeten abgeleitet werden.

2.3.1. Genereller Ablauf

SAML spezifiziert diverse Abläufe rund um die Authentifizierung und Autorisierung, dieser Überblick konzentriert sich auf die Anmeldung mit dem Web-Browser-Single-Sign-On-Profil. Abbildung 2.1 zeigt dazu eine schematische Übersicht des Nachrichtenaustausches.

Wenn der Benutzer eine geschützte Webseite bei dem Service Provider aufruft, bekommt er eine Auswahl an IDPs präsentiert, aus denen er seinen IDP auswählt und dann zu einer speziellen Login-Seite bei dem gewählten IDP weitergeleitet wird. Diese Auswahl wird im

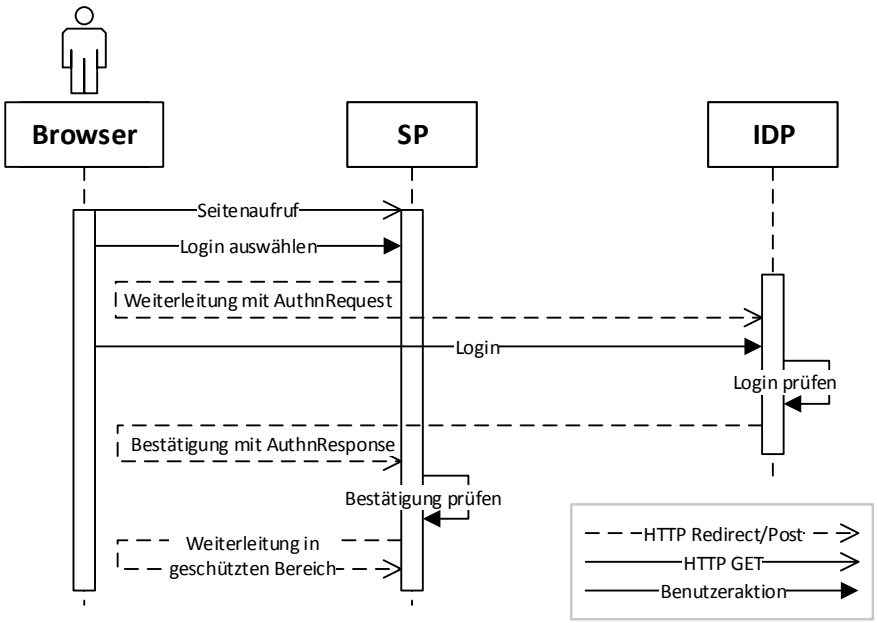

Abbildung 2.1.: Schematischer Ablauf bei der Anmeldung mit dem SAML Web-Browser-SSO-Profil.

Allgemeinen durch einen so genannten Discovery Service erstellt. Dieser kann, wie in Abbildung 2.1 gezeigt, lokal als Embedded Discovery Service bei dem Service Provider installiert sein oder als Centralized Discovery Service an einer zentralen Stelle von der Föderation betrieben werden. Falls es nur einen IDP für diesen Dienst gibt, kann der Benutzer auch direkt dorthin weitergeleitet werden. Die Weiterleitung zum IDP enthält einen Authentication Request, mit dem der SP die Authentifizierung des Benutzers bei dem IDP anfragt.

Der Benutzer muss sich daraufhin gegenüber dem IDP authentifizieren. Falls schon eine Sitzung für den Benutzer besteht, kann der IDP im Normalfall sofort eine Authentication Response an den SP zurückschicken. Als Authentifizierungsmethode wird üblicherweise eine Benutzername-/Passwort Kombination verwendet, SAML definiert die Art der Authentifizierung allerdings nicht, so dass auch andere zum Beispiel zertifikatsbasierte Methoden denkbar sind.

Wenn der IDP den Benutzer erfolgreich authentifizieren konnte, leitet er den Benutzer mit einer Authentication Response wieder zum SP zurück. Diese Authentication Response ist vom IDP signiert und üblicherweise nur eine begrenzte Zeit gültig.

Wenn der SP die Response des IDP erfolgreich validiert hat, kann er den Benutzer in den geschützten Bereich weiterleiten. Im Folgenden werden die für diesen Ablauf benötigten Elemente detaillierter beschrieben.

2.3.2. SAML Elemente

Die Spezifikation von SAML erstreckt sich über fünf Dokumente: Core [SAMLCore], Bindings [SAMLBind], Profiles [SAMLProf], Metadata [SAMLMeta] und Conformance [SAMLConf]. In diesen Spezifikationen werden fünf Kern-Elemente für SAML definiert und beschrieben, die im Folgenden kurz vorgestellt und in den nächsten Abschnitten näher betrachtet werden:

- **Assertions** (Kapitel 2.3.2.1) enthalten Informationen über Attribute, Authentifizierung oder Autorisierung.
- **Protokolle** (Kapitel 2.3.2.2) definieren einen Ablauf von Anfrage- und Antwortnachrichten zum Austausch von Assertions.
- **Bindings** (Kapitel 2.3.2.3) sind Verknüpfungen von SAML-Protokollen mit Übertragungsprotokollen wie HTTP/SOAP.
- **Profile** (Kapitel 2.3.2.4) spezifizieren wie SAML in einem bestimmten Anwendungsfall verwendet wird.
- **Metadaten** (Kapitel 2.3.2.5) enthalten Informationen über verwendete Zertifikate und Kommunikationsendpunkte.

2.3.2.1. Assertions

In SAML werden Assertions genutzt, um Aussagen über Subjekte zu machen, die innerhalb eines Protokolls verwendet werden. Das Dokument [SAMLCore] definiert den Aufbau solcher Assertions.

Innerhalb von Assertions gibt es drei verschiedene Arten von Statements:

- **Authentication** gibt an, dass ein Subjekt authentifiziert wurde. Zusätzlich kann hier angegeben werden, wie und wann die Authentifizierung durchgeführt wurde sowie ob diese erfolgreich war oder nicht.
- **Attribute** gibt Auskunft über Attribute eines Subjekts, zum Beispiel einen Namen oder Arbeitsgruppe.
- **Authorization Decision** teilt einem Anfragenden mit, ob ein Subjekt auf eine Ressource zugreifen darf oder nicht.

Listing 2.1 zeigt eine durch einen IDP generiete Assertion innerhalb einer Authentication-Response-Nachricht, die sowohl Authentifizierungs- als auch Attribute-Statements enthält. In diesem Abschnitt wird nur das `<saml2p:Assertion>` Element, das in Zeile 8 beginnt, betrachtet. Die Elemente, die dieses einschließen, werden in den nächsten Abschnitten beschrieben. Die Assertion selbst ist durch eine digitale Signatur integritätsgesichert. Das Format der Signatur folgt dem W3C-Standard XMLSig [XMLSig]. Mit dem SAML 2.0 Standard wird die Assertion normalerweise verschlüsselt übertragen, da dort der Aufbau aber schwerer zu zeigen ist, wird hier eine unverschlüsselte Assertion als Beispiel verwendet. Im Anhang in Listing A ist eine reale verschlüsselte Assertion dargestellt.

Listing 2.1: Beispiel für eine SAML 2.0 Authentication-Response-Nachricht

```
<?xml version="1.0" encoding="UTF-8"?>
<saml2p:Response xmlns:saml2p="urn:oasis:names:tc:SAML:2.0:protocol"
    Destination="https://gntb02.srv.lrz.de/Shibboleth.sso/SAML2/POST"
    ID="_88e396606e20b816bbb789ecde2a38fd"
    InResponseTo="_782da27af9c65de44b1db3f7022e472e"
    IssueInstant="2014-05-30T14:05:00.156Z" Version="2.0">
 <saml2:Issuer xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion"
    Format="urn:oasis:names:tc:SAML:2.0:nameid-format:entity"
   >https://gntb01.srv.lrz.de/IDP/shibboleth</saml2:Issuer>
 <saml2p:Status>
  <saml2p:StatusCode Value="urn:oasis:names:tc:SAML:2.0:status:Success"/>
 </saml2p:Status>
 <saml2:Assertion xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    ID="_b255d7f01d022c272d0a4485bed20126" IssueInstant="2014-05-30T14:05:00.156Z"
    Version="2.0">
  <saml2:Issuer Format="urn:oasis:names:tc:SAML:2.0:nameid-format:entity"
    >https://gntb01.srv.lrz.de/IDP/shibboleth</saml2:Issuer>
  <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
   <!-- ... -->
  </ds:Signature>
  <saml2:Subject>
   <saml2:NameID Format="urn:oasis:names:tc:SAML:2.0:nameid-format:transient"
    NameQualifier="https://gntb01.srv.lrz.de/IDP/shibboleth"
    SPNameQualifier="https://gntb02.srv.lrz.de/shibboleth"
    >_73614b544cb0c5030314e4db434bd1c1</saml2:NameID>
   <saml2:SubjectConfirmation Method="urn:oasis:names:tc:SAML:2.0:cm:bearer">
    <saml2:SubjectConfirmationData Address="93.233.86.106"
    InResponseTo="_782da27af9c65de44b1db3f7022e472e"
    NotOnOrAfter="2014-05-30T14:10:00.156Z"
    Recipient="https://gntb02.srv.lrz.de/Shibboleth.sso/SAML2/POST"/>
   </saml2:SubjectConfirmation>
  </saml2:Subject>
  <saml2:Conditions NotBefore="2014-05-30T14:05:00.156Z"
    NotOnOrAfter="2014-05-30T14:10:00.156Z">
   <saml2:AudienceRestriction>
    <saml2:Audience>https://gntb02.srv.lrz.de/shibboleth</saml2:Audience>
   </saml2:AudienceRestriction>
  </saml2:Conditions>
  <saml2:AuthnStatement AuthnInstant="2014-05-30T14:05:00.078Z"
    SessionIndex="_5312d63efd685cec5e5d4d0ccb454995">
   <saml2:SubjectLocality Address="93.233.86.106"/>
   <saml2:AuthnContext>
    <saml2:AuthnContextClassRef
     >urn:oasis:names:tc:SAML:2.0:ac:classes:PasswordProtectedTransport</
    saml2:AuthnContextClassRef>
   </saml2:AuthnContext>
  </saml2:AuthnStatement>
  <saml2:AttributeStatement>
   <saml2:Attribute FriendlyName="cn" Name="urn:oid:2.5.4.3"
    NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri">
    <saml2:AttributeValue xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:type="xs:string">Michael Grabatin</saml2:AttributeValue>
   </saml2:Attribute>
   <saml2:Attribute FriendlyName="eduPersonScopedAffiliation"
    Name="urn:oid:1.3.6.1.4.1.5923.1.1.1.9"
    NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri">
    <saml2:AttributeValue xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:type="xs:string">faculty@lrz.de</saml2:AttributeValue>
    <saml2:AttributeValue xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:type="xs:string">member@lrz.de</saml2:AttributeValue>
   </saml2:Attribute>
  </saml2:AttributeStatement>
 </saml2:Assertion>
</saml2p:Response>
```

Das erste SAML-Element der Assertion ist in Zeile 9 das `<saml2p:Issuer>`-Element, welches die EntityID des Providers angibt, der die Assertion ausgestellt hat. Auf dieses Element folgen einige Elemente aus dem XML-Signatur-Namespace, welche die Signatur der Assertion angeben. Da diese Elemente nicht direkt SAML-spezifisch sind, wurden sie hier zur besseren Lesbarkeit weggelassen. Das nächste SAML-Element `<saml2p:Subject>` in Zeile 14 gibt an, über wen die darauf folgenden Aussagen ausgestellt sind. Das Subjekt ist in diesem Fall durch die `<saml2p:NameID>` mit der Zeichenkette `_73614b544cb0c5030314e4db434bd1c1` identifiziert. Das Attribut `Format` zeigt an, welches Format die `NameID` hat. Möglich wären zum Beispiel auch eine E-Mail-Adresse, ein X.509 Subject Name oder ein Windows Domain Qualified Name. Der hier angegebene Typ `urn:oasis:names:tc:AML:2.0:nameid-format:transient` bedeutet, dass es sich um eine temporäre ID handelt, die für das Paar aus `NameQualifier` und `SPNameQualifier` gültig ist. Zusätzlich wird durch die `<saml2p:SubjectConfirmation>` eine Möglichkeit angegeben, mit der das Subjekt überprüft werden kann. Dazu werden dort die IP-Adresse des Subjekts, die ID der Request-Nachricht, ein Gültigkeitsfenster sowie der eigentliche Empfänger angegeben.

In dem Element `<saml2:Condition>` wird die Gültigkeit der Assertion auf einen bestimmten Zeitraum und einen bestimmten Empfänger eingeschränkt.

Das `<saml2:AuthnStatement>` gibt den Zeitpunkt und eine Session für die Authentifizierung an. Über die `<saml2:SubjectLocality>` wird angegeben, von welcher IP-Adresse aus sich das Subjekt authentifiziert hat. Der `<saml2:AuthnContext>` gibt über das Element `<saml2:AuthnContextRef>` und die URI `urn:oasis:names :tc:SAML:2.0:ac:classes:Password ProtectedTransport` an, dass die Authentifizierung mit einem Passwort über eine verschlüsselte Verbindung erfolgte.

Danach folgen zwei Attribute-Elemente, die innerhalb des Elements `<saml2:Attribute Statement>` gruppiert sind. Jeder der `<saml2:Attribute>` Elemente beschreibt in seinen Attributen über den `FriendlyName` einen leichter lesbaren Namen für das Attribut und in dem `Name`-Attribute einen Namen, dessen Format über das `NameFormat`-Attribut angegeben wird.

Der Wert der Attribute wird innerhalb des Attribut-Elements durch das `<saml2:Attribute Value>` gespeichert. Dieses Element hat den Typ `xs:anyType`, erlaubt also alle Werte, die durch den XML-Standard [XML] abgedeckt sind.

2.3.2.2. Protokolle

Durch SAML-Protokolle werden Request-Response-Abläufe definiert, die in [SAMLCore] spezifiziert sind. Dort werden Protokolle für die folgenden Aktionen definiert:

- Versenden einer oder mehrerer Assertions.
- Durchführen von Authentifizierung.
- Durch Artefakte benötigte Nachrichten empfangen.
- Single-Logout, d.h. nahezu gleichzeitiger Logout bei allen Diensten.
- Name Identifier registrieren oder entfernen.

Listing 2.2: Beispiel für eine SAML 2.0 Authentication-Request-Nachricht

```
<?xml version="1.0"?>
<samlp:AuthnRequest xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
    AssertionConsumerServiceURL="https://gntb02.srv.lrz.de/Shibboleth.sso/SAML2/POST"
    Destination="https://gntb01.srv.lrz.de/IDP/profile/SAML2/Redirect/SSO"
    ID="_782da27af9c65de44b1db3f7022e472e" IssueInstant="2014-05-30T14:03:00.156Z"
    ProtocolBinding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST" Version="2.0">
 <saml:Issuer xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
   >https://gntb02.srv.lrz.de/shibboleth</saml:Issuer>
 <samlp:NameIDPolicy AllowCreate="1"/>
</samlp:AuthnRequest>
```

- Name Identifier mapping.

Listing 2.2 zeigt die Request-Nachricht zu der Antwort, die im vorhergehenden Abschnitt in Listing 2.1 gezeigt wurde. Diese Nachricht fordert durch das `<samlp:AuthnRequest>`-Element die Authentifizierung eines Benutzers an. Der XML-Namespace `samlp` zeigt an, dass es sich um ein Element aus dem SAML-Protokoll-Namespace handelt.

Innerhalb des Requests wird über das Attribut `AssertionConsumerServiceURL` angegeben, an welche URL die Response-Nachricht geschickt werden muss. Das Attribut `Destination` gibt an, an welche Adresse der Request geschickt wurde. Über das Attribut `ID` wird ein eindeutiger Identifier für die Nachricht und über das Attribut `IssueInstant` der Zeitpunkt, an dem die Nachricht erstellt wurde, festgelegt.

Das `<saml:Issuer>` Element gibt an, wer den Request erstellt hat. Zusätzlich wird in dem Beispiel über das Element `<samlp:NameIDPolicy>` mit dem Attribut `AllowCreate` angegeben, dass der IDP eine NameID für das Subjekt erstellen darf. In der Standardeinstellung ist das nicht erlaubt, so dass der IDP schon einen Identifier erstellt haben muss.

2.3.2.3. Bindings

Bindings geben an, welches Transport-Protokoll für eine Request- oder Response-Nachricht verwendet wird. Das Transport-Protokoll ist unabhängig von dem verwendeten SAML-Protokoll. Die zur Verfügung stehenden Bindings werden in [SAMLBind] definiert.

Für den in Listing 2.2 gezeigten `AuthnRequest` wird zum Beispiel als `ProtocolBinding` der URI `urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST` angegeben. Das bedeutet, dass die Response-Nachricht mittels `HTTP POST` übermittelt werden soll. Um eine Protokoll-Nachricht, die durch ein XML-Dokument dargestellt ist, über `HTTP POST` zu verschicken, muss sie mit Base64 codiert werden. Base64 [RFC4648] übersetzt eine beliebige (binäre) Eingabedatei in ASCII-Zeichen, die geeignet dafür sind mit einem HTTP-Request verschickt zu werden.

Die von SAML [SAMLBind] spezifizierten Bindings sind:

- `HTTP Redirect` verwendet zum Übermitteln von Nachrichten URL-Parameter. Der HTTP-Standard setzt keine Grenze für die maximale Länge einer URL [RFC2616, Abschnitt 3.2.1]. In der Praxis ist die verwendbare Länge aber abhängig von dem

verwendeten Browser und der Server-Software [Plu10]. Daher rät auch SAML dazu, nur kurze Nachrichten mit diesem Binding zu übertragen.

- `HTTP POST` basiert auf der Übermittlung von HTML-Form-Daten. Auch hier wird zum Schutz der übermittelten Nachricht ein Base64-Encoding vorgeschrieben.
- `HTTP Artifact` übermittelt Referenzen auf die SAML-Nachrichten. Diese Methode kann verwendet werden, wenn der Browser des Benutzers nur HTTP Redirect unterstützt und trotzdem lange Nachrichten übertragen werden sollen.
- `SAML SOAP` verwendet zur Übertragung das ebenfalls XML-basierte SOAP [W3C07]. SOAP definiert eine Struktur aus Rahmen, Header und Daten. Innerhalb der Daten kann die SAML-Nachricht ohne zusätzliche Kodierung eingefügt werden. Eine SOAP-Nachricht darf immer nur ein SAML-Element enthalten. Zur Übertragung der SOAP-Nachricht wird wiederum HTTP verwendet.
- `Reverse SOAP (PAOS)` bietet die Möglichkeit in einem SAML-Request anzuzeigen, dass der Sender in der Lage ist SOAP-Nachrichten zu empfangen.
- `SAML URI` gibt eine URL zu einem SAML-Dokument an, das per HTTP abgerufen werden kann.

Die Bindings `HTTP Redirect` und `HTTP POST` werden in der Praxis am häufigsten verwendet.

2.3.2.4. Profile

SAML-Profile geben an, wie Assertions mit anderen Frameworks und Transport-Protokollen verwendet werden können. Zusätzlich kann durch ein Profil die Verwendung von verschiedenen SAML-Funktionen eingeschränkt oder explizit vorgeschrieben werden. Definiert sind Profile in [SAMLProf]:

- **SSO Profiles**: Eine Gruppe von Profilen, die zur Authentifizierung von Benutzern verwendet werden.
  - **Web-Browser-SSO Profile**: Beschreibt den Prozess der Authentifizierung über das SAML Authentication Request Protocol mit Hilfe von HTTP Redirect, HTTP POST und HTTP Artifact Bindings. Dieses Profil wird bei den meisten Logins über SAML verwendet.
  - **Enhanced Client or Proxy (ECP) Profile**: Verwendet SOAP und Reverse SOAP (PAOS) Bindings zur Authentifizierung einen Clients.
  - **Identity Provider Discovery Profile**: Mit diesem Profil kann ein Service Provider den Identity Provider eines Benutzers für das Web-Browser-SSO Profil bestimmen. Der Name des IDP wird dabei in einem Browser-Cookie in einer gemeinsamen Domain gespeichert. Eine 2008 veröffentlichte Erweiterung dieses Profiles sieht keine gemeinsame Domain mehr vor, da diese zwischen Organisationen schwer zu verwalten ist und verwendet stattdessen einen Parameter in der Antwortnachricht des Discovery Services [SAMLIDPDS].

  - **Single Logout Profile**: Über dieses Profil kann ein Identity Provider die Session eines Benutzers bei mehreren Service Providern nahezu gleichzeitig beenden.
  - **Name Identifier Management Profile**: Dieses Profil kann dazu verwendet werden, Name Identifier zu verwalten. Name Identifier sind Zeichenketten, die zwischen IDP und SP verwendet werden um einen Benutzer zu identifizieren. Mit diesem Profil kann dieser Identifier zum Beispiel geändert werden.

- **Artifact Resolution Profile**: Beschreibt das Vorgehen, ein durch ein HTTP Artifact Binding übermitteltes Artefakt abzurufen.
- **Assertion Query/Request Profile**: Beschreibt den Ablauf über synchrone Bindings wie zum Beispiel das SOAP Binding Attribute abzufragen.
- **Name Identifier Mapping Profile**: Wird verwendet um zwei Name Identifier eines Benutzers aufeinander abzubilden.
- **SAML Attribute Profile**: Spezifiziert nicht eindeutige die Namen von häufig benötigten SAML-Attributen.

Das Web-Browser-SSO Profil ist für die folgende Arbeit besonders von Bedeutung, da die vorgeschlagene Erweiterung dieses Profil erweitern wird.

2.3.2.5. Metadaten

Metadaten definieren eine standardisierte Methode, um die für die verwendeten Profile benötigten Informationen zwischen den IDP- bzw. SP-Instanzen verschiedener Organisationen auszutauschen. Dabei werden zum Beispiel Identifier, unterstützte Methoden für Bindings inklusive deren Endpunkte und Zertifikate sowie Schlüssel ausgetauscht [SAMLMeta]. Alle Metadaten spezifischen Informationen stammen aus [SAMLMeta].

Eine Metadaten-Datei beginnt immer mit einem der Elemente `<EntityDescriptor>` oder `<EntitiesDescriptors>`, je nachdem ob ein oder mehrere Objekte beschrieben werden. Ein `<EntitiesDescriptors>` kann sowohl Entity-Descriptoren als auch weitere Entities-Descriptoren enthalten. Ein `<EntityDescriptor>` muss eine `EntityID` enthalten, die das Objekt eindeutig identifiziert.

Im Folgenden wird der weitere Aufbau einer Metadatendatei anhand des LMU-IDP als Beispiel gezeigt. Diese Datei ist in Listing 2.3, in einer um die vollständigen Zertifikate gekürzten Version, dargestellt.

Innerhalb der Entity-Description können für das Objekt mehrere unterschiedliche Rollen angegeben werden. Die einzelnen Elemente folgen einer Vererbungshierarchie, welche in Abbildung 2.2 vereinfacht dargestellt ist. Das Root-Element dieser Hierarchie ist das abstrakte Element `<RoleDescriptor>`. Der `RoleDescriptor` legt einige Attribute fest, die für alle Rollen verwendet werden können bzw. müssen. Optionale Attribute sind zum Beispiel `ID`, `validUntil` und `cacheDuration`. Pflicht ist das Attribut `protocolSupportEnumeration`, welches die durch Leerzeichen getrennten URIs zu den unterstützten Protokollen enthält.

In diesem Beispiel instanziiert der LMU-IDP `<IDPSSODescriptor>`, welches wiederum von dem allgemeineren `<SSODescriptor>` abgeleitet ist, und `<AttributeAuthorityDescriptor>`.

Listing 2.3: Metadaten des LMU-IDP

```
<?xml version="1.0" encoding="UTF-8"?>
<EntityDescriptor xmlns="urn:oasis:names:tc:SAML:2.0:metadata"
    xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
    xmlns:shibmd="urn:mace:shibboleth:metadata:1.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    entityID="https://lmuIDP.lrz.de/IDP/shibboleth">
 <IDPSSODescriptor protocolSupportEnumeration="urn:mace:shibboleth:1.0
    urn:oasis:names:tc:SAML:1.1:protocol urn:oasis:names:tc:SAML:2.0:protocol">
  <Extensions>
   <shibmd:Scope regexp="false">lmu.de</shibmd:Scope>
  </Extensions>
  <KeyDescriptor>
   <ds:KeyInfo>
    <ds:X509Data>
     <ds:X509Certificate>MIIFjDCCBHSgAwIB[...]KNL0+E7LXkVRkH4=</ds:X509Certificate>
    </ds:X509Data>
   </ds:KeyInfo>
  </KeyDescriptor>
  <ArtifactResolutionService
    Binding="urn:oasis:names:tc:SAML:1.0:bindings:SOAP-binding"
    Location="https://lmuIDP.lrz.de:8443/IDP/profile/SAML1/SOAP/ArtifactResolution"
    index="1"/>
  <ArtifactResolutionService Binding="urn:oasis:names:tc:SAML:2.0:bindings:SOAP"
    Location="https://lmuIDP.lrz.de:8443/IDP/profile/SAML2/SOAP/ArtifactResolution"
    index="2"/>
  <NameIDFormat>urn:mace:shibboleth:1.0:nameIdentifier</NameIDFormat>
  <NameIDFormat>urn:oasis:names:tc:SAML:2.0:nameid-format:transient</NameIDFormat>
  <SingleSignOnService Binding="urn:mace:shibboleth:1.0:profiles:AuthnRequest"
    Location="https://lmuIDP.lrz.de/IDP/profile/Shibboleth/SSO"/>
  <SingleSignOnService Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST"
    Location="https://lmuIDP.lrz.de/IDP/profile/SAML2/POST/SSO"/>
  <SingleSignOnService
    Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST-SimpleSign"
    Location="https://lmuIDP.lrz.de/IDP/profile/SAML2/POST-SimpleSign/SSO"/>
  <SingleSignOnService Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Redirect"
    Location="https://lmuIDP.lrz.de/IDP/profile/SAML2/Redirect/SSO"/>
 </IDPSSODescriptor>
 <AttributeAuthorityDescriptor
    protocolSupportEnumeration="urn:oasis:names:tc:SAML:1.1:protocol
    urn:oasis:names:tc:SAML:2.0:protocol">
  <Extensions>
   <shibmd:Scope regexp="false">lmu.de</shibmd:Scope>
  </Extensions>
  <KeyDescriptor>
   <ds:KeyInfo>
    <ds:X509Data>
     <ds:X509Certificate>MIIFjDCCBHSgAwIB[...]KNL0+E7LXkVRkH4=</ds:X509Certificate>
    </ds:X509Data>
   </ds:KeyInfo>
  </KeyDescriptor>
  <AttributeService Binding="urn:oasis:names:tc:SAML:1.0:bindings:SOAP-binding"
    Location="https://lmuIDP.lrz.de:8443/IDP/profile/SAML1/SOAP/AttributeQuery"/>
  <AttributeService Binding="urn:oasis:names:tc:SAML:2.0:bindings:SOAP"
    Location="https://lmuIDP.lrz.de:8443/IDP/profile/SAML2/SOAP/AttributeQuery"/>
  <NameIDFormat>urn:mace:shibboleth:1.0:nameIdentifier</NameIDFormat>
  <NameIDFormat>urn:oasis:names:tc:SAML:2.0:nameid-format:transient</NameIDFormat>
 </AttributeAuthorityDescriptor>
</EntityDescriptor>
```

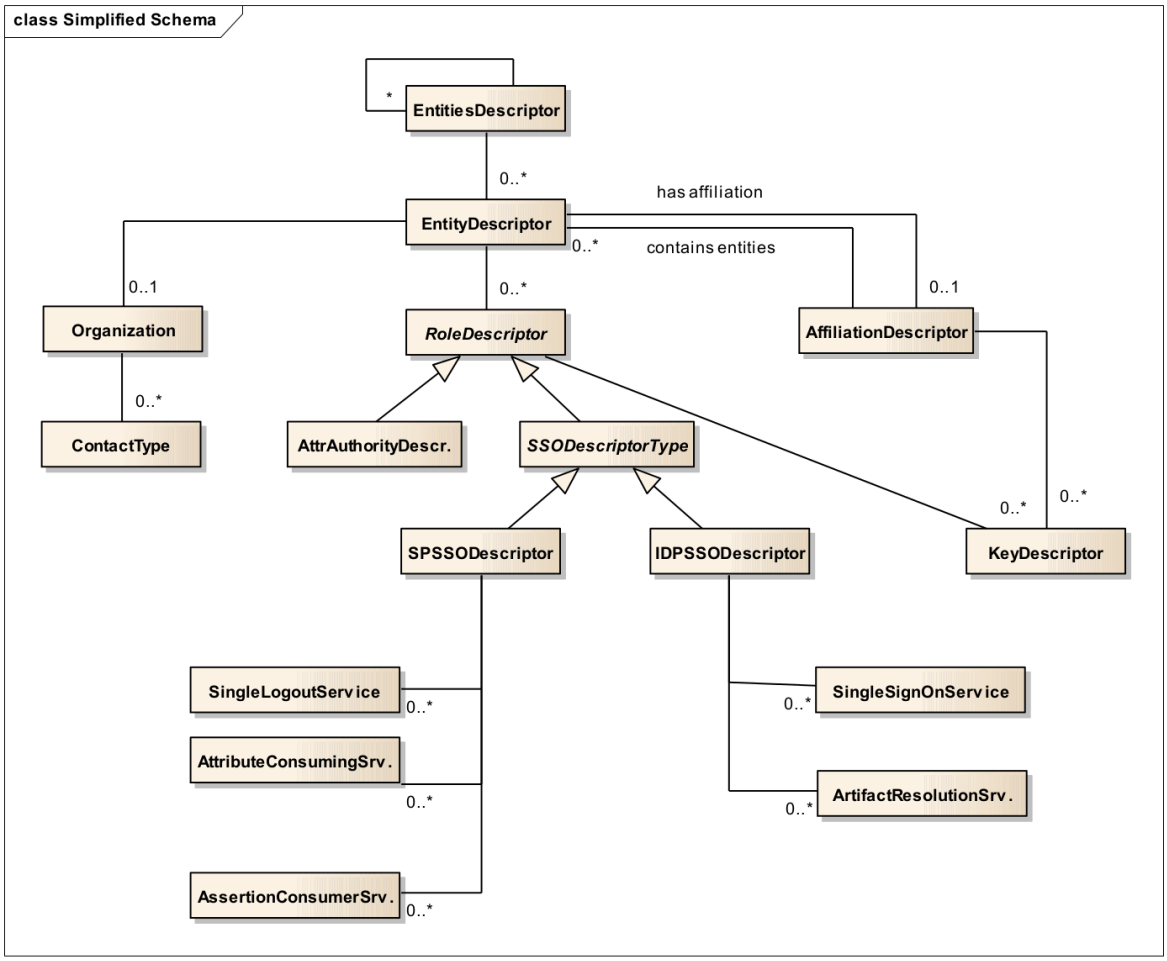

Abbildung 2.2.: Übersicht über wichtige Elemente der SAML Metadaten aus [Hö14]

Der `IDPSSODescriptor` gibt mit seinem `protocolSupportEnumeration`-Attribut an, dass er SAML 1.0, 1.1 und 2.0 unterstützt. Innerhalb des nächsten Elements `<Extensions>` wird aus dem Namespace, der SAML-Implementation Shibboleth `shibmd` das Attribut `Scope` definiert, das für „Scoped“-Attribute verwendet wird.

Innerhalb des nächsten Elements `<KeyDescriptor>` wird durch das Element `<KeyInfo>`, das aus dem von der W3C herausgegebenen Standard zu XML-Signaturen [XMLSig] stammt, ein Zertifikat angegeben. Dieses Zertifikat wird dazu verwendet, die Signaturen der von den Metadaten beschriebenen Installation zu prüfen.

Die nächsten Elemente `<ArtifactResolutionService>` spezifizieren Endpunkte, an denen das „Artifact Resolution Profil“ verwendet werden kann. Dazu wird sowohl ein Binding als auch eine Location angegeben. Der Index wird zum Verwalten mehrerer Elemente gleichen Typs verwendet.

Die Elemente `<NameIDFormat>` geben an, welche `NameID`s das Objekt unterstützt.

Die Elemente `<SingleSignOnService>` zeigen an, welche Endpunkte für die Authentifizie-

rung existieren. Ein IDP muss mindestens ein solches Element spezifizieren.

2.3.3. Aufbau der IDP–SP Vertrauensbeziehung mit SAML

Damit die Nutzer eines IDP einen SP mit ihrem beim IDP gespeicherten Account verwenden können, muss zunächst eine Vertrauensbeziehung zwischen dem IDP und dem SP hergestellt werden. Im einfachsten Fall beinhaltet der Vertrauensaufbau nur den Austausch der Metadaten. In den folgenden zwei Abbildungen 2.3 und 2.4 wird schematisch dargestellt, welche Schritte durchgeführt werden müssen, um auf SAML-Basis eine Vertrauensbeziehung zwischen IDP und SP herzustellen.

Abbildung 2.3 zeigt die Schritte, falls es keine gemeinsame Föderatione gibt. Im ersten Schritt „0“ entdeckt ein Benutzer einen Dienst, den er gerne nutzen möchte und fragt in Schritt 1 bei seinem IDP an, ob es möglich ist eine Authentifizierung bei diesem Dienst über den IDP zu konfigurieren. Wenn IDP und SP prinzipiell bereit sind, sich gegenseitig zu vertrauen, müssen in Schritt 2 und 3 neben organisatorischen und vertraglichen Absprachen die Administratoren des IDPs bzw. SPs manuell die Metadaten austauschen und in ihre Konfiguration einpflegen. Eventuell müssen auch noch, für abweichend benannte oder in unterschiedlichen Formaten gespeicherte Attribute, Formatierungsregeln erstellt und aktiviert werden. Nachdem die Konfiguration abgeschlossen, getestet und für zugelassene Benutzer aktiviert wurde, kann der Benutzer sich in Schritt 4 bei dem SP anmelden

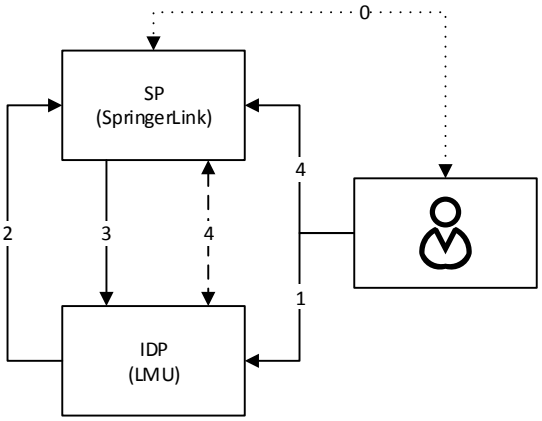

Abbildung 2.3.: Ablauf beim Erstellen von Vertrauensbeziehungen ohne Föderationen.

Diese Methode skaliert sehr schlecht, da diese Prozedur für jeden SP, der an einen IDP angeschlossen werden soll durchgeführt werden muss. Da es vorher keine vertraglichen Absprachen zwischen den Organisationen gibt und das Eintragen und Aktivieren der Metadaten und Regeln manuell durchgeführt wird, kann es bis zu mehrere Wochen dauern, bis ein SP mit einem IDP verwendbar ist. Außerdem wird bei der Vielzahl an geschlossenen Verträgen und Absprachen zwischen den Organisationen, besonders aus Sicht des Benutzers, unübersichtlich.

Abbildung 2.4 zeigt das momentan meistens verwendete Vorgehen. Im ersten Schritt treten SP und IDP derselben Föderation bei. Diese legt Regeln zur Dienstnutzung, Datenschutz und

den Attributsformaten fest und kümmert sich um ein für alle Mitglieder geltendes Vertragswerk. Die aggregierten Metadaten von neu beigetretenen Organisationen können meistens automatisiert in die Systeme der Provider geladen werden, so dass der Administrationsaufwand und die Zeit bis ein neuer Dienst oder IDP verwendet werden kann, deutlich reduziert wird.

Eine Föderation betreibt häufig zusätzliche Dienste wie einen Discovery Service (DS). Dieser Dienst erlaubt es einem Benutzer auf einfachere Weise seine Heimatinstitution als IDP auszuwählen. Unter anderem kann dort auch gespeichert werden, welchen IDP ein Benutzer zuletzt ausgewählt hatte, so dass ein Benutzer nicht für jeden Dienst seinen IDP aus einer langen Liste heraussuchen muss.

Schritt 2 zeigt, wie ein Benutzer auf einen Dienst zugreifen möchte. Da SP und IDP in derselben Föderation sind, kann der Benutzer über den Discovery Service seinen IDP auswählen und wie durch Schritt 3 gekennzeichnet sofort verwenden.

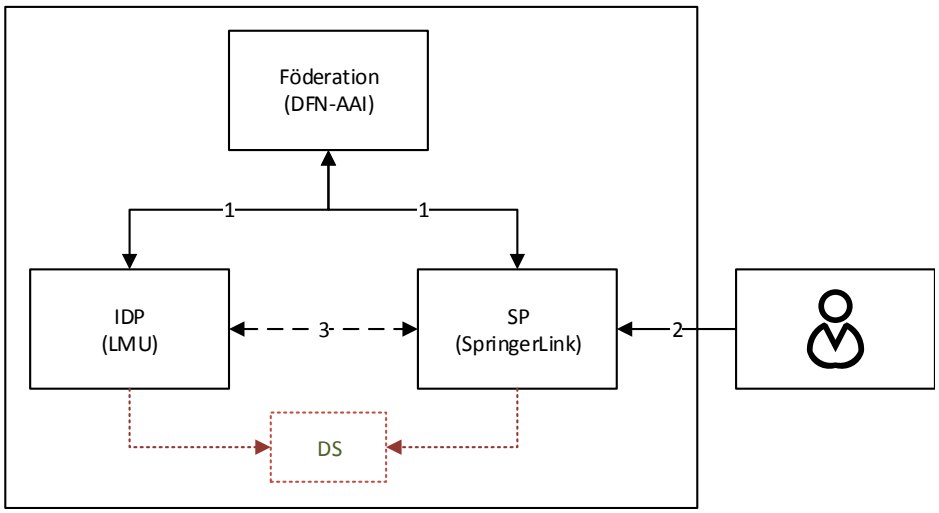

Abbildung 2.4.: Ablauf beim Erstellen von Vertrauensbeziehungen mit Föderationen.

2.4. OpenID Connect

OpenID (`http://openid.net`) ist ein weiterer Standard zum föderierten Identitymanagement der OpenID Foundation. Auch OpenID kennt die Parteien Identity Provider und Service Provider, wobei ersterer OpenID Provider (OP) und letzterer Relying Party (RP) genannt wird. OpenID kann wie SAML primär dazu verwendet werden, eine Identität bei mehreren Diensten zur Authentifizierung zu verwenden.

OpenID findet häufig im Internet Anwendung, wo große Unternehmen wie Google [Goo14a; Goo14b], Microsoft [Mic14] oder Yahoo! [Yah14] als Identity Provider fungieren. Service Provider können verschiedene Webdienste sein, die es ihren Benutzern ermöglichen wollen, ohne einen neuen Account erstellen zu müssen, auf der Webseite aktiv zu sein.

OAuth (`http://oauth.net`), das häufig zusammen mit OpenID erwähnt wird, dient primär

einem anderen Zweck, der Autorisierung. Über OAuth kann ein Benutzer einer Anwendung Zugriff auf bestimmte Funktionalitäten seines Accounts bei einem OAuth Service Provider geben. Einen detaillierten Vergleich von OAuth und OpenID bietet [Mah07].

Zu beachten ist allerdings, dass auf OAuth 2.0 basierend von der OpenID Foundation ein weiterer Standard mit dem Namen OpenID Connect [Ope14] entwickelt wurde. Dieser erlaubt auf Basis von OAuth 2.0 Identity Management zu betreiben. Die eigentliche Spezifikation zu der letzten OpenID Version 2.0 wird daher auf der Internetseite der OpenID Foundation als obsolet geführt [OpenIDSpec]. Daher wird im folgenden Abschnitt der aktuelle Standard OpenID Connect betrachtet.

OpenID Connect definiert auf OAuth 2.0 aufbauend ein Protokoll für die Identifizierung von Benutzern und die Übertragung von grundlegenden Informationen [OpenIDCore]. OpenID Connect 1.0 wird durch die Spezifikation [OpenIDCore], das zugrundeliegende OAuth 2.0 Authorization Framework durch [RFC6749] und OAuth 2.0 Bearer Token Usage in [RFC6750] beschrieben.

Die Spezifikation von OpenID Connect ist, wie in Abbildung 2.5 dargestellt, in sechs Dokumente aufgeteilt. Die grundlegende Funktionalität von OpenID Connect wird in der Core Spezifikation [OpenIDCore] beschrieben, auf die in Abschnitt 2.4.2 näher eingegangen wird. Des weiteren gibt es mit der Discovery Spezifikation [OpenIDDisc] eine zusätzliche Erweiterung für das Finden des OPs eines Benutzers sowie des dynamischen Registrierens bei einem OP durch die Dynamic Registration Spezifikation [OpenIDDyn]. In der Session Spezifikation [OpenIDSession] wird definiert, wie Sitzungen eines Benutzers behandelt werden müssen, inklusive der Beschreibung, wann ein Benutzer abgemeldet wird. Zusätzlich wird OAuth 2.0 für die Verwendung mit OpenID Connect um mehrere Response Types [OAuthRespType] erweitert sowie spezifiziert, wie Responses durch HTTP Form POST Requests [OAuthPostResp] übertragen werden.

Im nächsten Abschnitt wird der generelle Ablauf der Authentifizierung eines Benutzers mittels OpenID Connect dargestellt. In den darauffolgenden Abschnitten werden Kernkomponenten aus der OpenID Connect Core Spezifikation genauer beschrieben.

2.4.1. Genereller Ablauf

In diesem Abschnitt wird der generelle Ablauf der Authentifizierung über OpenID Connect beschrieben. Dieser Ablauf ist in Teilen sehr ähnlich zu dem SAML HTTP Redirect und POST Bindings. Eine detaillierte Beschreibung von OpenID Connect wird zum Beispiel in [Lod14] dargestellt. Im Folgenden wird der Ablauf einer Autorisierung mittels OpenID Connect kurz beschrieben, in den darauf folgenden Abschnitten genauer auf Details des Protokolls eingegangen. Abbildung 2.6 zeigt schematisch den Nachrichtenaustausch.

Wenn sich ein Benutzer bei einem Dienst anmelden möchte, generiert die Relying Party (die auch als Client bezeichnet wird) einen Authorization Request, welcher durch einen HTTP Redirect an den Identity Provider Server des Benutzers geschickt wird. Dieser Request ist ein normaler OAuth request, der einen zusätzlichen Parameter `scope=openid` enthält. Der OAuth-Server weiß dadurch, dass die OpenID-Connect-Erweiterung verwendet werden soll. Zusätzlich kann über weitere Scope-Parameter angegeben werden, welche

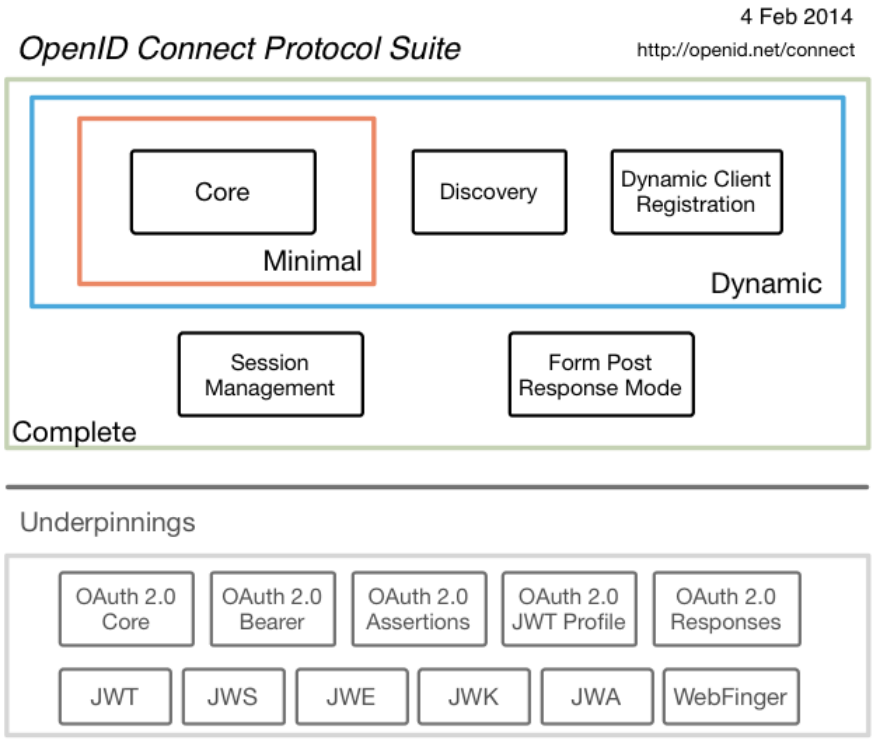

Abbildung 2.5.: Aufbau der OpenID Connect Spezifikation. Abbildung von `http://openid.net/connect`.

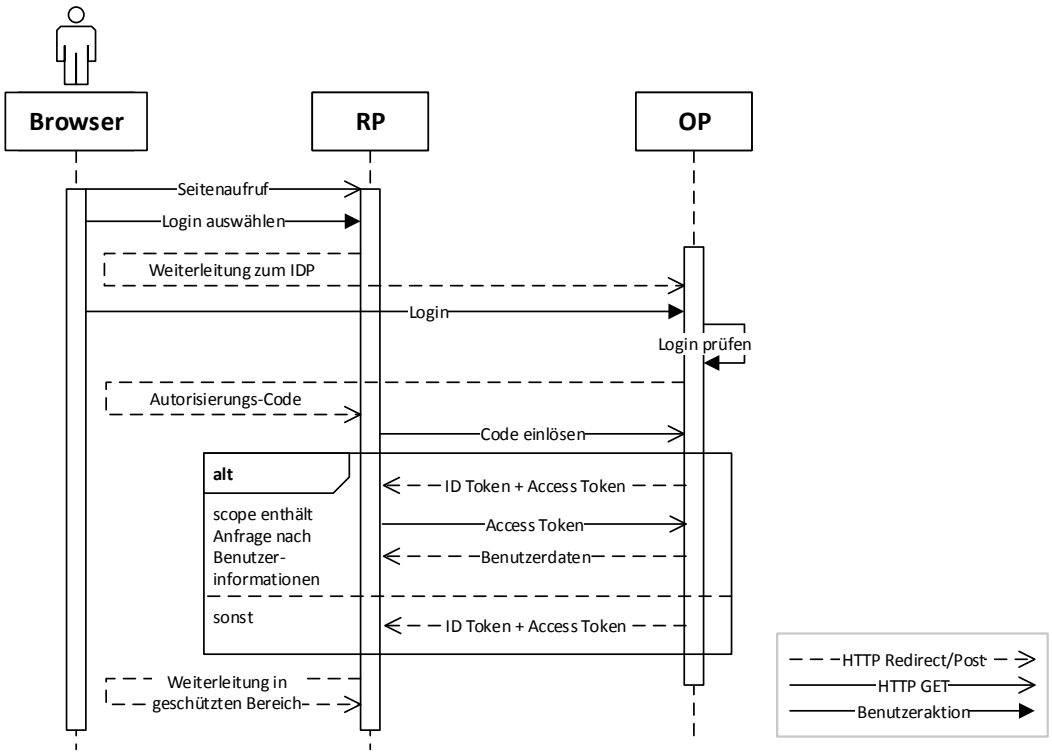

Abbildung 2.6.: OpenID-Connect Autorisierung mit Authorization Code Flow

Listing 2.4: Beispiel für ein ID Token [OpenIDCore]

```
{
  "iss": "https://server.example.com",
  "sub": "24400320",
  "aud": "s6BhdRkqt3",
  "nonce": "n-0S6_WzA2Mj",
  "exp": 1311281970,
  "iat": 1311280970,
  "auth_time": 1311280969,
  "acr": "urn:mace:incommon:iap:silver"
}
```

zusätzlichen Informationen wie Name, E-Mail-Adresse, etc. übertragen werden sollen. Für den Zugriff auf die zusätzlichen beim OP gespeicherten Informationen wird dann ein extra `access_token` übertragen. Der Benutzer muss sich dann bei dem Server – zum Beispiel durch seine Benutzername/Passwort-Kombination – authentifizieren und bestätigen, dass er sich bei dem Dienst, der den Request erstellt hat, anmelden möchte.

Je nach konfiguriertem Ablauf bestätigt der Server dem Client die Authentifizierung durch eine Weiterleitung über den Benutzer entweder direkt, in dem er eine ID Token genannte Struktur überträgt oder indirekt durch einen Autorisierungs-Code. Mit dem Code kann der Client das ID Token, ohne dass es durch den Browser des Benutzers, wo es eventuell entwendet oder modifiziert werden könnte, weitergeleitet werden muss, bei dem Server abfragen. Die Übermittlung eins Autorisierungs-Codes entspricht dem OAuth 2.0 Standard für eine Antwort. Die verschiedenen Abläufe werde in Abschnitt 2.4.2.2 noch einmal detaillierter beschrieben.

2.4.2. OpenID Connect Core

In diesem Abschnitt werden die Kern-Elemente von OpenID Connect beschrieben. Die Beschreibung erfolgt entsprechend der Spezifikation [OpenIDCore], aus der die OpenID Connect betreffenden Informationen stammen.

2.4.2.1. ID Token

Das ID Token ist eine Erweiterung, die OpenID Connect erlaubt, Benutzer über OAuth 2.0 zu authentifizieren. Im Vergleich zu SAML ist es ähnlich zu den im vorhergehenden Abschnitt 2.3.2.1 beschriebenen SAML Assertions. Das ID Token wird allerdings nicht in XML sondern durch JSON, wie in Listing 2.4 dargestellt, repräsentiert und Base64 kodiert übertragen.

Ein ID Token enthält Daten über die Identität und die Authentifizierung des Benutzers aus Sicht des OPs. Bei der Übertragung von OP zu RP muss sichergestellt werden, dass die Integrität, Authentizität und Nicht-Abstreitbarkeit des Tokens erhalten bleibt. Dazu wird das Verfahren der JSON Web Signature [JWS] verwendet. Dabei werden durch die Spezifikation allerdings die Signaturmethoden `x5u`, `x5c`, `jku` oder `jwk` ausgeschlossen. Diese Methoden erlauben es, im Header der Signatur den zum Überprüfen benötigten Public-Key

anzugeben. Stattdessen wird das Zertifikat in dem `Discovery document` des OP, wie bei SAML in den Metadaten des IDP, spezifiziert. Eine zusätzliche optionale Verschlüsselung zur Wahrung der Vertraulichkeit verwendet JSON Web Encryption [JWE].

Im Folgenden werden die möglichen Elemente eines ID Tokens aufgeführt. Ein ID Token muss mindestens die Schlüssel `iss`, `sub`, `aud`, `exp` und `iat` enthalten.

- **iss**: Durch den Schlüssel `iss` wird der Herausgeber (engl. Issuer) durch eine URL identifiziert.
- **sub**: Der Schlüssel `sub` identifiziert den Benutzer (Subject) durch einen eindeutigen String. Es gibt zwei verschiedene Typen für Subject Identifier. Zum einen `public`, bei dem für jeden Dienst der selbe Identifier verwendet wird und zum anderen `pairwise`, bei dem für jeden Dienst ein eigener Identifier generiert wird, um die Verfolgung von Benutzern über mehrere Dienste hinweg zu erschweren.
- **aud**: Mit diesem Schlüssel wird beschrieben, für welche Verbraucher (Audience) das ID Token gedacht ist.
- **exp**: Hier wird das Verfallsdatum (expiration) des Tokens angegeben.
- **iat**: Durch diesen Schlüssel wird angegeben, wann das Token erstellt wurde.
- **auth_time**: Dieser Schlüssel gibt an, wann die Authentifizierung des Benutzers durchgeführt wurde.
- **nonce**: Dieser Schlüssel kann dazu verwendet werden, eine Nonce anzugeben, um Replay-Angriffe zu vermeiden. Die Nonce einer Authentication-Response muss gleich der Nonce in dem Authentication-Request sein.
- **acr**: Über die Authentication Context Class Reference kann angegeben werden, wie die Authentifizierung durchgeführt wurde. Die Einstufung der verschiedenen Level und die Bedeutung dieses Attributs verwendet die in [ISO/IEC29115] definierten Level für die Sicherung der Authentifizierung.
- **amr**: Der Schlüssel `amr` gibt mit der Authentication Method Reference die Authentifizierungsmethode an. Die Bedeutung des Strings ist abhängig vom Kontext und muss zwischen den Parteien definiert werden.
- **azp**: Gibt an, für wen das Token ausgestellt wurde. Dieses Attribut wird nur benötigt, wenn der einzige durch `aud` angegebene Verbraucher nicht gleich dem Empfänger des Tokens ist.

2.4.2.2. Authentifizierungs Abläufe

Die Authentifizierung bei OpenID Connect wird zwischen einem Server (oder OpenID Provider bzw. im SAML Kontext IDP) und dem Client (oder Relying Party bzw. dem SAML SP) für einen Benutzer durchgeführt. Die Authentifizierung erzeugt ein ID Token, wie es im vorherigen Abschnitt beschrieben wurde.

Für die Durchführung der Authentifizierung gibt es drei mögliche Abläufe (Flows). Diese Abläufe sind vergleichbar mit den unterschiedlichen SAML Bindings und basieren alle auf

Listing 2.5: Äquivalent eines OpenID Claim Request zu dem Scope `email` aus [OpenIDCore]

```
{
 "userinfo":
 {
   "email": null,
   "email_verified": null
 }
}
```

dem HTTP-Protokoll:

Authorization Code Flow: Bei diesem Ablauf wird der Benutzer durch den Server authentifiziert und bekommt danach einen Autorisierungs-Code übermittelt. Diesen Code kann der Benutzer dem Client übermitteln, wodurch dieser beim Server die Authentifizierung überprüfen kann.

Implicit Flow: Dieser Ablauf sieht vor, dass das ID Token nach der Authentifizierung durch den Server über den Benutzer an den Client übermittelt wird. Server und Client kommunizieren hier nicht direkt, sondern nur über den Benutzer miteinander. Dies ist vergleichbar zu SAML HTTP Redirect und POST Bindings.

Hybrid Flow: Dieser Ablauf kombiniert die beiden oben genannten dadurch, dass in einem Parameter des Authentication Requests ausgewählt werden kann, ob an den Benutzer ein ID Token oder ein Code übermittelt wird.

2.4.2.3. Claims

Claims sind Aussagen, die der Server über einen Benutzer treffen kann und sind vergleichbar mit SAML Attributen bzw. Assertions. Die OpenID Connect Spezifikation definiert einen Standardsatz an Claims, die getroffen werden können, welcher aber durch eigene Spezifikationen erweitert werden kann. Claims können über zwei verschiedene Methoden in dem Authentication Request angefragt werden. Zum einen, wie in der Übersicht in Abschnitt 2.4.1 erwähnt, über zusätzliche `scope` Werte, zum anderen über ein weiteres Attribut `claims`.

Die Methode über das `scope`-Attribut erlaubt nur eine grobe Angabe der gewünschten Attribute. Es sind hier die Werte `profile`, `email`, `address` und `phone` möglich, die jeweils eine Gruppe von Attributen enthalten. In der Tabelle 2.1, welche alle durch den OpenID Connect Core Standard spezifizierten Attribute auflistet, ist durch die Spalte `scope` angegeben, welche Attribute in welchen Scope fallen. Es können bei einem Request mehrere Werte durch ein Leerzeichen getrennt angegeben werden.

Mit dem extra Attribut `claims` kann spezifischer angegeben werden, welche Benutzerinformationen abgefragt werden sollen. Ein Beispiel für einen sochen Request ist in Listing 2.6 abgebildet. Listing 2.5 zeigt einen OpenID Claim Request, der äquivalent zu der Angabe des Scopes `email` ist. In dem Element `userinfo` werden dabei die einzelnen Claims angegeben,

Listing 2.6: Beispiel für ein Claim Request aus [OpenIDCore]

```
{
 "userinfo":
 {
   "given_name": {"essential": true},
   "nickname": null,
   "email": {"essential": true},
   "email_verified": {"essential": true},
   "picture": null,
   "http://example.info/claims/groups": null
 },
 "id_token":
 {
   "auth_time": {"essential": true},
   "acr": {"values": ["urn:mace:incommon:iap:silver"] }
 }
}
```

die abgefragt werden sollen. Über die Werte der einzelnen Claim Requests können diese genauer spezifiziert werden. Der Wert `null` bedeutet, dass dieser Wert nicht unbedingt benötigt wird und freiwillig angegeben werden kann. Mit dem JSON-Objekt `{"essential": true}` wird der Claim als wichtig für die Authentifizierung und Benutzung des Dienstes markiert. Falls der Benutzer diesen Claim aber nicht freigibt oder der OP ihn nicht erstellen kann, soll allerdings keine Fehlermeldung vom OP erzeugt werden. Mit dem JSON-Object `{"values":["urn:mace:incommon:iap:silver"] }` wird angefragt, dass ein Claim mit einem der angegebenen Werte zurückgegeben werden soll. Über das Element `id_token` wird angegeben, dass die Antworten in das ID Token integriert werden sollen.

Die folgende Tabelle 2.1 enthält die von OpenID Connect definierten Claims. Zu erkennen ist dabei, dass diese Informationen es ermöglichen, eine Person sehr genau zu beschreiben. Es gibt sechs Claims, die den Namen eines Benutzers beschreiben. Wobei die Definition eines Claims für Spitzname darauf schließen lässt, dass der Einsatz dieses Protokolls eher für den privaten Bereich gedacht ist. Dies ermöglicht im Vergleich zu den bei SAML verwendeten Attributs-Schematas eine sehr genaue Beschreibung einer Person. Da SAML selbst keine Attribute spezifiziert, werden die hier verwendeten Schemata erst in Abschnitt 3.2.2.4 beschrieben. Üblicherweise enthalten diese Schemata allerdings nur jeweils ein Attribut für Vor- und Nachnamen und eventuell einen Anzeigenahmen.

2.4.3. OpenID Connect Discovery

In diesem Abschnitt wird der in der Spezifikation [OpenIDDisc] beschriebene Ablauf, wie eine RP den zu einem Benutzer gehörenden OP finden kann, vorgestellt. Dazu wird in der Spezifikation das WebFinger [RFC7033] Protokoll verwendet. Alternativ kann auch „out-of-band“ eine andere Methode verwendet werden. Im Folgenden wird die Methode mittels WebFinger beschrieben.

Attributsname	Beschreibung	Scope
sub	Serverspezifische Identifikation	
name	Vollständiger Name des Benutzers, inklusive Titel und Zusätzen	`profile`
given_name	Vorname(n)	`profile`
family_name	Nachname(n)	`profile`
middle_name	Zwischenname(n)	`profile`
nickname	Spitzname	`profile`
preferred_username	Name unter dem der Benutzer auftreten möchte	`profile`
profile	URL zu einer Profilseite	`profile`
picture	URL zu einem Bild	`profile`
website	URL zu einer persöhnlichen Webseite oder Blog	`profile`
gender	Geschlecht	`profile`
birthdate	Geburtsdatum der Form `YYYY-MM-DD` nach ISO 8601:2004	`profile`
zoneinfo	Informationen zur Zeitzone	`profile`
locale	Vom Benutzer bevorzugte Sprache	`profile`
updated_at	Datum der letzten Aktualisierung der Daten	`profile`
address	Postadresse	`address`
email	E-Mailadresse	`email`
email_verified	Indikator ob die E-Mailadresse verifiziert wurde	`email`
phone_number	Telefonnummer	`phone`
phone_number_verified	Indikator ob die Telefonnummer überprüft wurde	`phone`

Tabelle 2.1.: OpenID Connect standard Claims nach [OpenIDCore]

2.4.3.1. WebFinger

Für die Suche nach einem OP mit WebFinger benötigt die RP vom Benutzer einen Identifier, der `resource` genannt wird. Aus diesem Identifier wird mittels, durch OpenID Connect, standardisierter Normalisierungsregeln der Host, auf dem der WebFinger Service betrieben wird, ermittelt. Mögliche Formen eines Identifiers haben zum Beispiel die Struktur einer E-Mail-Adresse `acct:benutzer@host` oder einer URL `https://host/user`.

Mit dem aus dem Identifier extrahierten Host kann nun ein WebFinger Request erstellt werden. Ein Beispiel für einen solchen Request mit dem Identifier `acct:joe@example.com` ist in Listing 2.7 abgebildet. WebFinger verwendet für die HTTP GET Anfrage an den Host den Pfad `/.well-known/webfinger` und die Parameter `ressource`, welcher den Identifier des Benutzers enthält, und `rel`, welcher durch den für OpenID Connect festgelegten String `http://openid.net/specs/connect/1.0/issuer` die Art des gesuchten Dienstes angibt. Die Antwort des WebFinger Servers enthält eine Liste von Links, in der ein Verweis `href`, den Ort des von dem Benutzer verwendeten OP enthält.

Die Verwendung von WebFinger bietet dem Benutzer den Vorteil auf einfache Art, einen eigenen WebFinger Server zu betreiben, wodurch er bei Bedarf, ohne seinen Identifier zu ändern, den OP wechseln kann. Durch die einfache Struktur von WebFinger ist für den Betrieb ein einfacher Webserver/Webspace ausreichend.

2.4.3.2. Server Metadaten

In diesem Abschnitt werden die Metadaten des OpenID Providers näher beschrieben, die Metadaten der Relying Party werden später in Abschnitt 2.4.3.3 vorgestellt.

Listing 2.7: Beispiel für einen WebFinger Request aus [OpenIDDisc]

```
GET /.well-known/webfinger
  ?resource=acct%3Ajoe%40example.com
  &rel=http%3A%2F%2Fopenid.net%2Fspecs%2Fconnect%2F1.0%2Fissuer
  HTTP/1.1
 Host: example.com

 HTTP/1.1 200 OK
 Content-Type: application/jrd+json

 {
  "subject": "acct:joe@example.com",
  "links":
  [
   {
   "rel": "http://openid.net/specs/connect/1.0/issuer",
   "href": "https://server.example.com"
   }
  ]
 }
```

Wenn die RP den OP eines Benutzers bestimmt hat, kann sie die Metadaten des OP abfragen. Für diese Abfage wird wieder WebFinger verwendet. Ein OP veröffentlicht seine Metadaten dazu unter dem Pfad `/.well-known/openid-configuration`. Ähnlich wie bei einem SAML IDP enthalten die Metadaten des OP URLs für die verschiedenen Kommunikationsendpunkte sowie der öffentlichen Schlüssel, mit dem die RP die Signatur des OP überprüfen kann. Listing 2.8 zeigt als Beispiel die Metadaten des Google OpenID Connect Providers, anhand derer die einzelnen Elemente vorgestellt werden.

Das Attribut `issuer` identifiziert den OP, der Wert dieses Attributs muss der gleiche sein, wie der von diesem OP herausgegebenen ID Tokens. Googles OP folgt hier nicht dem Standard, der für das Attribut das `https` Schema vorschreibt, nach dem der `issuer` also als `https://accounts.google.com` angegeben werden müsste. Das hat zur Folge, dass in den verschiedenen Implementierungen zu OpenID Connect Spezialfälle eingebaut werden müssen, die bei Bedarf das „https“ergänzen. Das `issuer`-Attribut entspricht bei SAML der `entityID`.

Die Attribute `authorization_endpoint`, `token_endpoint`, `userinfo_endpoint` spezifizieren Kommunikationsendpunkte für OAuth und OpenID Connect. Ein `revocation_endpoint`, wie er auch in den von Google veröffentlichten Metadaten zu finden ist, ist in der OpenID Connect Discovery Spezifikation nicht definiert und wird daher in einigen Implementierungen nicht verwendet.

Der Wert des Attributes `jwks_uri` enthält eine URL, unter der sich der JSON Web Key Store befindet, der die öffentlichen Schlüssel zur Überprüfung von Signaturen des OPs enthält.

Die anderen Attribute geben an, welche Methoden von dem OP unterstützt werden.

Listing 2.8: OpenID Connect Metadaten von Google `https://accounts.google.com/.well-known/openid-configuration`

```
{
 "issuer": "accounts.google.com",
 "authorization_endpoint": "https://accounts.google.com/o/oauth2/auth",
 "token_endpoint": "https://accounts.google.com/o/oauth2/token",
 "userinfo_endpoint": "https://www.googleapis.com/plus/v1/people/me/openIdConnect",
 "revocation_endpoint": "https://accounts.google.com/o/oauth2/revoke",
 "jwks_uri": "https://www.googleapis.com/oauth2/v2/certs",
 "response_types_supported": [
  "code",
  "token",
  "id_token",
  "code token",
  "code id_token",
  "token id_token",
  "code token id_token",
  "none"
 ],
 "subject_types_supported": [
  "public"
 ],
 "id_token_alg_values_supported": [
  "RS256"
 ],
 "token_endpoint_auth_methods_supported": [
  "client_secret_post"
 ]
}
```

2.4.3.3. Client Metadaten

Die Metadaten eines Clients werden für die Registration bei einem Server benötigt und enthalten Informationen zu dem Dienst wie zum Beispiel den Namen, Kontaktadressen und AGBs sowie sicherheitsrelevante Angaben. Ein Beispiel für Client Metadaten in einem Registration Request ist in Listing 2.9 abgebildet, welches im nächsten Abschnitt zu finden ist.

Das Attribut `application_type` gibt an was für ein Dienst die RP anbietet. Es gibt hier die möglichen Werte `web` für Webanwendungen oder `native` für andere Applikationen.

Über die Angaben `client_name` und `logo_uri` kann der Dienst einen Namen und Logo angeben, welche der OP auf der Loginseite verwenden kann, um dem Benutzer zu beschreiben, bei welchem Dienst er sich gerade anmeldet. Weitere mögliche Attribute aus dieser Kategorie sind `tos_uri` für Allgemeine Geschäftsbedingungen (AGB), `policy_uri` für Datenschutzbestimmungen und `client_uri` für die Homepage des Dienstes. Über `contacts` können zusätzlich Ansprechpartner des Clients angegeben werden.

Laut der Spezifikation gibt es nur ein verpflichtendes Attribut in den Client Metadaten, die `redirect_uris`. Mit dieser Liste gibt der Client an, welche `redirect_uri` Parameter bei einem Authorization Request gültig sind. Der OP leitet den Browser nach der Authentifizierung an die unter `redirect_uri` angegebene URL weiter.

Mit dem Attribut `subject_type` wird angefordert, welche Art von Identifier für einen Benutzer generiert werden soll. Mögliche Werte sind `public` und `pairwise`.

Eine alternative Möglichkeit zum Attribut `redirect_uris` ist `sector_identifier_uri`, mit dem eine URL zu einem JSON-Array spezifiziert werden kann, in dem gültige `redirect_uris` angegeben werden. Für alle in dem Array definierten URIs wird für einen Benutzer der selbe Identifier generiert.

Über das Attribut `token_endpoint_auth_method` wird eine Methode, für die Authentifizierung des Clients am Token Endpoint angefragt. Das hier angegebene `client_secret_basic` ist die Standardeinstellung, bei der durch die Authentication Response übertragene Autorisierungs Code (Secret) mit der HTTP Basic Authentication Methode übertragen wird.

Auch der Client kann seine Requests signieren, die zur Überprüfung der Signatur notwendigen öffentlichen Schlüssel werden an der durch `jwks_uri` angegebenen URI in einem JSON Web Key Set hinterlegt.

Die beiden Attribute `userinfo_encrypted_response_{alg,enc}` geben den Algorithmus für die Verschlüsselung des Content Encryption Keys (CEK) und den Algorithmus, der mit dem CEK für die Verschlüsselung von UserInfo Responses verwendet wird, an.

Die unter `request_uris` angegebenen URIs können vom OP gecached werden. Für den Fall einer Änderung enthalten die URIs einen Base64 Encodeten SHA-256 Hash des Inhalts.

2.4.4. Aufbau der RP – OP Vertrauensbeziehung

OpenID Connect kann durch eine Großzahl an Implementierungen sehr flexibel in unterschiedlichsten Umgebungen wie zum Beispiel Webseiten und Mobil-Apps verwendet werden. Eine Liste von OpenID Connect Implementierungen findet sich unter `http://openid.net/developers/libraries`. Darunter zum Beispiel ein Apache Webserver Modul, mit dem eine RP eine Webseite absichern kann. Als Identity Provider kann entweder ein eigener Server betrieben oder ein Dienst verwendet werden, der als OpenID Provider auftritt. Zum Beispiel betreibt Google einen OP, den Entwickler verwenden können, um Benutzer zu authentifizieren. Diese Methode hat den Vorteil, dass der Benutzer keinen neuen Account anlegen muss, sondern einen bekannten Account weiterverwenden kann.

Wie im vorhergehenden Abschnitt 2.4.3.2 beschrieben, können die Metadaten eines OP über WebFinger abgerufen werden. Der OP benötigt allerdings auch die Metadaten der RP, dazu muss die RP sich bei dem OP registrieren. Die Methode der dynamischen Registrierung ist in der OpenID Spezifikation [OpenIDDyn] beschrieben und wird im Folgenden vorgestellt.

Bei der Registrierung schickt der Client dem Server seine Metadaten über einen HTTP-POST-Request, wie er in Listing 2.9 dargestellt ist. Der Server antwortet bei erfolgreicher Registrierung mit eventuell an die Serverkonfiguration angepassten Metadaten.

Für die Registrierung eines Clients bei einem Server ist eventuell ein Access Token notwendig, dass der Betreiber des Clients „out-of-band“ vom Betreiber des Servers beziehen kann. Falls so ein Token nicht benötigt wird, muss der Serverbetreiber andere Maßnahmen wie Rate-Limiting einsetzen, um denial-of-service (DoS) Angriffe auf den Registration Endpoint zu verhindern. Zum Beispiel wird für die Registrierung einer RP mit dem Google OP, wie

Listing 2.9: OpenID Connect Registration Request mit RP Metadaten aus [OpenIDDyn]

```
POST /connect/register HTTP/1.1
Content-Type: application/json
Accept: application/json
Host: server.example.com
Authorization: Bearer eyJhbGciOiJSUzI1NiJ9.eyJ ...

{
 "application_type": "web",
 "redirect_uris":
  ["https://client.example.org/callback",
  "https://client.example.org/callback2"],
 "client_name": "My Example",
 "logo_uri": "https://client.example.org/logo.png",
 "subject_type": "pairwise",
 "sector_identifier_uri":
  "https://other.example.net/file_of_redirect_uris.json",
 "token_endpoint_auth_method": "client_secret_basic",
 "jwks_uri": "https://client.example.org/my_public_keys.jwks",
 "userinfo_encrypted_response_alg": "RSA1_5",
 "userinfo_encrypted_response_enc": "A128CBC-HS256",
 "contacts": ["ve7jtb@example.org", "mary@example.org"],
 "request_uris":
  ["https://client.example.org/rf.txt
   #qpXaRLh_n93TTR9F252ValdatUQvQiJi5BDub2BeznA"]
}
```

es unter [Goo14b] beschrieben ist, ein vorher durch ein Webfrontend angelegter Schlüssel vereinbart.

2.5. Vergleich von SAML und OpenID Connect

In diesem Kapitel werden die Unterschiede zwischen den zwei populärsten FIM Technologien SAML 2.0 und OpenID Connect zusammengefasst. Generell unterscheiden sich SAML und OpenID Connect darin, dass SAML ein abstraktes Framework für die Authentifizierung definiert und OpenID Connect konkreter ein Authentifizierungsprotokoll beschreibt. In dieser Betrachtung wird zunächst auf die technischen und dann auf die organisatorischen Unterschiede eingegangen.

2.5.1. Vergleich der Technik

Aus technischer Sicht unterscheiden sich SAML und OpenID Connect in den verwendeten Technologien. SAML basiert stark auf XML und unterstützt dazu verschiedene Transport-Protokolle wie SOAP oder HTTP. OpenID Connect verwendet anstelle von XML, JSON und spezifiziert als Transport-Protokoll HTTP. Der Vorteil von JSON und HTTP ist, dass diese Webtechnologie im Umfeld von Browsern direkt unterstützt wird, SAML muss für die Übertragung von XML-Dokumenten mittels HTTP zusätzlich ein Base64-Encoding und für die Übertragung in einem URL-Parameter Kompression verwenden.

Die Verwendung von XML bei SAML erhöht hier allerdings die Strukturierbarkeit der Dokumente und erlaubt es, Erweiterungen zu spezifizieren, deren Elemente durch einen eigenen

XML-Namespace von anderen Erweiterungen zu unterscheiden sind. OpenID Connect ist nur rudimentär, durch das Hinzufügen von weiteren Key-Value-Paaren, erweiterbar. Da JSON keine Namespaces hat, müssen die Keys mit Bedacht gewählt werden, um Kollisionen zu vermeiden.

Sowohl für SAML als auch für OpenID Connect gibt es allerdings Open Source Implementierungen, die bei Bedarf für eine Erweiterung angepasst werden können.

Da sowohl SAML als auch OpenID Connect ein Framework spezifizieren, muss sichergestellt werden, dass die verschiedenen Implementierungen miteinander kompatibel sind. SAML spezifiziert dazu in [SAMLConf] Kriterien, die festlegen, welche Profile von einem IDP oder SP implementiert werden müssen. Für OpenID Connect gibt es Tests, mit denen die Funktionalität einer Implementierung überprüft werden kann [IDCInter]. UNINETT stellt sowohl für SAML als auch OpenID Connect unter `http://openidtest.uninett.no/` Testumgebungen zur Verfügung.

Für den Aufbau einer Vertrauensbeziehung ist in SAML der Austausch von Metadaten notwendig, der nicht durch die SAML Spezifikation beschrieben wird, sondern auf anderem Wege, eventuell auch mit der zusätzlichen Unterzeichnung eines Vertrages, zwischen SP und IDP stattfinden muss. OpenID verfolgt ein Konzept, jedem OP zu trauen und gibt ein Protokoll an, nach dem sich vorher unbekannte RPs und OPs bei Bedarf verbinden.

Einen wichtigen Unterschied gibt es zwischen SAML und OpenID Connect vor allem für den Benutzer bei der Auswahl seines IDPs bei einem SP. In SAML müssen sich SP und IDP bekannt sein (eventuell durch eine Föderation). OpenID Connect unterstützt die Verwendung eines beliebigen OP, den die RP über ein Discovery Protokoll aus einem durch den Benutzer angegebenen Identifier findet und sich eventuell auch automatisiert dort registrieren kann. Dieser Unterschied hat auch organisatorische Auswirkungen, die im nächsten Abschnitt betrachtet werden.

Ein Benutzer von OpenID Connect kann mit einem kleinen Webserver selbst einen WebFinger Server betreiben und so mit ein und dem selben Identifier eventuell den OP wechseln. Bei SAML ist der Benutzer an einen IDP gebunden.

2.5.2. Vergleich der organisatorischen Aspekte

Aus organisatorischer Sicht unterscheiden sich SAML und OpenID Connect vor allem durch die jeweils angesprochen Zielgruppe. SAML bietet hohe Sicherheit und feste Beziehungen zwischen IDP und SP wodurch es eher für den Einsatz im „enterprise“-Umfeld geeignet ist. OpenID Connect wird eher in dem „consumer“ Bereich eingesetzt. Das zeigt sich auch dadurch, dass jeder durch Anbieter wie Google einen OP für die Verwendung mit OpenID Connect bekommen kann und jeder eine RP für die Verwendung mit Googles oder anderen OPs einrichten kann.

OpenID Connect ermöglicht es durch ein Discovery Protokoll auf einfache Art und Weise, den OP eines Benutzers zu finden und auch die Möglichkeit, die RP bei dem OP dynamisch zu registrieren. Ein Benutzer kann durch einen eigenen WebFinger Server auch den OP wechseln ohne einen anderen Identifier verwenden zu müssen. Für das Finden eines IDP wird bei SAML ein Discovery Service verwendet, der dem Benutzer aber nur die Wahl schon

bekannter vertrauenswürdiger IDPs lässt. Außerdem ist der Identifier (bzw. Benutzername) bei SAML immer an den IDP bei dem der Benutzer einen Account hat gebunden.

Dadurch entstehen auch Unterschiede in der Verlässlichkeit der von einem IDP/OP gelieferten Informationen. Während ein SP innerhalb einer Föderation meist weiß, nach welchen Regeln ein IDP Benutzerdaten speichert und aktualisiert, ist es in dem offenen System von OpenID Connect kaum möglich festzustellen, wie aktuell die gemeldeten Daten sind. Dies ist natürlich nur im Allgemeinen gültig, auch mit OpenID Connect können RPs nur bestimmte OPs zulassen und über Verträge mit diesen regeln, wie die Benutzerdaten erhoben und aktualisiert werden müssen. Auch ist es nicht in jedem Fall wichtig, den genauen Namen/Anschrift eines Benutzers zu kennen, sondern dem Benutzer eine einfache Möglichkeit zu bieten einen Dienst überhaupt zu verwenden.

3. Anforderungen und Anwendbarkeit dynamischer virtueller Föderationen

In diesem Kapitel wird zunächst das Konzept dynamischer virtueller Föderationen vorgestellt. Dazu werden im ersten Abschnitt dynamische virtuelle Föderationen definiert und dann für diese Anforderungen aufgestellt. In dem darauf folgenden Abschnitt 3.2 wird überprüft, ob dieses Konzept und dessen Anforderungen mit bereits existierenden FIM-Konzepten

wie Community- und Projekt-Föderationen (Abschnitt 3.2.1) sowie nationalen und Inter-Föderationen (Abschnitt 3.2.2) kompatibel sind. Eventuell müssen dabei Anforderungen angepasst oder der Anforderungskatalog ergänzt werden. Abschließend werden die gesammelten Anforderungen in Abschnitt 3.3 zusammengefasst.

Bei der Erstellung der Anforderungen wird zwischen „funktionalen“ und „nichtfunktionalen“ Anforderungen unterschieden. Funktionale Anforderungen werden im Folgenden mit dem Präfix „F“ und nichtfunktionale mit dem Präfix „NF“ nummeriert. Zusätzlich werden die Anforderungen nach ihrer Relevanz für die Implementierung eines Systemes für dynamische virtuelle Föderationen gewichtet. Dabei entsprechen „grundlegende“ Anforderungen sehr wichtigen Anforderungen, die umgesetzt werden müssen, da ohne sie das System nicht funktionieren würde. „Erweiterte“ Anforderungen enthalten Zusatzfunktionen, die das System umsetzen sollte, weil sie die Bedienung und Einsetzbarkeit des Systems maßgeblich erleichtern. Die restlichen Anforderungen, die Funktionen beschreiben, die nicht sehr wichtig und nur „schön zu haben“ wären, werden als „optional“ bezeichnet.

3.1. Anforderungen dynamischer virtueller Föderationen

Dynamische virtuelle Föderationen sind ein Konzept, welches es einfacher macht, föderiertes Identitymanagement für eine Community oder ein Projekt zu implementieren. Der Name virtuelle Föderation (VF) orientiert sich an der Definition virtueller Organisationen aus dem Grid-Computing. Dort wird eine virtuelle Organisation (VO) beschrieben als „[...] eine Menge von Individuen, Ressourcen und/oder Institutionen, die sich durch Regeln (Policies) definiert, die das Teilen der Ressourcen festlegen“ [Rei08]. Der Zusatz „dynamisch“ bedeutet, dass diese virtuellen Föderationen automatisiert erstellt werden können und eventuell sehr kurzlebig sind.

Communities und Projekte, die FIM benötigen, haben, wie im vorhergehenden Kapitel beschrieben, besondere Anforderungen an die Flexibilität und Geschwindigkeit, mit der Änderungen umgesetzt werden können. Daher soll die von ihnen gegründete Föderation sehr dynamisch sein. Neben der Dynamik zeichnen sich die Föderationen dadurch aus, dass sie häufig rein virtuell, d.h. ohne reale Entsprechung und nur für die Zeit, in der sie benötigt werden, existieren. Das hat den Vorteil, dass nicht erst Verträge ausgehandelt und von allen Teilnehmern unterschrieben werden müssen. Stattdessen akzeptieren die Organisationen einen Verhaltenskodex, der die Zusammenarbeit regelt.

Üblicherweise bestehen Communities und Projekte aus einer begrenzten Zahl an Teilnehmern. Daher sind auch dynamische virtuelle Föderationen auf wenige (<10) teilnehmende Organisationen ausgelegt und verwenden die selben Systeme für die Teilnahme in mehreren dyn. virt. Föderationen. Dadurch wird es für Service Provider möglich, mit ihrem SP in verschiedenen Föderationen vertreten zu sein und IDPs können ihren Benutzern ebenso den Zugang zu diversen SPs gewähren.

Zum Aufbau und der Verwaltung von dynamischen virtuellen Föderationen gibt es verschiedene mögliche Ansätze. Zum einen kann ein Ansatz wie bei OpenID Connect verfolgt werden, bei dessen Konzept, wie in Abschnitt 2.4 beschrieben, idealerweise jede RP mit jedem OP eine Vertrauensbeziehung aushandeln kann. Dieser Ansatz hat allerdings einige Nachteile,

weshalb in der Praxis der vollständig automatische Verbindungsaufbau meistens nicht umgesetzt ist. Sowohl IDP und SP möchten in fast allen Fällen etwas Kontrolle darüber haben, aus welcher Quelle die Daten stammen, mit denen sich ein Benutzer authentifiziert, bzw. welchen Diensten die Benutzerinformationen übermittelt werden. Daraus ergibt sich zum Beispiel ein System wie bei Google, die als OP auftreten, aber verlangen, dass alle RPs, bei denen sich die Google-Benutzer anmelden können sollen, vorher registrieren.

Zum anderen kann ein Ansatz wie für (Inter-)Föderationen, die SAML verwenden, betrachtet werden. Eine zentrale Instanz ist dort dafür zuständig, die Teilnehmer zu registrieren und deren Metadaten zu verwalten und an alle anderen Teilnehmer zu veröffentlichen. Letztendlich haben beide „erfolgreichen“ Systeme eine zentrale Instanz. Im Fall von Google wird, da sich jeder mit einem Google-Account als RP registrieren kann, weniger streng reguliert, wer als RP auftreten kann. Bei (Inter-)Föderationen ist für die Aufnahme mindestens ein nachgewiesen echter Ansprechpartner und eine Organisation notwendig. Föderationen erlauben zusätzlich SPs und IDPs die Teilnahme, so dass ein echtes Netzwerk an Beziehungen zwischen SPs und IDPs innerhalb einer Föderation existieren kann. Bei Google und anderen großen RPs ist der Netzwerkgraph immer sternförmig, ausgehend von dem OP des Betreibers.

Ein System für dynamische virtuelle Föderationen wird aufgrund dieser Erfahrungen und Beobachtungen ebenfalls eine zentrale Instanz benötigen, die ein gewisses Maß an Kontrolle und Filterung ermöglicht. Zudem ermöglicht es eine diese Instanz festzustellen, wer ebenfalls in der Föderation vertreten ist und erleichtert die Suche nach potentiellen Projektpartner deutlich.

Um das Ziel der dynamischen virtuellen Föderationen, schnell und flexibel Föderationen aufbauen zu können, umzusetzen, werden Methoden benötigt, die automatisiert die Metadaten der beteiligten Systeme austauschen und eventuell nicht kompatible Attributsschemata konvertieren. Eine ausführliche Beschreibung der Anforderungen an dyn. virt. Föderationen wird in den folgenden Abschnitten erstellt.

Daraus ergeben sich einige Anforderungen an ein FIM-System, die im Folgenden dargestellt werden. Zunächst werden im nächsten Abschnitt dazu allgemeine Anforderungen betrachtet. Die zwei, in Abschnitt 1.1 gezeigten, in Föderationen allgemein auftretenden Teilprobleme der Metadatensynchronisation und Attributsformate werden darauf in den Abschnitten 3.1.2 und 3.1.3 beschrieben. Zusätzlich werden in Abschnitt 3.1.4 spezifische Anforderungen aus der Sicht einzelner Teilnehmer einer Föderation wie IDPs, SPs und Benutzer betrachtet.

3.1.1. Allgemeine Anforderungen

Zunächst werden in diesem Abschnitt funktionale (Abschnitt 3.1.1.1) und nichtfunktionale (Abschnitt 3.1.1.2) Anforderungen aufgestellt, die allgemein aus dem Konzept der dynamischen virtuellen Föderationen abgeleitet werden können.

3.1.1.1. Funktionale Anforderungen

Aus der Beschreibung dynamischer virtueller Föderationen ergibt sich, dass es nicht praktikabel ist, für jede dynamische virtuelle Föderation, die eine Organisation aufbauen möchte, manuelle Konfigurationen durchzuführen. Die Konfiguration der beteiligten Systeme soll daher automatisch stattfinden können. Für die Performance ist es vor allem entscheidend, dass im Regelfall keine Schritte manuell durch Administratoren durchzuführen sind und der Ablauf vollständig automatisch durchgeführt werden kann. Zusätzliche Latenzen durch persönliche Absprachen zwischen Administratoren und das manuelle Austauschen von zum Einrichten benötigter Informationen soll nicht mehr nötig sein. Die Automatisierung erleichtert es auch, dass die Vertrauensbeziehungen, die für eine Föderation nötig sind, erst bei Bedarf aufgebaut werden kann und das System so flexibel bleibt. Diese Anforderung ist grundlegend um die Dynamik von Communities und Projekten in der FIM-Infrastrukur abzubilden.

> **F1: Dynamischer Verbindungsaufbau**
>
> Das Hinzufügen und Entfernen von Organisationen in dynamischen virtuellen Föderationen soll nur bei Bedarf und automatisiert innerhalb weniger Minuten erfolgen.

Für die Verwaltung dynamischer virtueller Föderationen wird, wie zuvor beschrieben, ein zentrales System benötigt, welches bei Bedarf für die Einrichtung der Föderation eine Vermittlerrolle einnimmt. Dadurch muss eine Organisation sich nur einmal bei dieser vertrauenswürdigen dritten Partei (engl. Trusted Third Party (TTP)) registrieren und kann dann mit allen anderen registrierten Teilnehmern, durch deren Vermittlung, dynamische virtuelle Föderationen erstellen. Diese Anforderung ist grundlegend für das System, da es den Konfigurationsaufwand für die teilnehmenden Organisationen minimiert und gewährleisten kann, dass der Aufbau der Föderation nicht von der Verfügbarkeit einzelner Administratoren abhängt.

> **F2: Trusted Third Party**
>
> Um ohne viel Konfigurationsaufwand beliebig dynamische virtuelle Föderationen aufzubauen, soll eine Trusted Third Party als zentrale Vermittlungsinstanz verwendet werden.

Durch den Betrieb einer TTP fallen weitere Anforderungen an. Es wird eine Möglichkeit benötigt, mit der sich neue Benutzer an dem System registrieren und danach entsprechend authentifizieren können. Mit der Registrierung verbunden ist auch die Möglichkeit, einen Account nach der Registrierung modifizieren zu können und eventuell auch wieder aus dem System löschen zu können. Diese Anforderung entspricht einer üblichen Benutzerverwaltung, die grundlegend ist, um einen solchen Dienst zu betreiben.

> **F3: TTP-Anmeldung**
>
> Das System benötigt die Möglichkeit zur Registrierung, Modifizierung und Löschung von Accounts, sowie der Authentifizierung mit gültigen Accounts an der TTP.

Neben der reinen Anmeldung muss zum einen sichergestellt werden können, dass ein Benutzer wirklich für den Provider, den er anmeldet, zuständig ist. Diese Anforderung ist grundlegend, um einen solchen Dienst zu betreiben und zu vermeiden, dass durch falsche Registrierungen Dienste imitiert werden, wodurch Benutzer eventuell getäuscht werden könnten und daher Daten zu einem nicht vertrauenswürdigen Dienst übermitteln würden.

F4: Zuständigkeit

Es muss durch eine geeignete Methode festgestellt werden können, wer für einen Dienst verantwortlich bzw. berechtigt ist, den Dienst in das System aufzunehmen und Änderungen vorzunehmen.

Zum anderen muss durch ein System von Berechtigungen sicher gestellt werden, das nur von dem für ein System zuständigen Verantwortlichen dazu berechtige Benutzer Änderungen an den bei der TTP hinterlegten Eigenschaften eines Systems durchführen können. Zum Beispiel muss gerade bei größere Organisationen neben dem primären Verantwortlichen auch eine Vertretung in der Lage sein, Änderungen vorzunehmen. Diese Anforderung ist grundlegend, um einen solchen Dienst effektiv verwenden zu können.

F5: Zugriffsberechtigungen

Das System benötigt die Möglichkeit, Lese- und Schreib-Berechtigungen für die registrierten Systeme zu vergeben, so dass nur autorisierte Personen die Eigenschaften eines Dienstes verändern können.

Nach der erfolgreichen Anmeldung, muss der Benutzer in der Lage sein, den oder die ihm zugeordneten Dienste zu verwalten. Die Verwaltung umfasst dabei sowohl die Änderung von Eigenschaften und Einstellungen, die bei der TTP über den Dienst verwaltet werden, als auch das Übertragen der Verantwortung für einen Dienst an einen Nachfolger sowie das Löschen eines Dienstes.

F6: Dienstverwaltung

Zuständige Administratoren müssen die bei der TTP gespeicherten Eigenschaften ihrer Dienste verwalten können. Dazu gehört neben dem Modifizieren von Eigenschaften auch die Übertragung der Verantwortung für einen Dienst an einen Nachfolger sowie das komplette Entfernen eines Dienstes.

Damit die Organisationen, die eine dynamische virtuelle Föderation bilden möchten, dies einfach und schnell durchführen können, ist es wichtig, dass die jeweils verwendete Software über möglichst viele Versionen hinweg kompatibel ist. Ein neues Softwarepaket zu installieren oder Versionen wegen Inkompatibilität patchen zu müssen, kostet in fast allen Fällen viel Zeit. Um Ausfälle in gleichzeitig produktiv verwendeten Föderationen zu vermeiden, müssen zunächst Tests durchgeführt und ein für die Benutzer wenig störender Termin für die Umstellung gefunden werden. Dieser Aufwand würde das ganze System schon vom Start weg ausbremsen. Diese Anforderung ist zwar nicht für die Funktionalität eines Systems wichtig, aber für die Akzeptanz nicht zu vernachlässigen, weshalb sie als erweitert eingeordnet

wird.

> **F7: Kompatiblität**
>
> Damit Installation und Updates den Betrieb innerhalb einer Föderation nicht stören, müssen Updates abwärtskompatibel und die verwendeten Systeme idealerweise schon vorhanden bzw. bekannt sein.

Ein System, das darauf basiert, viele nicht unter der Kontrolle einer Organisation stehender Systeme zu verbinden, muss zudem sehr fehlertolerant sein. Dies wird üblicherweise durch Lose-Kopplung der Systeme erreicht. Diese Anforderung ist erweitert, da sie nicht unbedingt benötigt wird, aber stark zur Skalierbarkeit des Gesamtsystems beiträgt.

> **F8: Fehlertoleranz**
>
> Die eingesetzten Systeme müssen fehlertolerant und darauf ausgelegt sein, dass andere Organisationen eventuell leicht andere Implementierungen verwenden.

3.1.1.2. Nichtfunktionale Anforderungen

Verwandt mit der Kompatibilität und Fehlertoleranz der verwendeten Systeme, ist die Wartbarkeit. In einer kleinen Community oder einem kleinen Projekt sind meist nicht viele Ressourcen verfügbar, um die Authentifizierungsinfrastruktur zu warten. Zusätzlich soll die Hürde zum Beitritt einer Föderation nicht durch einen komplizierten Installationsvorgang erschwert werden. Dies ist eine erweiterte, nichtfunktionale Anforderung, da sie nicht direkt zur Funktionalität des Systems beiträgt, aber für dessen Akzeptanz nötig ist.

> **NF1: Wartbarkeit**
>
> Zur Minimierung des Supportaufwands soll das Einrichten und Aktualisieren der benötigten Softwarekomponenten möglichst einfach sein.

Bei einer großen Zahl an Organisationen, die an der AAI teilnehmen möchten, ist es wichtig, dass das verwendete System möglichst lange einsetzbar ist. Zudem arbeiten wissenschaftliche Einrichtungen und Hochschulen meist nicht gewinnorientiert und setzen gerne frei verfügbare Software ein, weshalb durch die eingesetzte Lösung keine weiteren Kosten entstehen sollten. Dies ist eine grundlegende Anforderung, da das System nur dann gut funktioniert, wenn es eine große Zahl an potentiell teilnehmenden Diensten gibt.

> **NF2: Kosten**
>
> Zusätzliche Kosten durch Lizenzen oder ähnliches sind zu vermeiden, um möglichst vielen SPs und IDPs einen einfachen Zugang zu ermöglichen.

Da selbst eine Lösung, bei der keine Lizenzkosten für Installation oder Betrieb anfallen, bei der Einrichtung und dem Betrieb Kosten durch das dazu nötige Personal und eventuellen Schulungen verursacht, ist es für die Akzeptanz wichtig, dass die Lösung mittel- bis langfristig eingesetzt werden kann. Dies ist eine nichtfunktionale erweiterte Anforderung. Die

Anforderung ist zwar wichtig, allerdings ist bei modernen, aktuell noch in der Entwicklung steckenden Methoden, für FIM nicht leicht absehbar, welche Lösung sich in Zukunft durchsetzten wird, weshalb es nicht machbar sein wird, die eine zukunftssichere und innovative Methode zu finden.

NF3: Zukunftssicherheit

Die Kernfunktionalität der Software muss auch in zukünftigen Funktionen erhalten und abwärtskompatibel sein.

Da das hauptsächliche Einsatzgebiet des Systems fokussiert auf Hochschulen und wissenschaftliche Einrichtungen ist, von denen einige speziell daran interessiert sind, moderne Sicherheitssysteme zu untersuchen und zu erweitern, sollte die Lösung als Open Source Projekt veröffentlicht werden können. Dies ist eine grundlegende nichtfunktionale Anforderung, die die Akzeptanz bei der für den Betrieb zuständigen Zielgruppe erhöht.

NF4: Open Source

Eine als Open Source verfügbare Software erleichtert die Erweiterung und Analyse der Software an den benötigten Stellen.

Zudem setzen die verschiedenen Organisationen unterschiedliche Systeme zur Datenhaltung im Hintergrund ein. Um möglichst vielen Organisationen den Zugang zu der FIM-Infrastruktur zu erleichtern, muss das im Hintergrund verwendete Speichersystem leicht abstrahierbar und mit der FIM-Lösung verwendbar sein. Dies ist eine erweiterte nichtfunktionale Anforderung, da sie nicht direkt zur Authentifizierung eines Nutzers beiträgt.

NF5: Anwendbarkeit

Durch eine flexible Abstraktion von bestehenden Speicher- und Authentifizierungssystemen, soll das System in möglichst vielen Fällen einsetzbar sein.

Ein weiterer wichtiger Punkt für die Interoperabilität von verschiedenen Systemen ist es, dass sie ein gemeinsames Protokoll verwenden und so auch verschiedene Implementierungen miteinander einsetzbar sind. Im Umfeld von Hochschulen wird für FIM das SAML-Protokoll verwendet, das zum Beispiel von den Softwaren Shibboleth und simpleSAMLphp implementiert wird. Eine Erweiterung muss daher um die Kompatibilität mit anderen Produkten zu wahren, über ein Protokoll verfügen, das von verschiedenen Softwareentwicklern implementiert werden kann. Diese Anforderung ist grundlegend für einen klar definierten Programmablauf und die Einsetzbarkeit in einer Umgebung, in der verschiedenste Implementierungen miteinander funktionieren müssen.

NF6: Protokoll

Damit die Erweiterung einem definierten Ablauf folgt und auch mit anderen Softwareumgebungen implementiert werden kann, wird ein Protokoll benötigt.

3.1.2. Metadatensynchronisation

Die zentrale Funktion der TTP ist es, die Metadaten zwischen den Teilnehmern zu synchronisieren. Der Metadatenaustausch wird benötigt, damit die an der dynamischen virtuellen Föderation beteiligten Organisationen nur initial, zum Austauschen der Metadaten die TTP benötigen und danach sicher direkt miteinander kommunizieren können. Dieser initiale Austausch soll auch nicht zu lange dauern. Als Richtwert werden hier zunächst wenige (< 5) Minuten als maximale Dauer angesetzt. Ob dieser Wert realistisch umsetzbar ist, bzw. wie weit sich die Zeit reduzieren lässt, wird genauer bei der Implementierung und Evaluation der Implementierung zu betrachten sein. Diese Anforderung ist grundlegend für die Skalierbarkeit und damit die Akzeptanz sowie der praktische Einsetzbarkeit des Systems.

F9: Metadatensynchronisation

Die Synchronisation von Metadaten soll erst bei Bedarf stattfinden, für den Nutzer transparent und mit einer maximalen Verzögerung von wenigen Minuten stattfinden.

Um festzustellen, wann es Bedarf für den Austausch von Metadaten gibt, bietet es sich an, dass der Benutzer eines Dienstes, zu dem sein IDP noch keine Verbindung hat bzw. den er gar nicht direkt kennt, den Vorgang anstoßen kann. Hier soll es für den Benutzer möglich sein, die Verbindung der Dienste auf simple Art und Weise zu initiieren und innerhalb weniger Minuten über den Erfolg der Verbindung benachrichtigt zu werden. Der Verbindungsaufbau selbst soll für den Benutzer weitestgehend transparent sein, so dass sich der Anmeldevorgang nur minimal von einer „normalen“ Anmeldung unterscheidet. Diese Anforderung ist grundlegend für die Dynamik des Systems.

F10: Benutzerinitiierter Verbindungsaufbau

Für den Fall, dass es zwischen dem SP, den der Benutzer verwenden möchte und seinem IDP noch keine Vertrauensbeziehung gibt, soll der Benutzer in der Lage sein, diesen Verbindungsaufbau selbst zu initiieren und im Erfolgsfall innerhalb von wenigen Minuten den Dienst benutzen können.

Da der Benutzer entsprechend der vorherigen Anforderung F10 in der Lage sein muss, den Verbindungsaufbau zu initiieren, muss er bei einem entsprechend eingerichteten SP die Option haben, durch die TTP „seinen“ Heimat-IDP auszuwählen. Hierzu muss eine geeignete Methode gewählt werden, von der Loginseite des SPs die TTP auszuwählen und eine Liste der über die TTP erreichbaren IDPs anzuzeigen. Die Auswahl des Heimat-IDPs sollte dabei so einfach wie möglich gemacht werden. Diese Anforderung ist grundlegend, da das System von regulären Benutzern bedient werden können muss.

F11: IDP Auswahl

Der Benutzer muss in der Lage sein ausgehend von der Loginseite des SPs die TTP und seinen dort vertretenen IDP selbst anzugeben oder auszuwählen.

3.1.3. Attributsformate

Neben der reinen Authentifizierung eines Benutzers sollen durch die AAI einer Föderation auch Attribute von Benutzern vom IDP zum SP übertragen werden können. Bei dem Zusammenstellen von Organisationen zu einer dynamischen virtuellen Föderation treten dabei zwei Probleme auf. Zum einen muss, wie im folgenden Abschnitt näher beschrieben wird, definiert sein, welche Bedeutung ein Attribut hat. Zum anderen muss, wie in Abschnitt 3.1.3.2 beschrieben werden wird, die Möglichkeit bestehen, zwischen Attributen zu konvertieren.

3.1.3.1. Attributsstandardisierung

Da es das Ziel dynamischer virtueller Föderationen ist, mit möglichst jeder anderen Organisation eine Föderation zu einem beliebigen Zweck zu betreiben, ist es nicht ideal, wie zum Beispiel in einer nationalen Föderation, von Beginn an festzulegen, dass eine spezifische Menge an Attributen von einem IDP angeboten werden muss. Stattdessen sollte nur eine standardisierte Methode zur eindeutigen Identifizierung der Attribute verwendet werden. Diese Anforderung ist grundlegender Natur, ohne eine eindeutige Identifizierung kann keine Aussagen über die Semantik eines Attributs gemacht werden.

> **F12: Attributsidentifikation**
>
> Zur eindeutigen Identifizierung der verwendeten Attribute soll ein standardisierter, eindeutiger Schlüssel verwendet werden.

3.1.3.2. Attributskonvertierung

Da im vorherigen Abschnitt nur die eindeutige Identifizierbarkeit von Attributen als Anforderung aufgestellt wurde, müssen entsprechend Voraussetzungen geschaffen werden, um in den meisten Fällen vorhandene von der Semantik ähnliche, aber nicht identische Attribute geeignet zu konvertieren. Beispiele für solche Attribute – wie verschiedene Datums- und Zeit-Formate – wurden schon in der Motivation in Abschnitt 1.1 beschrieben.

Für die Konvertierung dieser Attribute wird ein Verzeichnis zur Verwaltung der verfügbaren Konvertierungsregeln benötigt. Dieses Verzeichnis soll allen Teilnehmern der, im vorherigen Abschnitt beschriebenen, TTP zur Verfügung stehen. Administratoren sollen in diesem Verzeichnis Konvertierungsregeln suchen, anzeigen, erstellen, modifizieren und bewerten können. So kann zunächst gesucht werden, ob eine bestimmte Regel schon existiert, wenn diese unvollständig oder fehlerhaft ist, eventuell modifiziert und bewertet werden oder, falls es noch keine Regel gibt, eine neue erstellt werden. Das zentrale Verzeichnis soll verhindern, dass Regeln unnötigerweise doppelt erstellt werden, so dass im Idealfall nur zu Beginn Regeln erstellt werden müssen und danach von anderen Teilnehmern schon existierende Regeln weiterverwendet werden können. Diese Anforderung ist grundlegend, obwohl das System eventuell auch ohne ein zentrales Verzeichnis funktionieren würde, wenn jeder Administrator eigene Regeln verwalten würde. Die lokale Verwaltung würde aber die Skalierbarkeit einschränken, da jeder Administrator manuell die passende Regel suchen bzw. schreiben und implementieren müssten.

F13: Konv.-Regelverwaltung

Es soll einen zentralen Dienst zum Suchen, Anzeigen, Modifizieren und Erstellen von Konvertierungsregeln geben.

Für die Suche nach passenden Konvertierungsregeln muss angegeben werden können, welche Attribute eine Regel benötigt um ein anderes Attribut daraus abzuleiten. Dazu werden zum Beispiel Metainformationen wie `from` und `to` benötigt. Neben diesen Basisinformationen sind können beliebige Zusatzinformationen zu einer Regel hinterlegt werden. Zum Beispiel welcher Benutzer wann diese Regel erstellt hat. Zudem könnte eine Regel durch einen kurzen Freitext beschrieben werden. Für die effiziente Suche nach passenden Regeln und der Auswahl der am besten passenden Regel ist diese Anforderung grundlegend.

F14: Konv.-Regelbeschreibung

Die Konvertierungsregeln sollen über Metainformationen verfügen, die mindestens beschreiben, welches Attribut aus welchen Attributen abgeleitet wird. Diese Informationen sollen die Suche nach der am besten passenden Regel erleichtern.

Zusätzlich soll eine Bewertungsmetrik, die Werte wie Vollständigkeit, Korrektheit und Zuverlässigkeit widerspiegelt, für eine Regel gespeichert werden. Diese ist einer erweiterte Anforderung, da sie nicht unmittelbar notwendig, in einem System mit vielen Konvertierungsregeln, allerdings sehr nützlich ist.

F15: Konv.-Regelbewertung

Die Konvertierungsregeln sollen bewertet werden können, um deren Qualität in Hinsicht auf Vollständigkeit, Korrektheit und Zuverlässigkeit abschätzen zu können.

Für die Konvertierungsregel selbst muss eine geeignete Beschreibungssprache gewählt werden, die eventuell anfallende Konvertierungsarten unterstützt. Zu den voraussichtlich benötigten Konvertierungen gehören:

- **Umbenennung von Attributen**: Die Umbenennung von Attributen wird benötigt, wenn zwei ansonsten identische Attribute einen unterschiedlichen Namen/Identifier haben.
- **Abbilden von Attributen**: Das Abbilden von Attributs-Werten kann in Situationen eingesetzt werden, wenn zwei Attribute die selbe Eigenschaft durch unterschiedliche Werte beschreiben.
- **Kombination von Attributen**: Durch das Kombinieren zweier Attribute, kann eventuell ein weiteres Attribut abgeleitet werden.
- **Extraktion von Attributen**: Die Umkehrung der Kombination von Attributen ermöglicht es nur, Teile eines Attributes zu verwenden, um daraus ein anderes Attribut abzuleiten.
- **Skriptbasierte Verarbeitung**: Für komplizierte Kombinationen, die eventuell Be-

rechnungen wie bei der Datumstransformation benötigen, soll es möglich sein, die Konvertierung durch eine Skriptsprache zu definieren.

In Abschnitt 3.2.2.4 wird bei der Untersuchung momentan in Föderationen eingesetzter Schemata an konkreten Beispielen gezeigt, wie diese Konvertierungsarten angewandt werden können. Diese Anforderung ist grundlegend, da die Konvertierungsregeln nur dann wirklich einsetzbar sind, wenn alle in der Praxis anfallenden Konvertierungsarten durchgeführt werden können.

F16: Konv.-Regeloperationen

Das System zur Konvertierung von Attributen muss die Transformationen Umbenennung, Abbildung, Kombination, Extraktion und skriptbasierte Konvertierungen unterstützen.

Damit die Konvertierungsregeln auch verwendet werden, müssen sie einfach zugänglich gemacht und möglichst automatisch angewendet werden können. Dazu müssen IDP und/oder SP in der Lage sein, festzustellen, dass die von ihnen angeforderten/angebotenen Attribute nicht zueinander passen und versuchen, über das Konvertierungsregelverzeichnis entsprechend passende Konvertierungen zu finden. Diese Anforderung ist grundlegend, damit die Regeln, der Grundidee der dynamischen virtuellen Föderation entsprechend, automatisch und dadurch sehr dynamisch genutzt werden können.

F17: Konv.-Regelabgleich

Das verwendete System muss durch einen automatischen Abgleich von verfügbaren und geforderten Attributen geeignete Konvertierungsregeln finden und anwenden können.

Falls die Konvertierung nicht erfolgreich eingerichtet wird, kann der zuständige Administrator des IDPs und des SPs darüber informiert werden. Die Administratoren können dann eigene Regeln schreiben, um die Konvertierung zu ermöglichen. Da die Regeln allen anderen Teilnehmern zugänglich gemacht werden, profitiert das gesamte Teilnehmer-Netzwerk. Diese Anforderung ist eine Erweiterung, die optional ist und nicht für die Grundfunktionalität benötigt wird. Diese Anforderung nicht umzusetzen, widerspricht durch eine nicht definierte manuelle Fehlerbenachrichtigung aber der Idee der dynamischen virtuellen Föderation, die den Ablauf des Föderationsaufbaus möglichst weit automatisieren möchte.

F18: Fehlerbenachrichtigungen

Bei durch inkompatible Attribute nicht aufgebauter Verbindungen, sollen die für die Dienste zuständigen Administratoren informiert werden.

3.1.4. Rollenbasierte Anforderungen

In den folgenden Abschnitten werden Anforderungen an die AAI einer dynamischen virtuellen Föderation aus Sicht von Identity Providern, Service Providern und Benutzern betrachtet.

3.1.4.1. Identity Provider

Aus der Sicht eines Identity Providers ist die Teilnahme an föderiertem Identity Management ein Dienst, den er seinen Benutzern anbieten kann. Im Falle einer Universität muss generell ein System zur Verwaltung aller Mitarbeiter und Studenten existieren, welches über sehr aktuelle und detaillierte Daten verfügt, die durch FIM anderen SPs auf eine sichere Art und Weise zugänglich gemacht werden können. Der IDP profitiert dabei auch dadurch, dass er selbst häufig einen oder mehrere SPs betreibt, die dann von Benutzern anderer IDPs verwendet werden können.

Neben der erleichterten Bedienung von Diensten ist für einen IDP allerdings der Datenschutz seiner Benutzer besonders wichtig, da er bei unkontrollierter Weitergabe von Benutzerinformationen schnell das Vertrauen der Benutzer verlieren und sich eventuell sogar strafbar machen kann. Damit das System skalierbar ist, muss innerhalb der Föderation mindestens ein Verhaltenskodex (code of conduct) existieren, um Absprachen zwischen jeder IDP- SP-Kombination zu vermeiden. Diese nichtfunktionale Anforderung ist grundlegend für das Vertrauen der IDPs in das verwendete System.

> **NF7: Verlässlichkeit der SPs**
>
> IDP und SP sollen an einen Verhaltenskodex, der unter anderem regelt, wie die von einem IDP übermittelten Benutzerdaten verwendet werden und welche Datenschutzbestimmungen eingehalten werden müssen, gebunden sein.

Neben der organisatorischen Lösung, wie einem Verhaltenskodex, kann der IDP über Attribute Release Policies (ARP) angeben, welche Attribute er bereit ist, an einen Dienst zu übermitteln. Vielen Diensten reicht ein für den Dienst eindeutiges Benutzertoken und die Feststellung, dass der Benutzer einen Account an z.B. einer Universität hat, um den Dienst zu benutzen. Eine bessere Kontrolle über die Attributsfreigaben erlaubt es mehr IDPs an der Föderation teilzunehmen, da die Gefahr, ungewollt Informationen über die Benutzer an nicht vertrauenswürdige SPs herauszugeben, reduziert werden kann. Der IDP soll also selbst festlegen können, welche Attribute, die er von seinen Benutzern speichert, er an einen bestimmten SP herausgeben möchte. Natürlich ergibt sich daraus, dass ein IDP, der wenige oder keine Attribute herausgibt, eventuell nicht in einer Föderation einsetzbar ist, deren SPs bestimmte Attribute benötigen. Diese Anforderung ist daher erweitert, um dem IDP eine bessere Kontrolle zum Datenschutz der Benutzer zu geben.

> **F19: Attribute Release Policies**
>
> Der IDP soll über ARPs festlegen können, welche Attribute er an einen SP herausgibt.

Damit ein IDP noch besser kontrollieren kann, an welche SPs er Attribute herausgibt, möchte er eventuell eine Black- bzw. White-List führen. Damit kann er einzelne Dienste ausschließen bzw. nur bestimmte Dienste erlauben. Dies ist natürlich auch umgekehrt, aus Sicht des, im folgenden Abschnitt betrachteten, SPs interessant, um zum Beispiel IDPs mit schlechter Datenqualität von vornherein auszuschließen. Diese Anforderung ist ebenfalls erweiternd für den Datenschutz der Benutzer.

F20: Access Control Lists

Der IDP und SP sollen über ACLs festlegen können, welche Dienste mit ihrem Dienst verwendet werden können.

3.1.4.2. Service Provider

Ein Service Provider hat durch die Verwendung von FIM den Vorteil, dass er sich nicht vollständig mit der Authentifizierung von Benutzern und der Speicherung ihrer Daten beschäftigen muss. Einen Großteil der Verantwortung, der bei der Authentifizierung auftritt, wie die Wahl einer geeigneten Authentifizierungsmethode, der Einrichtung der Authentifizierungsinfrastruktur und der Durchführung der Identifizierung sowie der sicheren Speicherung von Passwörtern, der Einhaltung von Passwort-Richtlinien und Bearbeitung von Supportanfragen wegen vergessener Passwörter, kann der SP an den IDP übertragen. Der SP muss nur sicherstellen, dass er die Authentifizierungsbestätigung des IDP korrekt überprüft. Dies ist eine grundlegende Anforderung, die durch das FIM-System gegeben ist.

NF8: Authentifizierungs-Verantwortung

Übertragen der Verantwortung bei der Verwaltung von Authentifizierungsinformationen wie Passwörtern und des damit zusammenhängenden Supports von dem Dienst auf den IDP.

Ein weiterer Vorteil für den Serviceproviders ist, dass der IDP die Verwaltung der Benutzerinformationen übernimmt. Der IDP ist dazu in vielen Fällen geeignet, da der Benutzer eine direktere Beziehung zu dem IDP pflegt als zu den verschiedenen von ihm genutzten Diensten, wodurch der IDP schneller Änderungen an den Benutzerinformationen erhält als die verschiedenen Dienste. Informationen wie die aktuelle Semesterzahl kann eine Universität als IDP zum Beispiel immer selbständig aktuell halten. Zusätzlich zu der Verwaltung einiger Informationen gibt der IDP häufig Garantien über die Genauigkeit und das Alter der von ihm herausgegebenen Attribute eines Benutzers. Dies ist eine nichtfunktionale erweiterte Anforderung, die die Verwendung von FIM für einen SP attraktiver macht.

NF9: Datenaktualität

Entsprechend der Stellung des IDP kann dieser verlässlichere und aktuellere Benutzerinformationen liefern. Im Idealfall beinhaltet dies auch eine Zusicherung der Aktualität der Daten.

Neben den Vorteilen entstehen für einen SP auch Nachteile. Zum einen gibt es viele vorgefertigte Authentifizierungssysteme für webbasierte Dienste, die kein SAML verwenden und einfach zu installieren und administrieren sind. Der Umstieg von einem System, das die Authentifizierung lokal beim SP durchführt auf ein SAML-basiertes System ist aufwändig und benötigt eine Migrationsphase, in der bestehende Kunden in das neue System übernommen werden müssen.

3.1.4.3. Benutzer

In diesem Abschnitt wird beschrieben, wie SAML aus Benutzersicht verwendet werden kann, um nachzuweisen, dass man bei einem IDP bekannt ist und von diesem bestimmte Eigenschaften bestätigt bekommt. Ein ähnliches Beispiel würde auch für OpenID funktionieren, da im Hochschulumfeld allerdings SAML am weitesten verbreitet ist, wurde für das Beispiel SAML ausgewählt. Hier wird der Microsoft-Store als Service Provider und als Identity Provider die LMU verwendet. Ein Ziel ist es, durch eine Erweiterung diesen bei den Benutzern bekannten Ablauf nicht unnötig zu verändern oder komplizierter zu machen. Diese Anforderung ist wünschenswert, aber nicht für die Funktion des Systems essentiell, weshalb sie als optional bezeichnet wird.

> **NF10: User-Experience-Konsistenz**
>
> Der Ablauf der Authentifizierung über FIM soll bei einer Erweiterung für den Benutzer möglichst wenige Änderungen an dem ihm bekannten Ablauf bewirken.

Um im Microsoft-Store nachzuweisen, dass ein Benutzer als Student an einer Hochschule eingeschrieben ist, bietet Microsoft verschiedene Optionen der Verifikation an. In diesem Fall interessiert die Option, die – wenn auch nicht explizit erwähnt – SAML zur Abfrage von Benutzer-Attributen verwendet. In Abbildung 3.1 ist zu sehen, wie die entsprechende Option ausgewählt wurde, über die Länderauswahl implizit die nationale Föderation bestimmt wurde und dann aus den Metadaten der DFN-AAI die LMU ausgewählt wurde. Dies zeigt einen möglichen Lösungsansatz für die Auswahl des IDPs durch den Benutzer, wie es in Anforderung F11 gefordert wird.

Daraufhin wird der Benutzer zu einer Seite seiner Hochschule weitergeleitet, auf der er sich mit seiner Benutzerkennung und Passwort anmelden muss. Nach einer erfolgreichen Anmeldung wird dem Benutzer dann gezeigt, welche Attribute, d.h. Informationen über den Benutzer, an den SP Microsoft übermittelt werden. Ein solcher Dialog ist in Abbildung 3.2 zu sehen.

Stimmt der Benutzer der Übermittlung der angegebenen Attribute zu, wird er wieder zurück auf die Seite von Microsoft geleitet, die anhand der Attribute feststellen kann, dass der Benutzer tatsächlich ein aktuell eingeschriebener Student an der LMU ist.

In diesem Beispiel wird eine von der Internet2 Initiative standardisiertes Attributsschema `eduPerson` [Int12] verwendet. Dieses Schema definiert die Attribute, die mit `eduPerson` beginnen. Für Microsoft interessant ist zum einen die `eduPersonAffiliation`, die als „Specifies the person's relationship(s) to the institution in broad categories such as student, faculty, staff, alum, etc.“ [Int12] definiert ist, also angibt, welchen Gruppen die beschriebene Person an der Hochschule angehört. Außerdem wird eine Person eindeutig über die `eduPersonTargetedID` identifiziert. Diese ID ist eine dauerhaft, nicht neu vergebene und nicht interpretierbare Zeichenkette.

In diesem Beispiel wird gezeigt, dass diese Methode die Daten des Benutzers gut schützt. Microsoft erhält keinen Namen oder andere Informationen über den Benutzer, kann aber über die ID sicher sein, dass sich ein Student nur in einem Microsoft-Account für die Studentenrabatte registriert. Diese erweiterte Anforderung ist nicht grundlegend für das Funktionieren

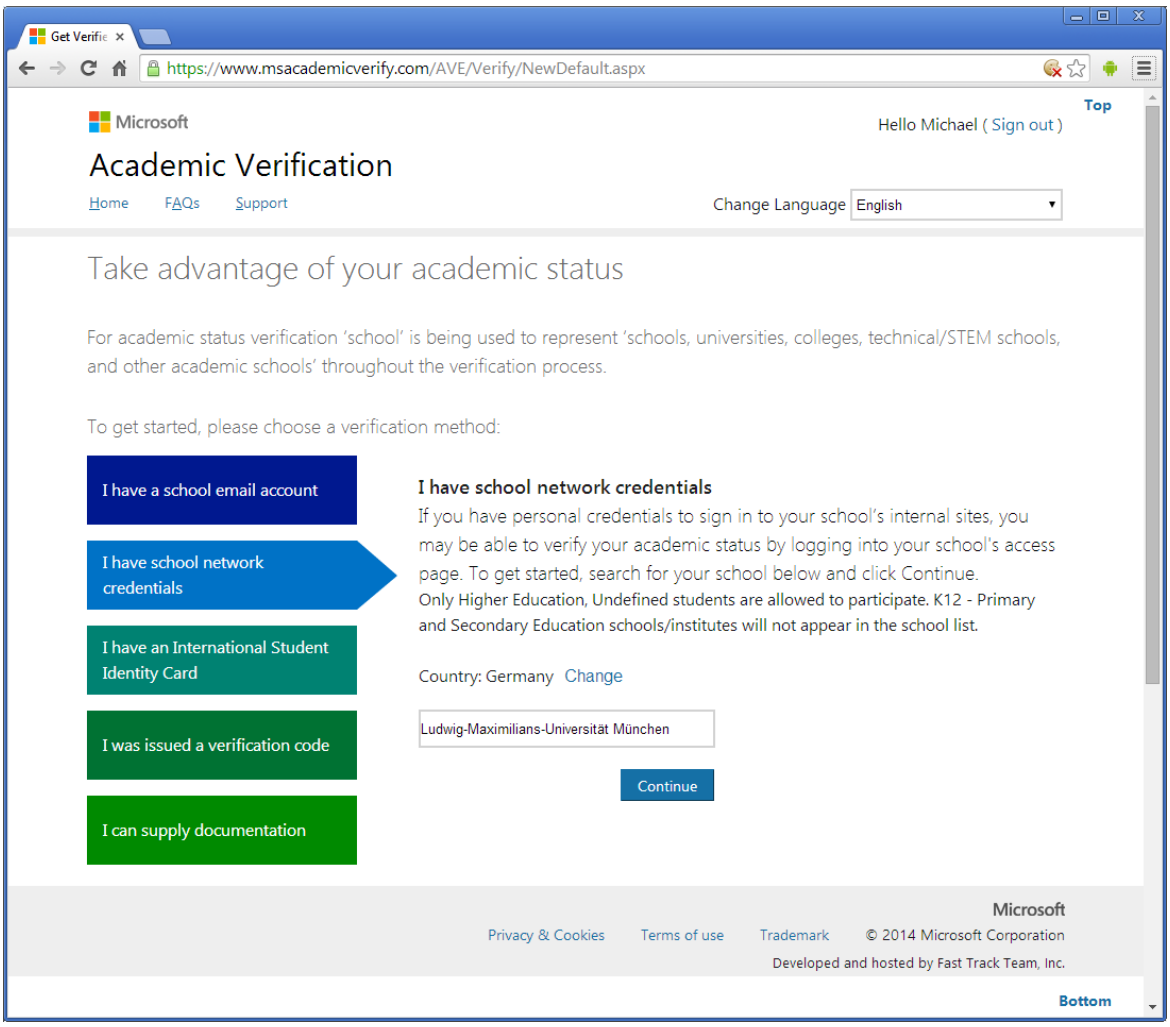

Abbildung 3.1.: Freigabe von Attributen durch den Benutzer am Beispiel des LMU-IDPs.

eines Authentifizierungssystems aber wichtig für die Akzeptanz durch den Benutzer.

F21: Attributsfreigabe

Der Schutz der Benutzerinformationen muss auch vom Benutzer kontrollierbar sein, indem er die Möglichkeit hat, zu überprüfen, welche Informationen an einen Dienst übermittelt werden.

3.2. Anwendbarkeit dynamischer Föderationen

In diesem Abschnitt wird das Konzept dynamischer virtueller Föderationen auf klassische Konzepte angewendet. Dazu werden im nächsten Abschnitt Communtiy- und Projekt-Föderation und in Abschnitt 3.2.2 (Inter-)Föderationen betrachtet. Die vorher aufgestellten Anforderungen werden dabei um speziell in diesen FIM-Anwendungsfällen auftretenden An-

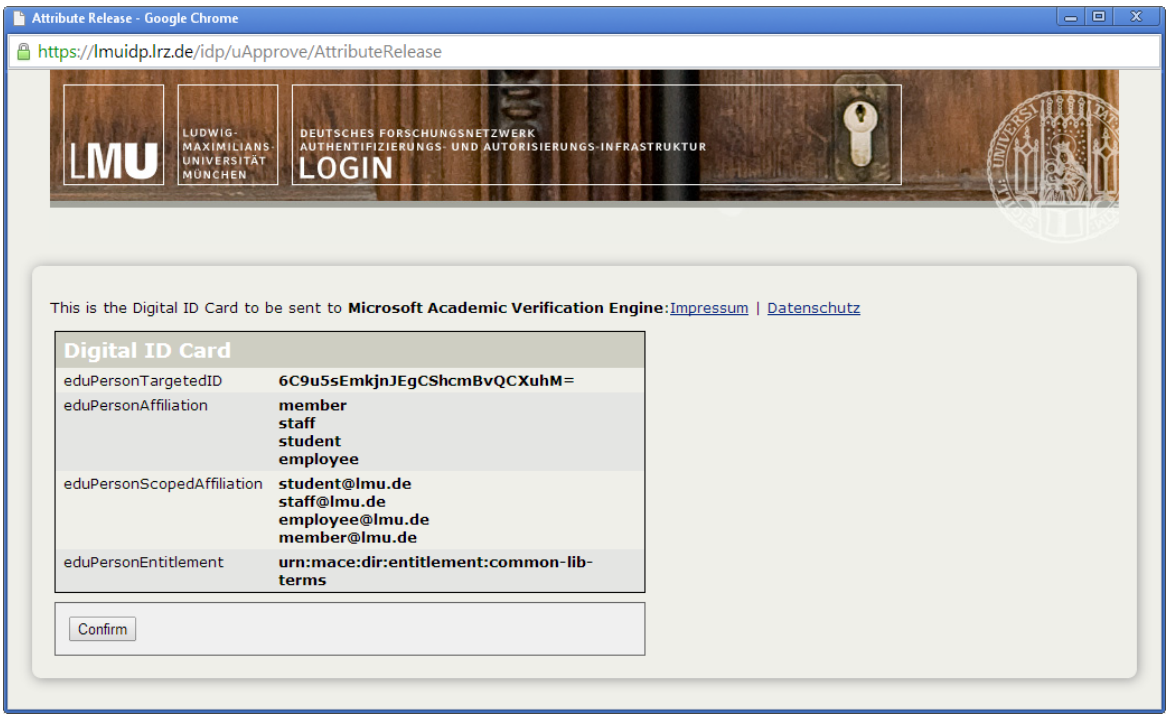

Abbildung 3.2.: Freigabe von Attributen durch den Benutzer am Beispiel des LMU-IDPs.

forderungen erweitert.

3.2.1. Community- und Projekt-Föderationen

Da das Konzept dynamischer virtueller Föderationen vor allem auf den Anforderungen von Communities und Projekten basiert, wird in diesem Abschnitt zuerst überprüft, ob die spezifischen Anforderungen an FIM durch Communities und Projekte von den Anforderungen an dynamische virtuelle Föderationen abgedeckt werden.

3.2.1.1. Allgemeine Anforderungen

Projektspezifische bzw. Community-Föderationen haben im Allgemeinen eine schlechte Skalierbarkeit, das heißt, dass sich die Mitglieder jeweils darum kümmern müssen, Metadaten auszutauschen und aktuell zu halten. Durch die Dynamik, dass während der Laufzeit eines Projektes Projektpartner neu hinzukommen oder aussteigen, ist der zur Einrichtung nötige Arbeitsaufwand nicht nur einmalig, sondern tritt bei jeder Änderung neu auf. Aus diesem Grund wurde das Konzept der dynamischen virtuellen Föderationen entworfen, durch das die Verbindungen zwischen IDPs und SPs bei Bedarf dynamisch aufgebaut werden können.

Zudem dauert es bei klassischen Projekten eine Weile, bis alle Teilnehmer die nötigen Änderungen an ihren Konfigurationen durchgeführt haben. Im Normalfall sind die für den

Betrieb der FIM-Infrastruktur zuständigen Personen nicht in ein spezielles Projekt eingebunden, sodass Änderungen erst beantragt werden müssen. Zudem müssen Protokolle zum Change-Management befolgt werden, welche das schnelle Einbringen von Änderungen verzögern. Gerade bei größeren Projekten mit vielen Teilnehmern kann es dadurch länger dauern, bis die Änderungen bei allen implementiert sind. Diese beiden Aspekte sind durch die Anforderungen F1 zum dynamischen Verbindungsaufbau abgedeckt.

Für den dynamischen Verbindungsaufbau ist durch die Anforderung F2 eine TTP vorgesehen. Für eine Community bzw. ein Projekt stellt sich dabei die Frage, ob die TTP selbst betrieben werden soll oder ob die TTP einer übergeordneten Instanz verwendet werden kann. Prinzipiell spricht nichts dagegen, eine eigene TTP zu betreiben, allerdings basiert die Skalierbarkeit darauf, dass es eine für möglichst viele Teilnehmer zugängliche TTP gibt. Wenn jede Community oder jedes Projekt wieder eine eigene AAI einrichten müsste, hat der Ansatz keine Vorteile gegenüber dem aktuell verwendeten. Daher haben Communities und Projekte die zusätzliche Anforderung, Zugang zu einer alle potentiellen Teilnehmer umfassenden TTP zu haben, die sie für ihre Zwecke verwenden können. Aus Sicht der in Abschnitt 2.2.3 vorgestellten CLARIN-Community könnte zum Beispiel eduGAIN eine solche TTP betreiben und damit die Probleme lösen, aus denen eduGAIN im Moment nicht für CLARIN interessant ist. Die Anforderung, das jeder Provider die Möglichkeit hat, Zugang zu einer TTP zu bekommen, ist daher grundlegend für die Einsetzbarkeit in Communities und Projekten.

NF11: Zugang zu einer TTP

Die Teilnehmer einer Community bzw. eines Projekts sollen Zugang zu einer TTP haben, durch die sie für ihre Zwecke eine dynamische virtuelle Föderation erstellen können.

Wenn diese Anforderung erfüllt wird, können Communities und Projekte von dynamischen virtuellen Föderationen profitieren. Unter dieser Voraussetzung sind auch Communities und Projekte abhängig von den Anforderungen F2, F3, F4, F5 und F6, die, die TTP und deren Verwaltung betreffen.

Da Communities und Projekte zum Teil hoch spezialisierte Forschungsgebiete haben und zum Beispiel bei medizinischen Projekten auch große Mengen an vertraulichen Patientendaten austauschen und analysieren wollen, haben sie besondere Anforderungen an eine sichere Authentifizierung und feingranulare Einschränkung der zugriffsberechtigten Personen. Diese Anforderung wurde nicht explizit im Abschnitt 3.1.1 für dynamischen virtuellen Föderationen gefordert. Für die betroffenen Communities und Projekte ist diese Anforderung allerdings grundlegend und ist unter Umständen bei der Verarbeitung persönlicher Daten sogar gesetzlich vorgeschrieben. Die Art und Weise, wie Zugriffe geregelt werden, ist allerdings nicht von der AAI, sondern von der Implementierung der SP-Software abhängig.

Im Zusammenhang mit der sicheren Zugriffssteuerung steht auch die Möglichkeit Zugriffe im Nachhinein zu auditieren, d.h. nachzuvollziehen, wer wann auf welche Attribute zugegriffen hat bzw. welcher IDP die Identität welches Benutzers bestätigt hat. Diese Anforderung ist nicht direkt für die AAI notwendig aber relevant für den praktischen Einsatz, daher wird diese Anforderung als optional eingeordnet.

F22: Auditierbarkeit

Es muss sowohl auf IDP- als auch SP-Seite die Möglichkeit geben einfach nachzuvollziehen, welches System, welche Attribute abgefragt, bzw. welche Aussagen über Attribute und Authentifizierung getroffen hat.

Ein weiteres hauptsächlich in Communities und Projekten auftretendes Problem ist es, dass ein Benutzer, der einen Dienst aus der Föderation verwenden möchte, keinen eigenen IDP hat. In diesen Fällen wird, wie am Beispiel von CLARIN in Abschnitt 2.2.3 beschrieben, ein von der Föderation betriebener IDP verwendet, an dem sich die Benutzer registrieren können und nach einer manuellen Überprüfung des Anliegens freigeschaltet werden. Diese Anforderung deckt einen Sonderfall ab, der selten benötigt wird und daher optional ist.

NF12: Homeless IDP

Es soll für Benutzer ohne eigenen Heimat-IDP eine Möglichkeit geben, trotzdem die Dienste der Föderation verwenden zu können.

Weitere hauptsächlich nichtfunktionale Anforderungen wie die Wartbarkeit NF1, Kosten NF2, Zukunftssicherheit NF3, Open Source NF4, Anwendbarkeit NF5, Fehlertoleranz F8 und ein Protokoll NF6, die für die dynamischen virtuellen Föderationen großteils von den Anforderungen durch Communities und Projekten abgeleitet wurden, werden beim Einsatz des neuen Konzeptes natürlich ebenfalls benötigt.

3.2.1.2. Metadatensynchronisation

Bei den Communities und Projekten wie sie in Abschnitt 2.2.3 vorgestellt wurden, werden die Metadaten auf unterschiedliche Methoden synchronisiert. CLARIN ermöglicht es, den teilnehmenden SPs zum Beispiel einen Metadatensatz für die teilnehmenden IDPs einfach herunterzuladen und dadurch in ihre Systeme zu integrieren. Die entgegengesetzte Richtung, die Metadaten der SPs in die Systeme der IDPs bzw. deren Föderationen zu integrieren, muss auf einer für jede Föderation unterschiedlichen Art und Weise erfolgen. Dadurch verlängert sich die Zeit, bis aktualisierte Metadaten an alle Teilnehmer verteilt wurden.

Durch die in Anforderung F9 geforderte bedarfsorientierte Metadatensysnchronisation würde sichergestellt, dass bei dem Verbindungsaufbau immer aktuelle Metadaten verwendet werden. Dazu muss wie in Anforderung F10 der Benutzer in der Lage sein, einen Verbindungsaufbau durch die Auswahl seines IDPs (Anforderung F11) anzustoßen.

3.2.1.3. Attributsformate

Da Communities und Projekte zu den unterschiedlichsten Themen und Zwecken gegründet werden können, ist es schwer, Aussagen über die von ihnen benötigten Attribute zu machen. Die Anforderung F12, welche eine eindeutige Identifikation der verwendeten Attribute fordert, ist in jeden Umfeld eine wichtige Voraussetzung, um den Wert des Attributs korrekt interpretieren zu können.

Eine größere Flexibilität bei der Verwendung von Attributen kann allerdings nicht schaden, solange die Komplexität des Systems dadurch nicht unverhältnismäßig erhöht wird. Daher profitieren Communities und Projekte auch durch die zur Attributskonvertierung spezifizierten Anforderungen F13, F14, F16 und F17 sowie der Benachrichtigung zu Fehlern bei der Konvertierung aus Anforderung F18.

3.2.1.4. Rollenbasierte Anforderungen

Identity Provider In der Regel sind Identity Provider daran interessiert, ihren Benutzern auf sichere Art und Weise Zugang zu möglichst vielen für sie relevanten Diensten zu ermöglichen. Daher sind IDPs meist Mitglieder in möglichst großen Föderationen, in denen, wie in Abschnitt 2.2.1 beschrieben, Verträge den Umgang mit durch die IDPs herausgegebenen Attributen regeln. Da es für eine Organisation in den meisten Fällen keinen Sinn macht, mehrere IDPs für verschiedene Föderationen zu betreiben, werden die selben Systeme für nationale und Community-Föderationen verwendet. Daher gibt es auch in Communities wie zum Beispiel CLARIN Verträge, die die Verwendung von Attributen und Diensten regeln. Die für dynamische virtuelle Föderationen aufgestellte Anforderung NF7, zur Verlässlichkeit des SPs, demnach auch in Communities und Projekten relevant.

In einem System, wie dem der dynamischen virtuellen Föderationen, in dem auch Verbindungen zwischen vorher sich nicht bekannten Systemen hergestellt werden können, ist es daher für IDPs generell wichtig, dass, wenn die Verlässlichkeit eines SPs nicht durch Verträge geregelt wird, die Anzahl der für diesen SP freigegebenen Attribute reduziert oder der SP komplett ausgeschlossen werden kann. Hierzu dienen die Anforderungen F19 und F20 zu Attribute Release Prolicies und Access Control Lists.

Service Provider In Communities und Projekten wollen Service Provider ihren Dienst den anderen Teilnehmern möglichst einfach zur Verfügung stellen. Dabei erleichtert ihnen FIM die Authentifizierung von Benutzern dadurch, dass sie einen Großteil der Verantwortung bei der Authentifizierung, wie in Anforderung NF8 beschrieben, auf den IDP übertragen können. Dabei möchte der SP auch eine Kontrolle darüber haben, welche IDPs mit seinem Dienst verwendbar sind. Dies wird durch die auch schon für IDPs beschriebene Anforderung F20, die fordert, dass durch ACLs IDPs auf eine Black-/White-List gesetzt werden können.

Die Anforderung NF9, über den IDP aktuelle Daten des Benutzers beziehen zu können, ist in Communities und Projekten besonders interessant. Da einzelne Mitarbeiter unter Umständen flexibel zu Projekten hinzukommen oder abgezogen werden, ist es durch die beim IDP hinterlegten Daten eventuell möglich, diese Veränderungen sofort bei der Authentifizierung zu betrachten.

Benutzer Innerhalb von Community- und Projekt-Föderationen haben Benutzer keine weiteren spezifischen Anforderungen an die AAI, die nicht schon in Abschnitt 3.1.4.3 aufgeführt wurden.

3.2.1.5. Fazit

Das Konzept der dynamischen virtuellen Föderation ist den Anforderungen nach in Community- und Projekt-Föderationen einsetzbar. Voraussetzung dazu ist, dass die Community bzw. das Projekt, wie in der zusätzlichen Anforderung NF11 beschrieben, Zugang zu einer TTP hat. Außerdem wäre es vorteilhaft, wenn die Interaktionen zwischen IDP und SP auditierbar wären. Dazu wurde eine weitere Anforderung F22 erstellt.

3.2.2. (Inter-)Föderationen

In diesem Abschnitt werden Anforderungen an eine AAI aus Sicht von nationalen Föderationen und Inter-Föderationen betrachtet. Klassische nationale Föderationen wurden bereits in Abschnitt 2.2 und Inter-Föderationen in Abschnitt 2.2.2 beschrieben.

3.2.2.1. Allgemeine Anforderungen

Eine Föderation bzw. deren Betreiber bietet den an der Föderation teilnehmenden Organisationen eine Rahmenstruktur sowohl in organisatorischer als auch technischer Hinsicht. Dazu gehört zum Beispiel das Verfassen von Richtlinien und Verträgen für die Mitglieder, sowie die Dokumentation von technischen Anforderungen und eventuell auch das Anfertigen von Implementierungsbeschreibungen und Installationsanleitungen. Daher ist es für eine Föderation wichtig, dass der Betrieb, der zum Teilnehmen nötigen Software möglichst einfach ist. Wie bei den dynamischen virtuellen Föderationen ist die Wartbarkeit (Anforderung NF1) daher auch in (Inter-)Föderationen wichtig.

Gerade das Einrichten und Aktualisieren von dem in nationalen Föderationen häufig verwendeten SAML ist relativ zu lokalen Authentifizierungssystemen komplex. Das wird vor allem dadurch deutlich, dass der 2005 durch SAML 2.0 abgelöste Standard SAML 1.1 heute noch von den meisten Diensten, um kompatibel zu alten Systemen zu sein, parallel unterstützt wird. Weitere Herausforderungen entstehen dadurch, dass die bei einem Authentifizierungsvorgang verwendeten Systeme nicht alle unter der Kontrolle einer Organisation stehen. Dadurch gibt es viele Stellen, an denen Fehler auftreten können. Um einen Großteil von Fehlern vor der Aufnahme in eine Produktionsumgebung zu finden und zu beheben, sieht zum Beispiel der Aufnahmevorgang bei der DFN-AAI vor, in einer Testföderation die korrekte Konfiguration eines Dienstes zu zeigen. Die Anforderung der Kompatibilität (F7) ist daher auch hier wichtig, wird aber um die optionale Möglichkeit Testsysteme aufzusetzten erweitert.

> **F23: Testsysteme**
>
> Bevor ein neuer Teilnehmer einer produktiven Föderation beitreten kann, soll innerhalb einer Testumgebung festgestellt werden können, ob die Software richtig konfiguriert ist.

Weitere allgemeine Anforderungen, wie die Wartbarkeit NF1, Kosten NF2, Zukunftssicherheit NF3, Open Source NF4, Anwendbarkeit NF5, Fehlertoleranz F8 und ein definiertes Protokoll NF6, finden auch in (Inter-)Föderationen Anwendung.

3.2.2.2. Metadatensynchronisation

Ein großes Problem bei der Skalierung von Föderationen tritt bei der Synchronisation der Metadaten zwischen den teilnehmenden Organisationen auf. Innerhalb von Föderationen werden deren Teilnehmer bei Änderungen an den zusammengefassten Föderations-Metadaten benachrichtigt, damit diese die neue Version herunterladen können (push-Modell) oder die Teilnehmer prüfen in regelmäßigen Abständen die Metadaten auf Änderungen (pull-Modell). Prinzipiell haben aber alle Teilnehmer immer den gesamten Satz Metadaten aller anderen an der Föderation teilnehmenden Organisationen bei sich gespeichert. Dadurch entsteht zwar im Moment durch die geringe Größe von Metadatendateien kein großer Overhead, der in Form von Speicherplatz oder verbrauchter Bandbreite. Bei einem System, dass Organisationen weltweit umfassen soll, wird die Größe der Gesamtmetadatendatei allerdings weiter steigen. Neben dem Speicherplatz und der Übertragung ist es für die lokale Administration eines Dienstes übersichtlicher, nur die Metadaten installiert zu haben, die auch tatsächlich verwendet werden.

Die Tabelle 3.1 zeigt anhand ausgewählter Födeationen die Größe der Föderations-Metadaten sowohl in Mega-Bytes (MB) als auch die Anzahl der dort eingetragenen Organisationen und die Anzahl bzw. den Prozentsatz der Organisationen, die sowohl in den Metadaten der Föderation als auch in eduGAIN veröffentlicht sind. Die hierzu analysierten Metadatendateien wurden direkt von den Föderationen heruntergeladen.

	DFN-AAI	Renater (Frankreich)	SWAMID (Schweden)	UK Access Management Federation	eduGAIN
Größe der Datei:	2.8 MB	4.1 MB	6.5 MB	14 MB	3.1 MB
Anzahl der Einträge:	280	625	975	1915	413
Einträge, die auch in eduGAIN verfügbar sind:	30 (∼11%)	10 (∼2%)	387 (∼40%)	324 (∼17%)	–

Tabelle 3.1.: Vergleich von Metadaten-Dateien nationaler Föderationen mit eduGAIN

Bei dem Vergleich fällt auf, dass nur ein kleiner Bruchteil der Organisationen aus der DFN-AAI und Renater sich dazu entschlossen hat, ihre Metadaten auch innerhalb von eduGAIN verfügbar zu machen. Das liegt daran, dass diese Föderationen die Teilnahme an eduGAIN für die Organisationen als „opt-in“ anbieten, also standardmäßig deaktiviert haben. Eine Organisation, die keinen konkreten Bedarf an der Teilnahme hat und nicht selbst aktiv wird, wird so nicht in die eduGAIN-Metadaten übernommen. Die Föderationen aus Schweden und dem Vereinigten Königreich haben die Aufnahme in eduGAIN hingegen als „opt-out“ und da die meisten Organisationen die Standardeinstellungen verwenden, sind aus diesen Föderationen mehr Teilnehmer auch in eduGAIN vorhanden.

In den aktuell betriebenen nationalen Föderationen herrscht kein Bedarf dazu den Verbindungsaufbau zwischen SP und IDP, wie in Anforderung F1 gefordert, dynamisch zu betreiben. Allerdings würde es auch nicht schaden, dort ein solches System einzusetzen. Die Dienstbetreiber hätten durch das System der dynamischen virtuellen Föderationen zum Beispiel eine bessere Kontrolle darüber, mit welchen anderen Teilnehmern der Föderation ihr

Dienst verwendet wird. In Inter-Föderationen, könnte dieses System aber das oben beschriebene Problem lösen. Dazu würde dann nach Anforderung F2 in einer Inter-Föderation eine TTP, mit daraus abgeleiteten Folgeabhängigkeiten, benötigt werden.

Die Anforderung von (Inter-)Föderationen zur Synchronisation von Metadaten unterscheidet sich in der aktuell betriebenen und oben beschriebenen Methode von der in Anforderung F9 vorgestellten, „on-demand“ Methode. Im Prinzip spricht in einer (Inter-)Föderation aber nichts dagegen, die Metadaten auch nur bei Bedarf auszutauschen. Im Moment werden die unnötig heruntergeladenen Metadaten von Diensten, die nie mit dem eigenen Dienst kommunizieren nur prophylaktisch gespeichert und nicht wirklich benötigt. Eventuell könnte sogar durch das Weglassen unnötiger Metadaten die Verarbeitungszeit, bei regulären Anmeldungen, durch die Reduktion der zu durchsuchenden Metadaten, wenn auch eher geringfügig, verringert werden.

3.2.2.3. Attributsformate

Auch in (Inter-)Föderationen spielen die zur Verfügung stehenden Attribute eine große Rolle. Daher wird auch hier im Folgenden zunächst die Attributsstandardisierung und eine eventuelle Konvertierung betrachtet.

3.2.2.4. Attributsstandardisierung

Innerhalb einer Föderation werden bestimmte Attribute definiert, die ein IDP erzeugen können muss und die ein SP für seine Arbeit abfragen und erwarten kann. Diese Standardisierung von Attributen wird von Föderation zu Föderation unterschiedlich gehandhabt. Einige schreiben bestimmte Attribute vor, andere empfehlen nur einige. Tabelle 3.2 zeigt einen Vergleich ausgewählter Föderationen und ihrer empfohlenen bzw. dokumentierten Attribute, die nicht speziell von der Föderation definiert wurden. Föderationseigene Attribute werden danach betrachtet und verglichen.

Mit acht Attributen werden von eduGAIN am wenigsten Attribute als empfohlen angegeben. Die deutsche DFN-AAI unterscheidet zwischen grundlegenden „✓“ und ergänzenden „(✓)“ Attributen, wobei insgesamt 18 Attribute aufgezählt werden. Zusätzlich werden von der DFN-AAI Schemata für das E-Learning angegeben. Die Schweizer Föderation SWITCH-AAI spezifiziert neben einem eigenen Schema für den Einsatz an Hochschulen auch 18 Attribute, die aber nicht mehr unterteilt werden. Die Schwedische SWAMID schreibt grundsätzlich keine bestimmten Attribute vor, erwähnt in der Dokumentation [JA+13] aber 11 üblicherweise verwendete Attribute. Die Tabelle 3.2 stellt die von den ausgewählten Föderationen dokumentierten allgemeinen Attribute gegenüber und zeigt, aus welchen Schemata die Attribute stammen.

Auffallend ist hier, dass es nur zwei Attribute (`eduPersonScopedAffiliation` und `mail`) gibt, die in allen vier Föderationen erwähnt werden. Es zeigt sich auch, dass nicht alle von eduGAIN spezifizierten Attribute auch in der Dokumentation der hier betrachteten teilnehmenden nationalen Föderationen erwähnt werden. Zwar kann ein IDP auch Attribute herausgeben, die nicht explizit von der Föderation gefordert oder dort dokumentiert sind, es kann dadurch allerdings nur schlecht abgeschätzt werden, wie gut eine Zusammenarbeit

Attributsname	Definiert in	eduGAIN	DFN-AAI	SWITCHaai	SWAMID
eduPersonAssurance	eduPerson			✓	
eduPersonEntitlement	eduPerson		✓	✓	✓
eduPersonPrincipalName	eduPerson		✓	✓	✓
eduPersonNickname	eduPerson			✓	
eduPersonOrgDN	eduPerson		(✓)	✓	
eduPersonOrgUnitDN	eduPerson		(✓)	✓	
eduPersonPrimaryOrgUnitDN	eduPerson			✓	
eduPersonAffiliation	eduPerson	✓	(✓)	✓	
eduPersonPrimaryAffiliation	eduPerson			✓	
eduPersonScopedAffiliation	eduPerson	✓	✓	✓	✓
eduPersonScopedPrimaryAffliliation	eduPerson				✓
eduPersonTargetedID	eduPerson		✓	✓	
eduPersonPersistentID	eduPerson				✓
common name (cn)	person	✓	(✓)		✓
surname (sn)	person		✓		✓
telephoneNumber	person		(✓)	✓	
organisationalUnitName (ou)	orgPerson		(✓)		
organisationName (o)	orgPerson		(✓)		✓
postalAddress	orgPerson		(✓)	✓	
displayName	inetOrgPerson	✓	(✓)		
givenName	inetOrgPerson	✓	(✓)		✓
homePhone	inetOrgPerson			✓	
homePostalAddress	inetOrgPerson			✓	
mail	inetOrgPerson	✓	✓	✓	✓
mobile	inetOrgPerson			✓	
preferredLanguage	inetOrgPerson			✓	
uid	inetOrgPerson		(✓)		
userCertificate	inetOrgPerson		(✓)		
schacHomeOrganization	SCHAC	✓			✓
schacHomeOrganizationType	SCHAC	✓			

Tabelle 3.2.: Vergleich der in verschiedenen Föderationen definierten und empfohlenen Attribute. Die DFN-AAI unterscheidet zwischen „grundlegenden“ Attributen, welche durch ein „✓“ und „ergänzenden“ Attributen, die als „(✓)“ markiert sind. [Lin11] [Gie+08] [LS+12] [JA+13]

zwischen verschiedenen Föderationen funktionieren würde. Das zeigt, dass zwischen den Föderationen nicht abgesprochen ist, welche Attribute auch föderationsübergreifend verwendet werden. Daher möchte ein Administrator eventuell schnell überprüfen können, ob ein anderer Dienst mit dem eigenen verwendbar ist. Da diese Funktion nicht für den grundsätzlichen Betrieb einer AAI notwendig ist, wird dies als optionale Anforderung eingeordnet.

F24: Kompatiblitätstest

Ein Benutzer oder Administrator soll schnell überprüfen können, ob ein SP mit einem IDP verwendbar ist, das heißt, ob der IDP die vom SP geforderten Attribute erbringen kann.

Zusätzlich zu den föderationsübergreifend spezifizierten Attributsschemata, die in Tabelle 3.2 verglichen wurden, spezifizieren viele Föderationen eigene Schemata. Die Tabellen 3.5, 3.3 und 3.6 listen föderationsspezifische Attribute auf. Zudem ist in Tabelle 3.4 ein von TERENA [Mas11] herausgegebenes und von eduGAIN, DFN-AAI und anderen Föderationen unterstütztes und eingesetztes Schema für das E-Learning angegeben. Die Tabellen sind chronologisch nach der Reihenfolge der ersten Veröffentlichung sortiert. Eine erste Version des Schemas der SWITCH-AAI (Tabelle 3.3) wurde 2002 veröffentlicht. 2006 wurde das SCHAC-Schema (Tabelle 3.4) veröffentlicht. Gefolgt von dem DFN-AAI Schema (Tabelle 3.5) im Jahre 2008 und der nordEdu (Tabelle 3.6) 2010.

Attributsname	**Beschreibung**
swissEduPersonCardUID	ID einer Karte
swissEduPersonDateOfBirth	Geburtsdatum basierend auf [RFC3339]
swissEduPersonGender	Geschlecht
swissEduPersonHomeOrganization	Heimatorganisation innerhalb von SWITCHaai
swissEduPersonHomeOrganizationType	Typ der Heimatorganisation
swissEduPersonMatriculationNumber	Matrikelnummer
swissEduPersonStaffCategory	Angestelltenkategorie
swissEduPersonStudyBranch1	Fächergruppe (z.B. Geistes und Sozialwissenschaften)
swissEduPersonStudyBranch2	Studienbereich (z.B. Naturwissenschaften)
swissEduPersonStudyBranch3	Studienfach (z.B. Mikrotechnik)
swissEduPersonStudyLevel	Studienart (Bachelor, Master, etc.)
swissEduPersonUniqueID	Institutsübergreifende ID

Tabelle 3.3.: Von der SWITCH-AAI spezifizierte Attribute [LS+12]

Es ist schnell ersichtlich, dass in diesen vier Schemata einige Eigenschaften über die Föderationen hinweg doppelt definiert sind. Da diese Schemata über fast 10 Jahre hinweg entstanden und weiterentwickelt worden sind, ist das nicht weiter verwunderlich. Zum Beispiel definieren die SWITCH-AAI, SCHAC und NORDUnet ein eigenes Feld für das Geburtsdatum `swissEduPersonDateOfBirth`, `schacDateOfBirth` und `norEduPersonBirthDate`. Neben den unterschiedlichen Namen werden auch unterschiedliche Formate für das Datum verwendet. NORDUnet und SCHAC verwenden das Format „YYYYMMDD“ wohingegen die SWITCH-AAI ein komplizierteres Format abgeleitet von RFC 3339 [RFC3339] verwendet.

Auch auffällig ist, dass sowohl die SWITCH-AAI als auch die DFN-AAI Attribute der Form `{dfn,swiss}EduPersonStudyBranch{1,2,3}` verwenden um die Studienart und Studienfach

Attributsname	Beschreibung
schacMotherTonge	Muttersprache
schacGender	Geschlecht
schacDateOfBirth	Geburtstag als YYYYMMDD
schacYearOfBirth	Geburtsjahr als YYYY
schacPlaceOfBirth	Geburtsort
schacCountryOfCitizenship	Staatsbürgerschaft
schacSn1	Erster Vorname
schacSn2	Zweiter Vorname
schacPersonalTitle	Titel des Benutzers
schacHomeOrganization	Domain Name der Heimatorganisation
schacHomeOrganizationType	Typ der Heimatorganisation
schacCountryOfResidence	Aufenthaltsland des Benutzers
schacUserPresenceID	Zur Angabe von Präsenzprotokollen wie xmpp, sip, skype
schacPersonalPosition	Position des Benutzers in der Organisation
schacPersonalUniqueCode	Eindeutige ID eines Benutzers (z.B. Matrikelnummer)
schacPersonalUniqueID	Amtliche Identifikationsnummer (z.B. Steuernummer)
schacExpiryDate	Verfallsdatum des gesammten Datensatzes
schacUserPrivateAttribute	Gibt an welche Attribute eines Benutzers privat seien sollen
schacUserStatus	Gibt den Status des Benutzers an
schacProjectMembership	Projektzugehörtigkeit
schacProjectSpecificRole	Rollen innerhalb eines Projektes

Tabelle 3.4.: Durch das SCHAC-Schema definierte Attribute [Mas11]

Attributsname	Beschreibung
dfnEduPersonCostCenter	Kostenstelle
dfnEduPersonStudyBranch1	Fächergruppe
dfnEduPersonStudyBranch2	Studienbereich
dfnEduPersonStudyBranch3	Studienfach
dfnEduPersonFieldOfStudy	Studienfachbezeichnung
dfnEduPersonFinalDegree	Studienabschluss
dfnEduPersonTypeOfStudy	Studienart
dfnEduPersonTermsOfStudy	Fachsemester
dfnEduPersonBranchAndDegree	Studienfach und Abschluss
dfnEduPersonBranchAndType	Studienfach und Studienfachart
dfnEduPersonFeaturesOfStudy	Gesamtstudiumsinformation

Tabelle 3.5.: Von der DFN-AAI spezifizierte Attribute [Deu+08]

Attributsname	Beschreibung
norEduOrgAcronym	Abkürzung für die Institution
norEduOrgNIN	Amtliche Identifikationsnummer
norEduOrgSchemaVersion	Versionsnummer des verwendeten Schemas
norEduOrgUniqueIdentifier	ID der Institution
norEduOrgUnitUniqueIdentifier	ID für eine Organisationseinheit
norEduPersonBirthDate	Geburtsdatum als YYYMMDD
norEduPersonLegalName	Vollständiger formaler Name
norEduPersonLIN	ID Nummer (z.B. Matrikelnummer/Angestelltennummer)

Tabelle 3.6.: Von NORDUnet spezifizierte Attribute [UNI14]

entsprechend einer hierarchischen Darstellung anzugeben. SCHAC gibt eine solche Aufschlüsselung nicht her und das NORDUnet Schema verwendet an dieser Stelle IDs, um Benutzer bestimmten Institutionen zuordnen zu können.

Die Föderationen benötigen also die Fähigkeit, auch eigene Schemata für die Kommunikation spezialisierter Dienste innerhalb der Föderation definieren und standardisieren zu können. Diese Anforderung, die eventuell auch bei Communities oder Projekten anfällt, wird einer Föderation zugeordnet, da deren Planung und Umsetzung nur dann sinnvoll ist, wenn viele Organisationen spezielle Schemata adaptieren. Diese Anforderung ist nicht direkt für die Authentifizierung von Benutzern wichtig, wird aber für den Betrieb mancher SPs benötigt und wird daher als erweitert kategorisiert.

> **F25: Schemaerweiterungen**
>
> Zur kontinuierlichen Weiterentwicklung des Systems muss es erweiterbar sein. Dazu gehört, dass die verwendeten Attributsschemata durch eigene Schemata erweitert werden können.

Zur Identifikation von Studenten oder Mitarbeitern werden zudem in den Schemata eigene Attribute für Matrikelnummern bzw. Personalnummer definiert. Diese Nummern sind natürlich nur innerhalb einer Organisationsdomäne eindeutig und müssen bei organisationsübergreifender Verwendung mit einem entsprechenden Scope verwendet werden. Die SWITCH-AAI definiert hierfür eine `swissEduPersonMatriculationNumber`, das SCHAC Schema einen `schacPersonalUniqueCode` und die NORDUnet eine `norEduPersonLIN`. Die DFN-AAI spezifiziert kein solches Attribut, sondern sieht direkt die Verwendung des SCHAC Attributes vor. Auch bei anderen Attributen gibt es äquivalente Überschneidungen.

Alle hier dargestellten Attribute lassen sich über eindeutige Objekt IDs (OIDs) durch den Internet Registrierungsbaum adressieren. Auch die von den Organisationen selbst definierten Schemata sind in einem der Organisation zugeordneten Teilbaum registriert. Zum Beispiel ist das Attribut `norEduPersonBirthDate` mit der OID `1.3.6.1.4.1.2428.90.1.3` unter dem Teilbaum „ISO(1). ISO-Organisation(3). Department-of-Defense(6). Internet(1). Private(4). Enterprise(1). UNINETT(2428). FEIDE(90). FEIDE-LDAP-Attributes(1)“ registriert.

Attributskonvertierung Wie im vorhergehenden Abschnitt beschrieben wurde, sind in unterschiedlichen Föderationen unterschiedliche Attribute empfohlen bzw. standardisiert. Um die Zusammenarbeit über Föderationen hinweg einfacher zu machen, muss ein Weg gefunden werden, die Attribute geeignet ineinander umzuwandeln. Aktuell gibt es dazu kein System, dass bei Inter-Föderationen wie eduGAIN eingesetzt wird. Daher wird anhand von Beispielen aus dem vorhergehenden Abschnitt überprüft, ob die durch Anforderung F16 beschriebenen Konvertierungsoperationen hier auch anwendbar wären:

- **Umbenennung von Attributen**: Für den Fall, dass es zwei Attribute mit unterschiedlicher OID und Namen gibt, die aber die gleiche Formatierung und Bedeutung haben. Ein Beispiel dafür ist die Konvertierung von `norEduPersonBirthDate` zu `schac DateOfBirth`, die beide das Geburtsdatum in der Form YYYYMMDD speichern.

- **Abbilden von Attributen**: Diese Konvertierung wird benötigt, wenn es keine 1:1 Übersetzung gibt. Beispiele hierfür sind die Attribute `dfnEduPersonStudyBranch1` und `swissEduPersonStudyBranch1`. Das Attribut des DFN beschreibt die Fächergruppe durch eine nummerische ID, die der vom Statistischen Bundesamt vergebenen ID entspricht. Die SWITCH-AAI verwendet auch eine nummerische ID, die aber auf einer anderen Liste basiert. In der DFN-AAI bedeutet der Wert „01“ Sprach- und Kulturwissenschaften [Sta13], wohingegen bei der SWITCH-AAI der Wert „1“ Geistes- und Sozialwissenschaften [LS+12] bedeutet und keine 1:1-Entsprechung gefunden werden kann.

- **Kombination von Attributen**: In manchen Fällen kann durch das Kombinieren zweier oder mehrerer Attribute ein anderes Attribut konstruiert werden. Zum Beispiel kann der `common name` und der `surname` aneinandergehängt werden, um einen `norEduPersonLegalName` zu erstellen. Eine weitere Anwendung dieser Konvertierung ist das Anhängen eines Scopes an ein Attribut, um aus dem Attribut `eduPerson Affiliation` und der Domain des Benutzers (zum Beispiel „lmu.de“) eine `eduPerson ScopedAffiliation` zu erstellen.

- **Extraktion von Teilen eines Attributs**: Diese Konvertierung verwendet nur einen Teil eines anderen Attributes weiter. Hier kann zum Beispiel aus einem Scoped-Attribut wie dem `eduPersonScopedAffiliation` die Domain der Heimatorganisation für das Attribut `schacHomeOrganization` ausgelesen werden.

- **Komplexe Konvertierungen**: Für manche Konvertierungen ist ein Konvertierungsschritt eventuell nicht ausreichend und die Konvertierung von dem Ausgangsattribut in das Zielattribut benötigt mehrere Schritte, die in einem Skript zusammengefasst werden. Eine solche Konvertierung wird benötigt, falls bei der Konvertierung Berechnungen stattfinden müssen. Zum Beispiel bei der Konvertierung von Datumsformaten wie dem Unix-Timestamp zu einer Form YYYYMMDD.

Die in Anforderung F16 vorgeschlagenen Operationen können also bei realen Schemata eingesetzt werden, um dort Konvertierungen vorzunehmen. Über die zur Attributsidentifizierung verwendeten Identifier können die nötigen Konvertierungsregeln wie in Anforderung F14 beschrieben werden. Einen zentralen Dienst zur Verteilung der Metadaten gibt es bei Föderationen ebenfalls schon, welcher prinzipiell dazu erweitert werden könnte, auch Konvertierungsregeln entsprechend Anforderung F13 zu verwalten. Für den in Anforderung F17 geforderten automatischen Abgleich von Attributen und Konvertierungsregeln könnte das zentrale System sowie die Systeme auf IDP und SP Seite ebenfalls angepasst werden.

3.2.2.5. Rollenbasierte Anforderungen

Die spezifischen Anforderungen von IDPs, SPs und Benutzern, die in den vorhergehenden Abschnitten (3.1.4 und 3.2.1.4) betrachtet wurden, unterscheiden sich nicht weiter von den Anforderungen dieser Teilnehmer in (Inter-)Föderationen, weshalb hier nicht explizit erneut auf deren Anforderungen eingegangen wird.

3.2.2.6. Fazit

Da der internationale Bedarf zur Zusammenarbeit an Projekten und einer Authentifizierungs- und Authorisierungs-Infrastruktur ständig wächst, haben sich im Rahmen von eduGAIN europäische und internationale Föderationen zu einer Inter-Föderation zusammengeschlossen. Die Inter-Föderation eduGAIN wurde schon in Kapitel 1 und Abschnitt 2.2.2 kurz beschrieben.

Das von eduGAIN gewählte System zur Aggregierung der Dienste aus den nationalen Föderationen ist allerdings auch nicht problemfrei. Durch die deutlich größere Menge an potentiellen IDPs und SPs ist die Menge der in der Föderation definierten Attribute reduziert worden. In der Praxis reduziert sich die Schnittmenge der von den IDPs lieferbaren Attribute mit steigender Anzahl von IDPs. Innerhalb größerer Föderation ist daher die minimale Menge an Attributen, die von jedem IDP erbracht werden können muss, geringer. Das schränkt aber wiederum die generelle Nutzbarkeit der Föderation ein, da SPs, die mehr als die wenigen standardisierten Attribute verwenden möchten, keine Garantie haben, dass ein IDP diese auch liefern kann. Daraus ergibt sich die Anforderung, dass ein SP auch die Attribute, die er benötigt, abfragen kann. Diese Anforderung ist grundlegend für den Betrieb von SPs innerhalb von Föderationen und würde durch das Konzept dynamischer virtueller Föderationen erfüllt.

Prinzipiell profitieren also auch (Inter-)Föderationen von einem System, das auf den Prinzipien und Anforderungen von dynamischen virtuellen Föderationen basiert. Es ist daher also denkbar, dass eine Implementierung auch in diesen Umgebungen eingesetzt werden kann.

3.3. Zusammenfassung der Anforderungen

Sowohl für (Inter-)Föderationen als auch für Communities und Projekte könnte das Prinzip dynamischer virtueller Föderationen also angewendet werden. Die dazu gesammelten Anforderungen werden der Übersicht wegen hier tabellarisch aufgeführt. Die Bewertung der Wichtigkeit der einzelnen Anforderungen wird hierbei durch die Zahlen eins bis drei ausgedrückt. Dabei entspricht eine „1“ einer grundlegenden Anforderung, eine „2“ einer erweiterten Anforderung und eine „3“ einer optionalen Anforderung.

Nr.	Kurzbeschreibung	Relevanz
F1	**Dynamischer Verbindungsaufbau**: Das Hinzufügen und Entfernen von Organisationen in dynamischen virtuellen Föderationen soll nur bei Bedarf und automatisiert innerhalb weniger Minuten erfolgen. (Abschnitt 3.1.1)	1
F2	**Trusted Third Party**: Um ohne viel Konfigurationsaufwand beliebig dynamische virtuelle Föderationen aufzubauen, soll eine Trusted Third Party als zentrale Vermittlungsinstanz verwendet werden. (Abschnitt 3.1.1)	1
F3	**TTP-Anmeldung**: Das System benötigt die Möglichkeit zur Registrierung, Modifizierung und Löschung von Accounts, sowie der Authentifizierung mit gültigen Accounts an der TTP. (Abschnitt 3.1.1)	1

F4	**Zuständigkeit**: Es muss durch eine geeignete Methode festgestellt werden können, wer für einen Dienst verantwortlich bzw. berechtigt ist, den Dienst in das System aufzunehmen und Änderungen vorzunehmen. (Abschnitt 3.1.1)	1
F5	**Zugriffsberechtigungen**: Das System benötigt die Möglichkeit, Lese- und Schreib-Berechtigungen für die registrierten Systeme zu vergeben, so dass nur autorisierte Personen die Eigenschaften eines Dienstes verändern können. (Abschnitt 3.1.1)	1
F6	**Dienstverwaltung**: Zuständige Administratoren müssen die bei der TTP gespeicherten Eigenschaften ihrer Dienste verwalten können. Dazu gehört neben dem Modifizieren von Eigenschaften auch die Übertragung der Verantwortung für einen Dienst an einen Nachfolger sowie das komplette Entfernen eines Dienstes. (Abschnitt 3.1.1)	1
F7	**Kompatibilität**: Damit Installation und Updates den Betrieb innerhalb einer Föderation nicht stören, müssen Updates abwärtskompatibel und die verwendeten Systeme idealerweise schon vorhanden bzw. bekannt sein. (Abschnitt 3.1.1)	2
F8	**Fehlertoleranz**: Die eingesetzten Systeme müssen fehlertolerant und darauf ausgelegt sein, dass andere Organisationen eventuell leicht andere Implementierungen verwenden. (Abschnitt 3.1.1)	2
F9	**Metadatensynchronisation**: Die Synchronisation von Metadaten soll erst bei Bedarf stattfinden, für den Nutzer transparent und mit einer maximalen Verzögerung von wenigen Minuten stattfinden. (Abschnitt 3.1.1)	1
F10	**Benutzerinitiierter Verbindungsaufbau**: Für den Fall, dass es zwischen dem SP, den der Benutzer verwenden möchte und seinem IDP noch keine Vertrauensbeziehung gibt, soll der Benutzer in der Lage sein, diesen Verbindungsaufbau selbst zu initiieren und im Erfolgsfall innerhalb von wenigen Minuten den Dienst benutzen können. (Abschnitt 3.1.1)	1
F11	**IDP Auswahl**: Der Benutzer muss in der Lage sein ausgehend von der Loginseite des SPs die TTP und seinen dort vertretenen IDP selbst anzugeben oder auszuwählen. (Abschnitt 3.1.1)	1
F12	**Attributsidentifikation**: Zur eindeutigen Identifizierung der verwendeten Attribute soll ein standardisierter, eindeutiger Schlüssel verwendet werden. (Abschnitt 3.1.3.1)	1
F13	**Konv.-Regelverwaltung**: Es soll einen zentralen Dienst zum Suchen, Anzeigen, Modifizieren und Erstellen von Konvertierungsregeln geben. (Abschnitt 3.1.3.2)	1
F14	**Konv.-Regelbeschreibung**: Die Konvertierungsregeln sollen über Metainformationen verfügen, die mindestens beschreiben, welches Attribut aus welchen Attributen abgeleitet wird. Diese Informationen sollen die Suche nach der am besten passenden Regel erleichtern. (Abschnitt 3.1.3.2)	1
F15	**Konv.-Regelbewertung**: Die Konvertierungsregeln sollen bewertet werden können, um deren Qualität in Hinsicht auf Vollständigkeit, Korrektheit und Zuverlässigkeit abschätzen zu können. (Abschnitt 3.1.3.2)	2

F16	**Konv.-Regeloperationen**: Das System zur Konvertierung von Attributen muss die Transformationen Umbenennung, Abbildung, Kombination, Extraktion und skriptbasierte Konvertierungen unterstützen. (Abschnitt 3.1.3.2)	1
F17	**Konv.-Regelabgleich**: Das verwendete System muss durch einen automatischen Abgleich von verfügbaren und geforderten Attributen geeignete Konvertierungsregeln finden und anwenden können. (Abschnitt 3.1.3.2)	1
F18	**Fehlerbenachrichtigungen**: Bei durch inkompatible Attribute nicht aufgebauter Verbindungen, sollen die für die Dienste zuständigen Administratoren informiert werden. (Abschnitt 3.1.3.2)	3
F19	**Attribute Release Policies**: Der IDP soll über ARPs festlegen können, welche Attribute er an einen SP herausgibt. (Abschnitt 3.1.4.1)	2
F20	**Access Control Lists**: Der IDP und SP sollen über ACLs festlegen können, welche Dienste mit ihrem Dienst verwendet werden können. (Abschnitt 3.1.4.1)	2
F21	**Attributsfreigabe**: Der Schutz der Benutzerinformationen muss auch vom Benutzer kontrollierbar sein, indem er die Möglichkeit hat zu überprüfen, welche Informationen an einen Dienst übermittelt werden. (Abschnitt 3.1.4.3)	2
F22	**Auditierbarkeit**: Es muss sowohl auf IDP- als auch SP-Seite die Möglichkeit geben einfach nachzuvollziehen, welches System, welche Attribute abgefragt, bzw. welche Aussagen über Attribute und Authentifizierung getroffen hat. (Abschnitt 3.2.1.1)	3
F23	**Testsysteme**: Bevor ein neuer Teilnehmer einer produktiven Föderation beitreten kann, soll innerhalb einer Testumgebung festgestellt werden können, ob die Software richtig konfiguriert ist. (Abschnitt 3.2.2.1)	3
F24	**Kompatiblitätstests**: Ein Benutzer oder Administrator soll schnell überprüfen können, ob ein SP mit einem IDP verwendbar ist, das heißt, ob der IDP die vom SP geforderten Attribute erbringen kann. (Abschnitt 3.2.2.4)	3
F25	**Schemaerweiterungen**: Zur kontinuierlichen Weiterentwicklung des Systems muss es erweiterbar sein. Dazu gehört, dass die verwendeten Attributsschemata durch eigene Schemata erweitert werden können. (Abschnitt 3.2.2.4)	2

Tabelle 3.7.: Funktionale Anforderungen

Nr.	Kurzbeschreibung	Relevanz
NF1	**Wartbarkeit**: Zur Minimierung des Supportaufwands soll das Einrichten und Aktualisieren der benötigten Softwarekomponenten möglichst einfach sein. (Abschnitt 3.1.1)	2
NF2	**Kosten**: Zusätzliche Kosten durch Lizenzen oder ähnliches sind zu vermeiden, um möglichst vielen SPs und IDPs einen einfachen Zugang zu ermöglichen. (Abschnitt 3.1.1)	1
NF3	**Zukunftssicherheit**: Die Kernfunktionalität der Software muss auch in zukünftigen Funktionen erhalten und abwärtskompatibel sein. (Abschnitt 3.1.1)	2
NF4	**Open Source**: Eine als Open Source verfügbare Software erleichtert die Erweiterung und Analyse der Software an den benötigten Stellen. (Abschnitt 3.1.1)	1
NF5	**Anwendbarkeit**: Durch eine flexible Abstraktion von bestehenden Speicher- und Authentifizierungssystemen, soll das System in möglichst vielen Fällen einsetzbar sein. (Abschnitt 3.1.1)	2
NF6	**Protokoll**: Damit die Erweiterung einem definierten Ablauf folgt und auch mit anderen Softwareumgebungen implementiert werden kann, wird ein Protokoll benötigt. (Abschnitt 3.1.1)	1
NF7	**Verlässlichkeit der SPs**: IDP und SP sollen an einen Verhaltenskodex, der unter anderem regelt, wie die von einem IDP übermittelten Benutzerdaten verwendet werden und welche Datenschutzbestimmungen eingehalten werden müssen, gebunden sein. (Abschnitt 3.1.4.1)	1
NF8	**Auth.-Verantwortung**: Übertragen der Verantwortung bei der Verwaltung von Authentifizierungsinformationen wie Passwörtern und des damit zusammenhängenden Supports von dem Dienst auf den IDP. (Abschnitt 3.1.4.2)	1
NF9	**Datenaktualität**: Entsprechend der Stellung des IDP kann dieser verlässlichere und aktuellere Benutzerinformationen liefern. Im Idealfall beinhaltet dies auch eine Zusicherung der Aktualität der Daten. (Abschnittt 3.1.4.2)	2
NF10	**User-Experience-Konsistenz**: Der Ablauf der Authentifizierung über FIM soll bei einer Erweiterung für den Benutzer möglichst wenige Änderungen an dem ihm bekannten Ablauf bewirken. (Abschnitt 3.1.4.3)	3
NF11	**Zugang zu einer TTP**: Die Teilnehmer einer Community bzw. eines Projekts sollen Zugang zu einer TTP haben, durch die sie für ihre Zwecke eine dynamische virtuelle Föderation erstellen können. (Abschnitt 3.2.1.1)	1
NF12	**Homeless IDP**: Es soll für Benutzer ohne eigenen Heimat-IDP eine Möglichkeit geben trotzdem die Dienste der Föderation verwenden zu können. (Abschnitt 3.2.1.1)	3

Tabelle 3.8.: Nichtfunktionale Anforderungen

4. Vergleich zu bestehenden Lösungen

Dieses Kapitel wird untersucht, welche Software-Lösung bzw. Implementierung am besten zu den im vorhergehenden Kapitel 3 aufgestellten Anforderungen passt. Dazu werden nur SAML-Implementierungen betrachtet. Für das in Abschnitt 2.4 beschriebene OpenID Connect gibt es zwar Implementierungen, die auf der Webseite `http://openid.net/developers/libraries` referenziert werden, allerdings ist OpenID Connect nicht direkt für die Problemlösung geeignet. Zum einen gibt es keine Kompatiblität zwischen OpenID Connect und

SAML bzw. Shibboleth, welches das Ergebnis unterstützen muss, da dieses hauptsächlich an Universitäten und Forschungseinrichtungen verwendet wird. Zum anderen ist die Auswahl, mit alleine 13 von OpenID referenzierten, Implementierungen so groß, dass deren Filterung und Analyse für eine reine Vorstellung in dieser Arbeit nicht zielführend wäre.

Im ersten Abschnitt wird der vielversprechendste – da schon weit verbreitet eingesetzte – Kandidat betrachtet. Als Alternative dazu wird in Abschnitt 4.2 simpleSAMLphp analysiert.

Neben diesen FIM-Implementierungen werden zusätzlich Metadatenverwaltungssysteme betrachtet. Diese Systeme können dazu verwendet werden, Metadaten zu registrieren und auszutauschen. Die betrachteten Systeme sind JANUS (Abschnitt 4.3) und PEER (Abschnitt 4.4).

Zusätzlich werden Systeme analysiert, deren Aufgabe es ist, für Benutzer die Auswahl des IDPs einfacher und komfortablert zu machen. Hierzu werden, neben dem Shibboleth Discovery Service (Abschnitt 4.1.3), DiscoJuice (Abschnitt 4.5) und AccountChooser (Abschnitt 4.6) betrachtet.

Bei dem Vergleich kann eine Anforderung entweder vollständig oder nicht erfüllt sein. Eventuell kann sie auch durch ein zusätzliches Plug-In erfüllt sein, welches dann entsprechend angegeben wird. Am Ende des Kapitels, in Abschnitt 4.7, werden die Ergebnisse der Analyse zusammengefasst.

4.1. Shibboleth

In diesem Abschnitt wird die Open-Source-Implementierung von SAML, Shibboleth, mit den im vorhergehenden Kapitel 3 aufgestellten Anforderungen verglichen. Shibboleth (`https://shibboleth.net`) umfasst eine Implementierung des kompletten SAML-Protokolls und ist im föderierten Identitymanagement im Hochschulumfeld häufig eingesetzt bzw. de facto Standard. Shibboleth enthält verschiedene Teilmodule, die voneinander unabhängig eingesetzt werden können. Zum einen eine auf Java und Apache Tomcat basierende Implementierung eines Identity Provider. Zum anderen eine auf C++ basierte Implementierung für einen Service Provider, der zum Beispiel als Authentifizierungsmodul im Apache Web-Server verwendet werden kann. Neben diesen Basiskomponenten IDP und SP, implementiert Shibboleth sowohl einen Embedded als auch Centralized Discovery Service (DS), der Benutzern dabei hilft, einen Identity Provider aus einer Liste der für einen Service Provider möglichen IDPs auszuwählen [WL12]. Bei dem Vergleich von Shibboleth mit den Anforderungen wird daher angegeben in welchen Teilmodulen bzw. Plug-ins eine Anforderung umgesetzt wird.

4.1.1. Service Provider

Für den Service Provider gibt es verschiedene Versionen, die sowohl mit dem Apache als auch dem Microsoft Internet Information Services (IIS) Webserver verwendet werden können [Shi14]. Die Einrichtung eines Shibboleth SPs mit dem Apache Webserver wird zum Bei-

Listing 4.1: Beispiel für die Datei `shibboleth2.xml`

```
<SPConfig [...]>
  <!-- The ApplicationDefaults element is where most of Shibboleth's SAML bits are
       defined. -->
  <ApplicationDefaults entityID="https://gntb02.srv.lrz.de/shibboleth" [...]>
    <Sessions lifetime="28800" timeout="3600" checkAddress="false"
        relayState="ss:mem" handlerSSL="false">
      <SSO discoveryProtocol="SAMLDS"
          discoveryURL="http://gntb08.srv.lrz.de:8080/discovery/WAYF">
      SAML2 SAML1
      </SSO>
    </Sessions>

    <!-- Example of locally maintained metadata. -->
    <MetadataProvider type="XML"
        file="/etc/shibboleth/gntb01.srv.lrz.de-idp-metadata.xml"/>

    <CredentialResolver type="File" key="/etc/ssl/private/gntb02.srv.lrz.de.key"
        certificate="/etc/ssl/certs/gntb02.srv.lrz.de.crt"/>
  </ApplicationDefaults>

</SPConfig>
```

spiel von der DFN-AAI unter `https://www.aai.dfn.de/dokumentation/service-provider` beschrieben.

Die Konfiguration des Shibboleth SPs erfolgt, wie bei dem IDP auch, über eine XML-Datei. In Listing 4.1 ist eine solche Konfigurationsdatei als gekürztes Beispiel angegeben. In dieser Datei werden grundlegende Einstellungen des SPs angegeben.

Das `ApplicationDefaults` Element enthält zum Beispiel die `entityID`, mit der sich der SP eindeutig identifiziert. Über das `Sessions` Element, wird zum Beispiel angegeben wie lange eine Sitzung maximal gültig ist. Mit dem `SSO` Element kann ein Discovery Service (DS) angegeben werden, der verwendet wird, um den IDP eines Benutzers zu finden. Die Funktion eines DS wird in Abschnitt 4.1.3 näher betrachtet.

Für den Betrieb eines SP muss auch angegeben werden, welche Metadaten von anderen Diensten zur Verfügung stehen. Hierzu dient das `MetadataProvider` Element, welches sowohl eine lokale Datei als auch eine URL als Quelle enthalten kann.

In dem letzten in der Abbildung angegebenen Element `CredentialResolver` wird der Speicherort für den privaten Schlüssel, zum Erstellen von Signaturen, sowie dem Zertifikat mit dem dazugehörigen öffentlichen Schlüssel spezifiziert.

4.1.2. Identity Provider

Der Shibboleth IDP speichert selbst keine Daten, sondern greift zur Datenhaltung und Authentifizierung auf andere Systeme wie MySQL-Datenbanken und LDAP-Systeme zurück. Zur Verwendung externer Datenquellen müssen drei Schritte durchlaufen werden, um die Attribute zu extrahieren, danach eventuell zu konvertieren, zu filtern und verfügbar zu machen. Diese Schritte sind in der Shibboleth Wiki [KW13] dokumentiert.

Die Datei `attribute-resolver.xml`, für die in Listing 4.2 ein Beispiel angegeben ist, definiert die vom IDP herausgegebenen Attribute und deren Quellen.

Im ersten Schritt der **Attribute Definition** wird eine Attribut-ID vergeben und Abhängigkeiten von Datenquellen und eventuell anderen Attributen angegeben. Zudem wird über den **Attribute Encoder** definiert wie das Attribut bei der Veröffentlichung in einer SAML Assertion genannt wird. Hier werden eindeutige OIDs verwendet. Shibboleth stellt außerdem Konvertierungsregeln zur Verfügung, durch die Attribute transformiert werden können. Darunter zum Beispiel das Mapping von einzelnen Werten, Verarbeitung durch reguläre Ausdrücke und Skripte.

In einem zweiten Schritt werden über den **Data Connector** Datenquellen angebunden. Über einen Shibboleth IDP sind folgende Datenquellen verfügbar:

- Der **static data connector** spezifiziert Attribute, die für alle Benutzer gleich sind. Zum Beispiel kann über diese Datenquelle die Zugehörigkeit aller Benutzer zu der Organisation des IDP angegeben werden.
- Mit einem **computed id data connector** wird eine eindeutige ID für einen Benutzer durch das Kombinieren und Hashen der `entityID` des SP, eines Attributes des Benutzers und eines Saltes berechnet.
- Der **stored id data connector** ist eine Alternative zu dem computed id data connector, der eine Datenbank verwendet, um jedem Benutzer eine eindeutige ID pro SP zuzuweisen.
- Über den **relational database data connector** können auf einer relationalen Datenbank SQL-Befehle ausgeführt werden, um ein Attribut zu bestimmen.
- Mit dem **LDAP data connector** können aus einem LDAP-Verzeichnis Attribute geladen werden.

Im dritten Schritt wird in der Datei `attribute-filter.xml` festgelegt, welche Attribute an welchen SP herausgegeben werden dürfen. Die Herausgabe kann dabei allgemeine Freigegeben werden oder für einzelne SPs konfiguriert werden. Eine Kombination ist ebenfallst möglich, sodass einige Attribute an alle SPs veröffentlicht werden aber andere nur für bestimmte SPs zulässig sind. Zudem kann festgelegt werden, welche Werte für ein Attribut verwendet werden können. Ein Beispiel für diese Datei ist in Listing 4.3 gezeigt. Dabei können bestimmte Attribute auch nur für einzelne SPs verfügbar gemacht werden.

4.1.3. Discovery Service

Bei der Installation eines Discovery Services kann zwischen zwei Arten unterschieden werden. Zum einen kann der Embedded Discovery Service direkt beim Service Provider installiert sein, wodurch dieser die Möglichkeit hat entsprechend das Branding mit eigenen Stylesheets und der den Dialog umgebenden Webseite zu beeinflussen. Zum anderen kann ein Centralized Discovery Service von einer Föderation betrieben werden, wodurch ein DS allen SPs innerhalb der Föderation zur Verfügung steht und ihnen den Betrieb abnimmt.

Ist ein Service Provider Mitglied in mehreren Föderationen, muss er entweder den Benutzer

Listing 4.2: Beispiel für die Datei `attribute-resolver.xml`

```
<resolver:AttributeResolver [...]>

  <resolver:AttributeDefinition xsi:type="ad:Simple" id="email"
      sourceAttributeID="mail">
    <resolver:Dependency ref="myLDAP" />
    <resolver:AttributeEncoder xsi:type="enc:SAML1String"
    name="urn:mace:dir:attribute-def:mail" />
    <resolver:AttributeEncoder xsi:type="enc:SAML2String"
    name="urn:oid:0.9.2342.19200300.100.1.3" friendlyName="mail" />
  </resolver:AttributeDefinition>

  <resolver:DataConnector id="myLDAP" xsi:type="dc:LDAPDirectory"
    ldapURL="ldap://gntb01.srv.lrz.de"
    baseDN="ou=users,dc=srv,dc=lrz,dc=de"
    principal="uid=shibboleth,ou=users,dc=srv,dc=lrz,dc=de"
    principalCredential="[...]">
    <dc:FilterTemplate>
      <![CDATA[
        (uid=$requestContext.principalName)
      ]]>
    </dc:FilterTemplate>
  </resolver:DataConnector>

</resolver:AttributeResolver>
```

Listing 4.3: Beispiel für die Datei `attribute-filter.xml`

```
<afp:AttributeFilterPolicyGroup id="ShibbolethFilterPolicy" [...]>
  <afp:AttributeFilterPolicy>
    <afp:PolicyRequirementRule xsi:type="basic:ANY"/>

    <afp:AttributeRule attributeID="eduPersonAffiliation">
      <afp:PermitValueRule xsi:type="basic:OR">
        <basic:Rule xsi:type="basic:AttributeValueString" value="faculty"
    ignoreCase="true" />
        <basic:Rule xsi:type="basic:AttributeValueString" value="student"
    ignoreCase="true" />
        <basic:Rule xsi:type="basic:AttributeValueString" value="staff"
    ignoreCase="true" />
      </afp:PermitValueRule>
    </afp:AttributeRule>

    <afp:AttributeRule attributeID="displayName">
      <afp:PermitValueRule xsi:type="basic:ANY" />
    </afp:AttributeRule>

  </afp:AttributeFilterPolicy>
</afp:AttributeFilterPolicyGroup>
```

Listing 4.4: Request auf geschützten Bereich

```
1 GET https://sp.srv.lrz.de/secure-memberonly/ HTTP/1.1
2
3 HTTP/?.? 302 Found
4 Location: http://ds.srv.lrz.de:8080/discovery/WAYF?
5   entityID=https://sp.srv.lrz.de/shibboleth&
6   return=https://sp.srv.lrz.de/Shibboleth.sso/Login?
7     SAMLDS=1&
8     target=ss%3Amem%3A4080a9323ec6ed655f314894d6750960b11dc8e0
```

die zu verwendende Föderation auswählen lassen, bevor er ihn zu dem entsprechenden Föderations Discovery Service weiterleitet, oder einen Discovery Service betreiben, der Identity Provider aus allen Föderationen zusammenfasst.

4.1.3.1. Embedded Discovery Service

Der Embedded Discovery Service besteht aus zwei JavaScript-Dateien, eine zur Konfiguration und eine mit der Programm-Logik, durch deren Einbinden in eine beliebige vom SP gehostete HTML-Datei, ein Element zur Auswahl des IDPs erstellt werden kann.

Der Embedded Discovery Service ruft über den Discovery Feed, die aktuell bei dem SP bekannten IDPs ab und stellt diese als durchsuchbare Liste dar. Die bekannten IDPs werden von dem SP aus den ihm zur Verfügung stehenden Metadaten gewonnen. Zusätzlich wird über Cookies, die letzte Auswahl des Benutzers gespeichert und diese bei einem erneuten Aufruf, zum schnelleren Auffinden, gesondert dargestellt.

4.1.3.2. Centralized Discovery Service

Der Shibboleth Centralized Discovery Service ist im Gegensatz zum Embedded Discovery Service ein eigenständiges Programm, welches in Java implementiert ist und auf einem Apache Webserver mit Tomcat läuft. Der Centralized Discovery Service, im Folgenden auch nur mit Discovery Service gemeint, hat die Metadaten der IDPs bei sich gespeichert und benötigt daher nicht den Discovery Feed der SPs. Konfiguriert wird der DS über XML-Dateien.

Im Folgenden wird der Nachrichtenaustausch entsprechend des SAML IDP Discovery Service Protocols [SAMLIDPDS] beschrieben. Dieses Protokoll wird dazu verwendet, um durch den Benutzer beim Discovery Service einen Identity Provider für den Service Provider zu bestimmen. Dieser Vorgang bereitet, das im Abschnitt 2.3.1 beschriebene, Web-Broser-SSO-Profile vor.

Die erste Phase beginnt damit, dass der Service Provider bei einem Loginversuch, den Benutzer über einen Redirect an den Discovery Service weiterleitet. Die Weiterleitung ist über den HTTP Return Code `302 Found` implementiert, die in dem Attribut `Location` die URL des Discovery Services enthält. Außerdem werden Parameter übermittelt, die angeben, welcher Service Provider die Weiterleitung gestartet hat (`entityID`) und wohin die IDP-Auswahl des Benutzers zurück gemeldet werden soll (`return`). Listing 4.4 zeigt einen Ausschnitt dieses HTTP-Requests.

Listing 4.5: Übermittlung der IDP-Auswahl und Weiterleitung zum IDP

```
GET https://sp.srv.lrz.de/Shibboleth.sso/Login?
SAMLDS=1&
target=ss%3Amem%3A4080a9323ec6ed655f314894d6750960b11dc8e0&
entityID=https://IDP.srv.lrz.de/IDP/shibboleth HTTP/1.1

HTTP/?.? 302 Found
Location: https://gntb01.srv.lrz.de/IDP/profile/SAML2/Redirect/SSO?
  SAMLRequest=fZJdb4IwFIb%2FCum9tICoaYTE6cVM3EaE7WI3S4GjNqkt6ynu49cPxS1uybzue573n
  Cedotirhs9at9NreG0Bnfe%2BVxr56SEhrdXcCJTItdgDclfxfHa34qHPeGONM5VRxJshgnXS6LnR
  2O7B5mAPsoLH9SohO%2Bca5JRutStZ6KM9%2BMp%2B%2BjXQfCfL0ihwOx%2FR0CM4pNlDXhBv0W0
  itTgy%2FxCCS4KsG9qtsZEKzuNrqKWFytE8fyDecpGQlzgKgLGYwZhtRuFwEgYTBiMRbSZ1NYyH4y
  6G2MJSoxPaJSRkwXDARoMwLIKIRzEPxs%2FEy87X3khdS729rqbsQ8hviyIb9Cc9gcXTOV2ApNOjY
  H4qthfKr2PFt2eS%2Fm8Vf6xO6UVJ39jw%2B466XGRGyerDmyll3uYWhIOEBISm%2FcjvH5F%2BAQ
  %3D%3D&
RelayState=ss%3Amem%3A4080a9323ec6ed655f314894d6750960b11dc8e0
```

Nachdem der Benutzer beim Discovery Service seinen Identity Provider ausgewählt hat, wird die Auswahl zurück an den Service Provider gemeldet. Dieser leitet den Benutzer direkt an den ausgewählten IDP weiter und fügt einen base64 codierten SAML-Request hinzu. Diese Weiterleitung ist in Listing 4.5 dargestellt.

Von hier an kann, vorausgesetzt, dass die beteiligten Parteien die Metadaten des jeweils anderen konfiguriert haben, die Kommunikation direkt zwischen Service und Identity Provider stattfinden. Die Bedingung, dass die Metadaten zu diesem Zeitpunkt ausgetauscht worden seien müssen, ist für dynamische virtuelle Föderation ein Problem, da diese vorsehen, erst jetzt die Metadaten auszutauschen.

4.1.4. Vergleich mit Anforderungen

In diesem Abschnitt werden die im vorherigen Kapitel aufgestellten Anforderungen mit den Funktionen von Shibboleth verglichen. Da Shibboleth durch die Implementierung des gesamte SAML Protokolls alle Teilbereiche von FIM abdeckt, können hier Vergleiche mit allen Anforderungen durchgeführt werden.

4.1.4.1. Funktionale Anforderungen

Zunächst werden die funktionalen Anforderungen, wie sie in Tabelle 3.7 zusammengefasst sind, mit Shibboleth verglichen.

F1: Dynamischer Verbindungsaufbau: Das Hinzufügen, Entfernen oder Modifizieren von Diensten und deren Metadaten ist in Shibboleth nicht sehr dynamisch, da es mehrere Stunden dauert, bis die Änderungen durch die Föderation an alle anderen Teilnehmer propagiert wurde. Diese Anforderung wird daher nicht erfüllt.

F2: Trusted Third Party: In Shibboleth gibt es keine zentrale Vermittlungsinstanz, über die SPs und IDPs dynamisch virtuelle Föderationen aufbauen können, weshalb diese Anfor-

derung nicht erfüllt wird. Es gibt allerdings mit dem Centralized Discovery Service einen zentralen Dienst, der in der Föderation zum Auffinden des richtigen IDPs verwendet werden kann.

Da es in Shibboleth keine TTP gibt, werden die Anforderungen F3, F4, F5 und F6, die speziell an eine TTP gestellt werden auch nicht erfüllt.

F7: Kompatibilität: Weiterentwicklungen von Shibboleth achten in der Regel auf eine geeignete Abwärtskompatibilität. Seit dem Sprung von SAML 1.1 auf SAML 2.0 im Jahre 2005 ist das grundsätzlich verwendete Protokoll gleich geblieben. Daher wird die Anforderung der Kompatibilität erfüllt.

F8: Fehlertoleranz: Da Shibboleth zusammen mit anderen SAML-Implementierungen wie simpleSAMLphp eingesetzt werden kann, wird diese Anforderung erfüllt.

F9: Metadatensynchronisation: Für die Metadatensynchronisation existieren Schnittstellen, die von den Föderationen genutzt werden, um automatisiert die Metadaten der Föderation zu installieren. An dieser Stelle fehlt eine Filterung der wirklich benötigten Metadaten und die Möglichkeit nur bei Bedarf Metadaten zu synchronisieren. Daher wird diese Anforderung nicht erfüllt.

F10: Benutzerinitiierter Verbindungsaufbau: Bei Shibboleth ist es nicht vorgesehen, dass ein Benutzer einen Verbindungsaufbau zwischen SP und IDP direkt anstößt. Der Benutzer müsste dazu eine Anfrage an die zuständigen Administratoren stellen, welche dann die in jeder Organisation anders definierten organisatorischen und technischen Schritte zum Verbindungsaufbau zwischen den Parteien durchführen müssten. Dadurch ist diese Anforderung nicht erfüllt.

F11: IDP Auswahl: Sowohl über den Embedded als auch über den Centralized Discovery Service ist der Benutzer bei Shibboleth in der Lage, seinen IDP aus einer Liste von in der Föderation teilnehmenden IDPs, auszuwählen. Diese Anforderung wird erfüllt.

F12: Attributsidentifikation: Shibboleth verwendet in IDP und SP zur Identifizierung von Attributen deren OIDs aus dem Internet-Registrierungsbaum oder Uniform Resource Names (URN). Der Discovery Service dient nur der Auswahl eines IDPs und adressiert nicht direkt Attribute. Daher ist diese Anforderung erfüllt.

F13: Konvertierungsregelverwaltung: Die Verwaltung dieser Regeln wird allerdings nicht vollständig, wie von der Anforderung gefordert, umgesetzt. Die Regeln müssen lokal durch jeden Administrator in der XML-Datei konfiguriert werden und es gibt kein allgemeines Repository, in dem Regeln zur Übersetzung von verschiedenen Attributen angeboten werden. Diese Anforderung ist daher nicht erfüllt.

F14: Konvertierungsregelbeschreibung: Shibboleth hat keine zusätzlichen Beschreibungen für Konvertierungsregeln. Eventuell können Zusatzinformationen als Kommentare in der XML-Datei hinterlegt werden. Die Anforderung wird daher nicht erfüllt.

F15: Konvertierungsregelbewertung: Es gibt für Shibboleth Konvertierungsregeln keine Möglichkeit Bewertungen zu vergeben. Dies ist auch nicht notwendig, da diese Regeln nur lokal verwendet werden. Die Anforderung wird dadurch allderings nicht erfüllt.

F16: Konvertierungsregeloperationen: Die Konvertierung von Attributen unterstützt der Shibboleth IDP mit allen geforderten Konvertierungsoperationen durch die Datei `attribute-resolver.xml`.

F17: Konvertierungsregelabgleich: In Shibboleth können die von SP geforderten und IDP unterstützten Attribute nicht durch automatisch ausgetauschte Konvertierungsregeln angeglichen werden. Diese Anforderung wird daher nicht umgesetzt.

F18: Fehlerbenachrichtigung: Shibboleth benachrichtigt Administratoren nicht direkt, bei wegen unpassenden Attributen fehlgeschlagenen Authentifizierungen. Ein Administrator kann durch die Einsicht von Logdateien diese Funktionalität eventuell nachstellen. Diese Anforderung ist daher nicht erfüllt.

F19: Attribute Release Policies: Shibboleth verfügt durch die Attribute Filter bei der IDP-Implementierung über ARPs. Diese Anforderung wird daher erfüllt.

F20: Access Control Lists: Shibboleth verfügt nicht direkt über ACLs, über die Attribute Filter könnten von einem IDP an einzelne SPs keine Attribute übermittelt werden, der Zugriff wird aber nicht direkt verhindert.

F21: Attributsfreigabe: Durch die weit verbreitete Erweiterung **uApprove** (`https://www.switch.ch/aai/support/tools/uApprove.html`) kann der Benutzer kontrollieren, welche Daten an einen Service Provider weitergegeben werden. Diese Funktion kann also durch ein Plug-In nachgerüstet werden.

F22: Auditierbarkeit: Der IDP führt ein `idp-audit.log`, in dem Metadaten zu Logins und die Art der übermittelte Attribute gespeichert werden. Der SP führt ein `transaction.log`, in dem Metadaten zu neu erstellten Sitzungen gespeichert werden. Diese Anforderung wird daher erfüllt.

F23: Testsysteme: Zum Testen der Shibboleth Installation kann eine separate Föderation verwendet werden. Eine spezielle Option zum Testen aller Funktionen der Software in der aktuellen Installation gibt es jedoch nicht. Diese Anforderung wird daher nicht erfüllt.

F24: Kompatibilitätstests: Es gibt in Shibboleth keine Funktion, die es erlaubt zu testen, ob ein IDP mit einem bestimmten SP verwendbar ist. In der Regel wird versucht dies durch organisatorische Maßnahmen in einer Föderation sicherzustellen. Diese Anforderung wird daher nicht erfüllt.

F25: Schemaerweiterungen: Bei den verwendeten Schemata macht Shibboleth keine Einschränkungen. Daher können auch eigens erstellte Schemata verwendet werden. Diese Anforderung wird erfüllt.

Anforderung	**Erfüllt**			**Teilmodule**
	ja	mit Plug-in	nein	
F1: Dynamischer Verbindungsaufbau			✓	–
F2: Trusted Third Party			✓	–
F3: TTP-Anmeldung			✓	–
F4: Zuständigkeit			✓	–
F5: Zugriffsberechtigungen			✓	–
F6: Dienstverwaltung			✓	–
F7: Kompatibilität	✓			IDP, SP, DS
F8: Fehlertoleranz	✓			IDP, SP, DS
F9: Metadatensynchronisation			✓	–
F10: Benutzerinitiierter Verbindungsaufbau			✓	–
F11: IDP Auswahl	✓			SP, DS
F12: Attributsidentifikation	✓			IDP, SP
F13: Konv.-Regelverwaltung			✓	–
F14: Konv.-Regelbeschreibung			✓	–
F15: Konv.-Regelbewertung			✓	–
F16: Konv.-Regeloperationen	✓			IDP
F17: Konv.-Regelabgleich			✓	–
F18: Fehlerbenachrichtigungen			✓	–
F19: Attribute Release Policies:	✓			IDP
F20: Access Control Lists			✓	IDP, SP
F21: Attributsfreigabe		✓		SP (uApprove)
F22: Auditierbarkeit	✓			IDP, SP
F23: Testsysteme			✓	IDP, SP, DS
F24: Kompatibilitätstests			✓	–
F25: Schemaerweiterungen	✓			IDP, SP

Tabelle 4.1.: Vergleich der funktionalen Anforderungen mit Shibboleth

4.1.4.2. Nichtfunktionale Anforderungen

NF1: Wartbarkeit: Die Wartbarkeit von Shibboleth ist nicht ideal, allerdings existiert eine Menge guter Dokumentationen und verschiedener Anleitungen. Der Aufwand für die Installation und Konfiguration von Shibboleth ist aber auch abhängig, ob IDP, SP oder ein DS betrieben werden sollen, wobei die Einrichtung eines SPs am einfachsten ist. Die Anforderung ist daher erfüllt.

NF2: Kosten: Shibboleth kann ohne zusätzliche Anschaffungs- oder Lizenzkosten verwendet werden, wodurch es die Anforderung „Kosten“ erfüllt.

NF3: Zukunftssicherheit: Durch die hohe Zahl der bereits vorhandenen Installationen und das stabile Grundgerüst von Shibboleth ist diese Anforderung erfüllt.

NF4: Open Source: Shibboleth ist Open-Source und unter der Apache 2.0 Lizenz veröffentlicht, wodurch diese Anforderung erfüllt wird.

NF5: Anwendbarkeit: Durch die modulare Struktur ist Shibboleth in vielen Fällen einsetzbar. Es setzt kein bestimmtes Backend voraus und erfüllt daher diese Anforderung.

NF6: Protokoll: Shibboleth spezifiziert für alle Teile des Systems genaue Protokolle. Diese Anforderung ist daher erfüllt.

NF7: Verlässlichkeit der SPs: Innerhalb von Föderationen wird häufig Shibboleth als die primäre Implementierung von FIM mit SAML verwendet. Innerhalb dieses Systems regeln die Föderationen die Rahmenbedingungen, um die von der Anforderung „Verlässlichkeit“ geforderten Eigenschaften umzusetzen. Neben der Verlässlichkeit wird von Föderationen auch die Anforderung „Datenaktualität“ reglementiert. Daher ist diese Anforderung nicht durch die verwendete Implementierung bestimmbar.

NF8: Authentifizierungsverantwortung: Die Verantwortung für die Authentifizierung kann mit Shibboleth von einem SP an einen IDP übertragen werden. Der SP braucht selber keine Passwörter oder ähnliches zu speichern. Diese Anforderung wird daher erfüllt.

NF9: Datenaktualität: Ähnlich wie die Verlässlichkeit der SPs wird die Datenaktualität der IDPs innerhalb von Föderationen geregelt. Es können zum Beispiel verschiedene Vertrauens-Level für unterschiedliche IDPs definiert werden, die festlegen, wie schnell ein IDP Änderungen an den Daten eines Benutzers nachvollziehen muss. Diese Anforderung ist daher nicht durch die Software an sich bestimmbar.

NF10: User-Experience-Konsistenz: Das User-Interface und der Ablauf bei einer Anmeldung wird durch die Umstellung auf Shibboleth und FIM gegenüber einer klassischen Anmeldung geringfügig geändert. Da Shibboleth aber von einigen Diensten verwendet wird, ist dies für Benutzer nicht unbedingt etwas Neues und der Ablauf nicht unnötig kompliziert. Die Anforderung User-Interface-Konsistenz ist daher erfüllt.

NF11: Zugang zu einer TTP: Da es in der Implementierung von Shibboleth keine TTP gibt, kann diese Anforderung nicht erfüllt werden.

NF12: Homeless IDP: Shibboleth implementiert keinen speziellen Homeless IDP. Ein solcher IDP kann aber von der Föderation oder einer Organisation in der Föderation betrieben werden. Daher kann diese Anforderung erfüllt werden.

Anforderung	Erfüllt	
	ja	nein
NF1: Wartbarkeit	✓	
NF2: Kosten	✓	
NF3: Zukunftssicherheit	✓	
NF4: Open Source	✓	
NF5: Anwendbarkeit	✓	
NF6: Protokoll	✓	
NF7: Verlässlichkeit der SPs	–	–
NF8: Auth.-Verantwortung	✓	
NF9: Datenaktualität	–	–
NF10: User-Experience-Konsistenz	✓	
NF11: Zugang zu einer TTP		✓
NF12: Homeless IDP	✓	

Tabelle 4.2.: Vergleich der nichtfunktionalen Anforderungen mit Shibboleth

4.2. simpleSAMLphp

SimpleSAMLphp (`https://simplesamlphp.org`, Abgekürzt SSP) implementiert SAML und einige andere Standards zum föderierten Identitymanagement wie OpenID und OAuth unter der Leitung von UNINETT. Die Implementierung in PHP erlaubt einen flexiblen Einsatz mit den meisten Webservern. Allerdings unterstützt simplSAMLphp nicht das aktuelle OpenID Connect[sim14a], weshalb hier nur die Funktionalität im Zusammenhang mit SAML 2.0 betrachtet wird. SimpleSAMLphp bietet, wie das zuvor beschriebene Shibboleth, Funktionen für Service Provider und Identity Provider und ist auch über zusätzliche Module erweiterbar. Zunächst wird in den nächsten Abschnitten 4.2.1 und 4.2.2 der SP und IDP der Basisversion von simpleSAMLphp und danach in Abschnitt 4.3 eine Erweiterung zum Metadatenaustausch mit dem Namen JANUS betrachtet.

Die Installation von simpleSAMLphp benötigt nur einen Webserver, der PHP in der Version 5.3.0 oder neuer ausführen kann und über einige PHP-Erweiterungen verfügt [sim14b]. Konfiguriert wird die simpleSAMLphp Installation dann über eine PHP-Datei `config.php` sowie ein Web-Interface. In den folgenden Abschnitten wird auf die Funktionen von simpleSAMLphp Service Provider und Identity Provider näher eingegangen.

4.2.1. Service Provider

Die Konfiguration eines simpleSAMLphp Service Providers wird in [sim14c] beschrieben. Im Folgenden wird ein kurzer Überblick über die zur Konfiguration nötigen Schritte gegeben. Der Service Provider wird über die Datei `authsources.php`, für die in Listing 4.6 ein Beispiel angegeben ist, konfiguriert. Dort wird der SP definiert und der private Schlüssel sowie das Zertifikat mit dem öffentlichen Schlüssel angegeben. Auf ähnlich Art und Weise werden auch

Listing 4.6: Beispiel für die simpleSAMLphp SP Konfiguration von [sim14c].

```
<?php
$config = array(

  /* This is the name of this authentication source, and will be used to access it
    later. */
  'default-sp' => array(
    'saml:SP',
    'privatekey' => 'saml.pem',
    'certificate' => 'saml.crt',
  ),
);
```

die URLs der IDPs über PHP-Strukturen in den Dateien `saml20-idp-remote.php` und `shib13-idp-remote.php` konfiguriert.

Das Auslösen einer Authentifizierung erfolgt dann durch von der simpleSAMLphp gestellten Bibliothek, die mit wenigen Zeilen Code vom Entwickler einer Applikation an einer geeigneten Stelle aufgerufen werden muss.

4.2.2. Identity Provider

In [sim14a] wird beschrieben, wie ein Identity Provider für simpleSAMLphp konfiguriert wird. Hier wird analog zum vorhergehenden Abschnitt eine kurze Übersicht über den IDP gegeben.

Zunächst muss die IDP-Funktionalität in der Datei `config.php` aktiviert werden. Danach muss ein Authentifizierungsmodul ausgewählt werden. SimpleSAMLphp bringt mehrere Module zur Authentifizierung mit, darunter sind zum Beispiel Module für `.htpasswd` Dateien, X.509 Benutzerzertifikate, LDAP und SQL Datenbanken. Die ausgewählte Methode muss, bevor sie verwendet werden kann, aktiviert und eventuell konfiguriert werden.

In einer dem SP ähnlichen Konfiguration müssen dann wieder der zum Signieren benötigte Schlüssel und das dazugehörige Zertifikat angegeben werden. Außerdem muss in einer weiteren Konfiguration angegeben werden, welche Service Provider mit diesem IDP verwendet werden können. Für die Kompatibilität mit anderen SAML-Implementierungen müssen eventuell Filter eingerichtet werden, die zwischen den verschiedenen Formaten für die Benennung von Attributen konvertieren.

4.2.3. Vergleich mit Anforderungen

In diesem Abschnitt werden die im vorherigen Kapitel aufgestellten Anforderungen mit den Funktionen von simpleSAMLphp verglichen. Dabei wird hauptsächlich die Kernfunktionalität von simpleSAMLphp unter Verwendung von SAML 2.0 betrachtet.

4.2.3.1. Funktionale Anforderungen

F1: Dynamischer Verbindungsaufbau: SimpleSAMLphp unterstützt für SAML keinen dynamischen Verbindungsaufbau. Diese Anforderung ist daher nicht erfüllt.

F2: Trusted Third Party: Wie Shibboleth implementiert simpleSAMLphp auch keine TTP, weshalb diese Anforderung nicht erfüllt wird und Anforderungen F3, F4, F5 und F6 dementspechend nicht erfüllt sein können.

F7: Kompatibilität: SimpleSAMLphp ist, mit einigen Konfigurationsänderungen, kompatibel zu Shibboleth. Daher wird diese Anforderung erfüllt.

F8: Fehlertoleranz: Da simpleSAMLphp in großen Installationen eingesetzt wird und kompatibel mit Shibboleth ist, wird diese Anforderung erfüllt.

F9: Metadatensynchronisation: Das Integrieren von Metadaten im Rahmen einer Föderation, die die aktuellen Metadaten unter einer URL veröffentlichen, ist über die Module `cron` und `metarefresh` möglich. Es gibt in simpleSMALphp keinen automatischen Mechnanismus zum dynamischen Metadatenaustausch bei Bedarf. Diese Anforderung wird daher nicht erfüllt.

F10: Benutzerinitiierter Verbindungsaufbau: Für den Aufbau einer Vertrauensbeziehung gibt es in simpleSAMLphp keinen automatischen Mechanismus, der die Anforderung erfüllen würde.

F11: IDP Auswahl: SimpleSAMLphp bringt in dem Download keinen vollständigen Discovery Service mit sich. Durch ein Zusatzmodul wie `discopower` können allerdings ähnlich zu dem Shibboleth Embedded Discovery Service IDPs ausgewählt werden. Alternativ kann das SAML 2 Identity-Provider-Discovery-Protocol verwendet werden, um zum Beispiel über einen Shibboleth CDS den Benutzer den IDP auswählen zu lassen. Diese Anforderung wird daher erfüllt.

F12: Attributsidentifikation: SimpleSAMLphp verwendet zur Identifizierung von Attributen verschiedene Formate. Es kann angegeben welches, Format verwendet werden soll oder simpleSAMLphp versucht anhand der Angaben von SP und IDP abzuleiten welches Format korrekt ist. Diese Anforderung wird daher erfüllt.

F13: Konvertierungsregelverwaltung: SimpleSAMLphp hat keine Möglichkeit zur Konvertierung von Attributen mit Hilfe von Konvertierungsregeln. Daher wird diese Anforderung und die weiteren Anforderungen an die Konvertierung von Attributen F14, F15, F16, F17 nicht erfüllt.

F18: Fehlerbenachrichtigung: SimpleSAMLphp hat keine direkte Benachrichtigung bei Verbindungsfehlern zwischen IDP und SP, wodurch diese Anforderung nicht erfüllt wird.

F19: Attribute Release Policies: SimpleSAMLphp implementiert keine direkten Attribute Release Policies. Eine Option ist es die lokalen Metadaten des SPs zu modifizieren, da bei simpleSAMLphp dort angegeben ist, welche Attribute ein SP anfragt. Da dies nicht sehr praktikabel ist, vor allem da es jedes mal durchgeführt werden muss, wenn die Metadaten des SPs aktualisiert werden, wird diese Anforderung nicht erfüllt.

F20: Access Control Lists: Für simpleSAMLphp gibt es keine ACLs, die die Filterung von IDPs oder SPs erlauben. In diesem Fall müssen die Metadaten des betreffenden IDPs oder SPs manuell durch den Administrator aus dem lokalen Verzeichnis entfernt werden. Daher wird diese Anforderung nicht erfüllt.

F21: Attributsfreigabe: Die Attributsfreigabe kann bei simpleSAMLphp wie bei Shibboleth auch durch ein Plugin (Consent Modul) ermöglicht werden. Diese Anforderung wird dadurch erfüllt.

F22: Auditierbarkeit: SimpleSAMLphp hat die Möglichkeit durch Logfiles und zusätzlich durch das Statistics Modul generierte Informationen Anmeldungen und herausgegeben Attribute nachzuvollziehen. Diese Anforderung wird daher erfüllt.

F23: Testsysteme: Für simpleSAMLphp gibt es ein Testskript, das den Login auf einem SP testet. Zusätzlich bietet das `autotest` Modul die Möglichkeit die konfigurierten Authentifizierungsquellen zu testen. Daher wird diese Anforderung erfüllt.

F24: Kompatibilitätstests: Einen direkten Vergleich von vorhandenen und geforderten Attributen gibt es in simpleSAMLphp ebenfalls nicht. Daher wird diese Anforderung nicht erfüllt.

F25: Schemaerweiterungen: SimpleSAMLphp ermöglicht es auch eigens definierte Attribute und Schemata zu verwenden. Diese Anforderung wird daher auch erfüllt.

4.2.3.2. Nichtfunktionale Anforderungen

NF1: Wartbarkeit: Das Einrichten von simpleSAMLphp ist sehr einfach und gut dokumentiert. Dadurch ist diese Anforderung erfüllt.

NF2: Kosten: Da simpleSAMLphp wie Shibboleth mit einer Open-Source-Lizenz veröffentlicht ist, kann es ohne weitere Installations- oder Lizenz-Kosten verwendet werden. Damit erfüllt simpleSAMLphp diese Anforderung.

Anforderung	Erfüllt			Teilmodule
	ja	mit Plug-in	nein	
F1: Dynamischer Verbindungsaufbau			✓	–
F2: Trusted Third Party			✓	–
F3: TTP-Anmeldung			✓	–
F4: Zuständigkeit			✓	–
F5: Zugriffsberechtigungen			✓	–
F6: Dienstverwaltung			✓	–
F7: Kompatibilität	✓			IDP, SP
F8: Fehlertoleranz	✓			IDP, SP
F9: Metadatensynchronisation			✓	–
F10: Benutzerinitiierter Verbindungsaufbau			✓	–
F11: IDP Auswahl	✓	✓		SP, `discopower`
F12: Attributsidentifikation	✓			IDP, SP
F13: Konv.-Regelverwaltung			✓	–
F14: Konv.-Regelbeschreibung			✓	–
F15: Konv.-Regelbewertung			✓	–
F16: Konv.-Regeloperationen			✓	–
F17: Konv.-Regelabgleich			✓	–
F18: Fehlerbenachrichtigungen			✓	–
F19: Attribute Release Policies:			✓	IDP
F20: Access Control Lists			✓	–
F21: Attributsfreigabe		✓		IDP (`consent` Modul)
F22: Auditierbarkeit	✓			IDP, SP
F23: Testsysteme	✓			IDP, SP (Test-Skript, `autotest` Modul)
F24: Kompatibilitätstests			✓	–
F25: Schemaerweiterungen	✓			IDP, SP

Tabelle 4.3.: Vergleich der funktionalen Anforderungen mit simpleSAMLphp

NF3: Zukunftssicherheit: SimpleSAMLphp wird aktiv weiterentwickelt und hat eine große Community. Dadurch ist die Zukunftssicherheit sichergestellt und diese Anforderung erfüllt.

NF4: Open Source: SimpleSAMLphp ist unter der GNU Lesser General Public License (LGPL) Version 2.1 veröffentlicht. Damit erfüllt es diese Anforderung.

NF5: Anwendbarkeit: Durch eine Vielzahl an Modulen und die Unterstützung von nicht nur SAML als Authentifizierungs- und Autorisierungsprotokoll ist simpleSAMLphp in vielen Fällen anwendbar. Dadurch erfüllt es auch diese Anforderung.

NF6: Protokoll: SimpleSAMLphp implementiert SAML und einige weitere FIM-Protokolle, wodurch es diese Anforderung erfüllt.

NF7: Verlässlichkeit der SPs: Die Verlässlichkeit eines SPs kann nicht direkt durch die verwendete SAML Implementierung gewährleistet sein. Es muss also innerhalb der Föderation oder explizit zwischen SP und IDP eine Vereinbarung über die Verwendung von Benutzerattributen geben. Diese Anforderung ist daher unabhängig von der verwendeten Implementierung und kann nicht an dieser festgemacht werden.

NF8: Authentifizierungsverantwortung: Durch die Verwendung von simpleSAMLphp kann der SP die Verantwortung für die Authentifizierung an den IDP übertragen. Diese Anforderung wird daher erfüllt.

NF9: Datenaktualität: Wenn die Aktualität der Daten innerhalb der Föderation bzw. zwischen IDP und SP vereinbart ist, können über simpleSAMLphp aktuelle Daten bezogen werden. Unter dieser Voraussetzung ist diese Anforderung erfüllt.

NF10: User-Experience-Konsistenz: Durch die Verwendung von PHP kann simpleSAMLphp in viele, am besten PHP basierte, Webanwendungen integriert werden. Dadurch kann die UI individuell an die Anwendung angepasst werden, wodurch diese Anforderung erfüllt wird.

NF11: Zugang zu einer TTP: Ohne Erweiterungen gibt es von simpleSAMLphp keine TTP, daher kann diese Anforderung nicht erfüllt werden.

NF12: Homeless IDP: Ähnlich wie bei Shibboleth gibt es keine explizite Implementierung für einen Homeless IDP, allerdings kann innerhalb einer Föderation ein IDP die Registrierung von Benutzern erlauben und kontrollieren und so als IDP für Benutzer ohne Heimat-IDP agieren. Dadurch wird diese Anforderung erfüllt.

Anforderung	**Erfüllt**	
	ja	nein
NF1: Wartbarkeit	✓	
NF2: Kosten	✓	
NF3: Zukunftssicherheit	✓	
NF4: Open Source	✓	
NF5: Anwendbarkeit	✓	
NF6: Protokoll	✓	
NF7: Verlässlichkeit der SPs	–	–
NF8: Auth.-Verantwortung	✓	
NF9: Datenaktualität	✓	
NF10: User-Experience-Konsistenz	✓	
NF11: Zugang zu einer TTP		✓
NF12: Homeless IDP	✓	

Tabelle 4.4.: Vergleich der nichtfunktionalen Anforderungen mit simpleSAMLphp

4.3. JANUS

Eine Erweiterung von simpleSAMLphp mit dem Namen JANUS (`https://github.com/janus-ssp/janus`), bietet einen Web-Dienst zur Verwaltung von Metadaten. Mit dieser Erweiterung können ähnlich einer Föderation, die Mitglieder die Metadaten ihrer Systeme an einem zentralen Punkt selber verwalten und Beziehungen zu anderen IDPs und SPs herstellen. JANUS wird seit 2009 entwickelt. Ursprünglich durch WAYF (`www.wayf.dk`) und seit September 2013 durch SURFnet (`www.surfnet.nl`) [Chr13]. Im Folgenden werden die wichtigsten Funktionen aus der Dokumentation [Lie14] und dem WAYF Youtube-Channel (`https://www.youtube.com/user/wayfsekretariat/videos`), auf dem die Funktionen von JANUS präsentiert werden, beschrieben.

4.3.1. Übersicht

Die Metadatenverwaltung wird bei JANUS durch ein Web-Portal konfiguriert. Um die Metadatenverwaltung nutzen zu können, muss der Dienstbetreiber sich zunächst bei dem System registrieren und anmelden. Authentifiziert werden Benutzer bei der Anmeldung in JANUS durch simpleSAMLphp, es können hier also alle Methoden, die simpleSAMLphp unterstützt, verwendet werden. In den Demonstrationen wird ein E-Mail-Token-basiertes System für den Login verwendet. Neue Accounts können über ein Benutzerverwaltungsinterface erstellt werden. Die Berechtigungen der Benutzer können über dasselbe Interface rollenbasiert verwaltet werden. Die Benutzer können dann für ihre SPs bzw. IDPs die Verbindungen untereinander, Metadaten sowie Attribute Release Policies verwalten.

IDPs und SPs werden in JANUS als Verbindungspunkte (Connections) bezeichnet. Für jeden dieser Punkte existiert ein Revisionssystem, bei dem Änderungen an den Einstellungen nachverfolgt werden können. Mögliche Konfigurationen umfassen das Sperren einzelner SPs/IDPs, bei IDPs dem Deaktivieren des Zustimmungsdialogs, der einen Benutzer normalerweise über die übertragenen Attribute informiert, sowie die Verwaltung seiner Metadaten. Die Metadaten können dabei direkt über das Webinterface durch ein Formular modifiziert und auch exportiert oder importiert werden. SPs können zusätzlich über Attribute Release Policies angeben, welche Attribute ein IDP einem SP bereitstellen muss.

Um über Änderungen bei anderen SPs/IDPs informiert zu werden, gibt es ein Subscription-System, mit dem ein Benutzer einzelne Dienste auswählen kann, bei denen er über Änderungen informiert werden möchte.

Das System unterstützt außerdem Certificate Rollovers, also das Austauschen von Zertifikaten. Dabei werden eine Zeit lang mehrere Zertifikate in den Metadaten verwendet.

Zusätzlich bietet JANUS eine REST API an, so dass nicht für alle Aufgaben das Web-Interface verwendet werden muss.

4.3.2. Vergleich mit Anforderungen

Da JANUS auf simpleSAMLphp aufbaut und dieses um ein System zum Metadatenaustausch erweitert, werden im folgenden Vergleich nicht alle Anforderungen abgeglichen sondern le-

diglich diese, die simpleSAMLphp nicht erfüllt. Alle Anforderungen, die simpleSAMLphp erfüllt, können auch in Verbindung mit JANUS erfüllt werden.

F1: Dynamischer Verbindungsaufbau: Über JANUS können Verbindungen zwischen simpleSAMLphp IDPs und SPs aufgebaut werden. Die Anforderung beschreibt allerdings auch, dass dieser Verbindungsaufbau dynamisch bei Bedarf stattfinden soll, was bei JANUS nicht der Fall ist. Daher wird diese Anforderung auch durch das Hinzufügen von JANUS nicht erfüllt.

F2: Trusted Third Party: JANUS tritt in dem System als eine Art Trusted Third Party auf, in dem es die Metadaten der verschiedenen Teilnehmer verwaltet und deren Austausch ermöglicht. Daher können Administratoren, durch diesen Dienst unterstützt, einfacher virtuelle Föderationen aufbauen. Allerdings sind diese nicht dynamisch, da sie nicht direkt durch den Benutzer initiiert werden können. Die grundlegende Anforderung an eine TTP wird trotzdem erfüllt.

F3: TTP-Anmeldung: JANUS verwendet für die Anmeldung die von simpleSAMLphp zur Verfügung gestellten Authentifizierungsbackends. Die registrierten Accounts können dann über das Webinterface verwaltet werden. Daher ist diese Anforderung erfüllt.

F4: Zuständigkeit: Die Berechtigungen eines Benutzers können in JANUS durch einen Administrator festgelegt werden. Dadurch können einzelne Personen dazu berechtigt werden Eigenschaften eines Dienstes zu modifizieren. Diese Methode ist nicht ideal in der Skalierbarkeit, da ein Benutzer auf die Freischaltung durch einen Administrator warten muss, erfüllt aber diese Anforderung.

F5: Zugriffsberechtigungen: Über die Benutzerverwaltung in dem JANUS Webinterface können die Berechtigungen der einzelnen Benutzer modifiziert werden. Dadurch wird diese Anforderung erfüllt.

F6: Dienstverwaltung: JANUS bietet über das Webinterface und eine API die Möglichkeit die Dienste zu verwalten. Durch das Anpassen von Berechtigungen kann die Kontrolle über einen Dienst auch an einen Nachfolger übergeben werden. Diese Anforderung wird daher ebenfalls erfüllt.

F9: Metadatensynchronisation: JANUS ist ein Metadatenregistrierungs-Modul, für simpleSAMLphp mit dem Metadaten eines Dienstes bei einer zentralen Instanz registriert werden können. Ein dynamischer, durch einen Benutzer angestoßener und schnell beendeter, Austausch von Metadaten ist allerdings nicht vorgesehen, weshalb diese Anforderung nicht erfüllt wird.

F10: Benutzerinitiierter Verbindungsaufbau: Ein Benutzer kann auch durch JANUS nicht den Aufbau eines Vertrauensverhältnis zwischen IDP und SP anstoßen, wodurch diese Anforderung nicht erfüllt wird.

F13: Konvertierungsregelverwaltung: Es gibt in JANUS keine Option zur Verwaltung von Konvertierungsregeln, wodurch diese und die direkt damit in Verbindung stehenden Anforderungen F14, F15, F16, F17 nicht erfüllt werden.

F18: Fehlerbenachrichtigung: JANUS bietet einen Subscription-Dienst, durch den sich Administratoren über Änderungen und Fehler informieren lassen können. Daher wird diese Anforderung erfüllt.

F19: Attribute Release Policies: Zu der Verwaltung der Metadaten gehört in JANUS auch das Anlegen von Attribute Release Policies. Dabei kann ein IDP pro SP definieren, welche Attribute er herausgeben möchte. Diese Anforderung wird daher erfüllt.

F20: Access Control Lists: Durch die Verwaltung in JANUS können explizit Verbindungen zwischen Systemen aufgesetzt werden, wodurch der Zugriff auf die Dienste explizit gesteuert werden kann. Diese Anforderung wird daher erfüllt.

In der folgenden Zusammenfassung der funktionalen Anforderungen in Tabelle 4.5 sind aus dem Vergleich mit simpleSAMLphp in Abschnitt 4.2.3 übernommene Anforderungen mit * markiert.

Die nichtfunktionalen Anforderungen werden von simpleSAMLphp bereits weitestgehend erfüllt. Einige davon werden hier auch erneut betrachtet.

NF1: Wartbarkeit: Zur Einrichtung von JANUS gibt es eine Dokumentation, die das Vorgehen beschreibt. Daher wird die Anforderung erfüllt.

NF2: Kosten: JANUS ist als Erweiterung für simpleSAMLphp ebenfalls kostenfrei benutzbar, wodurch diese Anforderung erfüllt bleibt.

NF3: Zukunftssicherheit: Die Zukunftssicherheit von JANUS ist etwas ungewiss, da die Dokumentation schon länger nicht mehr aktualisiert wurde, es aber Weiterentwicklungen im Git-Repository und Bestrebungen gibt JANUS in einem größeren Umfeld einzusetzen. Die Zukunftssicherheit wird für die nahe bis mittlere Zukunft daher vermutlich sichergestellt sein.

NF4: Open Source: JANUS ist unter der MIT-License veröffentlicht, wodurch es diese Anforderung erfüllt.

Anforderung	Erfüllt	
	ja	nein
F1: Dynamischer Verbindungsaufbau		✓
F2: Trusted Third Party	✓	
F3: TTP-Anmeldung	✓	
F4: Zuständigkeit	✓	
F5: Zugriffsberechtigungen	✓	
F6: Dienstverwaltung	✓	
F7: Kompatibilität	*	
F8: Fehlertoleranz	*	
F9: Metadatensynchronisation		✓
F10: Benutzerinitiierter Verbindungsaufbau		✓
F11: IDP Auswahl	*	
F12: Attributsidentifikation	*	
F13: Konv.-Regelverwaltung		✓
F14: Konv.-Regelbeschreibung		✓
F15: Konv.-Regelbewertung		✓
F16: Konv.-Regeloperationen		✓
F17: Konv.-Regelabgleich		✓
F18: Fehlerbenachrichtigungen	✓	
F19: Attribute Release Policies:	✓	
F20: Access Control Lists	✓	
F21: Attributsfreigabe	*	
F22: Auditierbarkeit	*	
F23: Testsysteme	*	
F24: Kompatibilitätstests		*
F25: Schemaerweiterungen	*	

Tabelle 4.5.: Vergleich der funktionalen Anforderungen mit JANUS

NF5: Anwendbarkeit: Die Verwendung von JANUS als Metadatenregistrierungs-Software ist sowohl für große Inter-Föderationen als auch für kleine Projekt-/Community-Föderationen denkbar. Dadurch erfüllt es auch diese Anforderung.

NF6: Protokoll: JANUS spezifiziert kein neues Protokoll, bietet aber über eine dokumentierte API die Möglichkeit mit dem System, auch aus anderer Software heraus, zu kommunizieren. Daher ist diese Anforderung erfüllt.

NF11 Zugang zu einer TTP zum Zugang zu einer TTP hat sich geändert. Ein potentieller Teilnehmer kann unter von der Föderation festzulegenden Regeln Zugang zu JANUS bekommen, wodurch diese Anforderung erfüllt wird.

4.4. PEER

Die Public Endpoint Entities Registry (PEER, `https://github.com/Yaco-Sistemas/peer`) ist ein Projekt, mit dem Ziel ein globales Registrierungssystem für SPs und IDPs zu erstellen, damit diese dort ihre Metadaten hinterlegen können. Das System ist dabei nicht nur auf

Anforderung	Erfüllt	
	ja	nein
NF1: Wartbarkeit	✓	
NF2: Kosten	✓	
NF3: Zukunftssicherheit	✓	
NF4: Open Source	✓	
NF5: Anwendbarkeit	✓	
NF6: Protokoll	✓	
NF7: Verlässlichkeit der SPs	–	–
NF8: Auth.-Verantwortung	*	
NF9: Datenaktualität	*	
NF10: User-Experience-Konsistenz	*	
NF11: Zugang zu einer TTP	✓	
NF12: Homeless IDP	*	

Tabelle 4.6.: Vergleich der nichtfunktionalen Anforderungen mit JANUS

SAML beschränkt, sondern unterstützt auch OpenID und IMI. PEER wird von der TERENA (Trans-European Research and Education Networking Association) entwickelt.

4.4.1. Übersicht

PEER ist ein reines Verwaltungssystem, in dem die Metadaten von allen angemeldeten Diensten durchsuchbar gemacht werden. Dokumentiert ist das in Python implementierte System auf der Plattform `https://pythonhosted.org/peer`. Die Metadaten werden bei PEER in einem lokalen Git-Repository und Benutzer- und Domain-Informationen in einer relationalen Datenbank wie PostgreSQL, MySQL oder SQLite verwaltet. Das Frontend wird über einen Webserver mit Web Server Gateway Interface (WSGI) Ünterstüzung, wie zum Beispiel dem Apache-Webserver dargestellt.

PEER hat außerdem eine detaillierte Berechtigungsverwaltung, bei der über XPath-Ausdrücke bestimmte Berechtigungen festgelegt werden können. Diese bestimmen dann, welche Teile einer Metadatendatei von einem Nutzer bearbeitet werden können. Für die Bearbeitung der Metadaten wird das von simpleSAMLphp (`http://samlmetajs.simplesamlphp.org`) entwickelte jQuery Plugin SAMLmetaJS verwendet. Zusätzlich können eigene Metadaten-Validatoren auf der Serverseite eingesetzt werden um die Metadaten zu überprüfen und auf Fehler hinzuweisen. Außerdem können Metadaten automatisch aktualisiert werden, wenn eine entsprechende URL angegeben wurde.

Eine wichtige Funktion ist es, zu überprüfen, ob der Benutzer dazu berechtigt ist, eine bestimmte Domain in dem System anzumelden. Hierzu muss auf der Domain eine bestimmte Datei erstellt werden, die PEER dann versucht abzufragen. Alternativ kann die Validierung über einen TXT DNS-Eintrag erfolgen. So kann ein Administrator nachweisen, dass er Kontrolle über die Domain hat.

Eine weitere fortgeschrittene Funktion ist die Integration mit der Monitoring-Plattform Nagios, an die Events gesendet werden können, wenn Einträge angelegt, modifiziert oder gelöscht werden.

4.4.2. Vergleich mit Anforderungen

PEER implementiert keine spezielle Authentifizierungs-Infrastruktur, sondern analog zu JANUS nur eine zentrale Verwaltungsmöglichkeit für Metadaten. Daher ist es offensichtlich, dass einige der in dieser Arbeit erstellten Anforderungen nicht erfüllt werden.

4.4.2.1. Funktionale Anforderungen

F1: Dynamischer Verbindungsaufbau: Da PEER nur zur Verwaltung von Metadaten gedacht ist, und keine Interaktion für End-Benutzer vorsieht, wird diese Anforderung nicht erfüllt.

F2: Trusted Third Party: PEER stellt eine Art von TTP dar, da es die Verwaltung und Verteilung von Metadaten über möglichst viele Dienste hinweg übernimmt. Daher wird diese Anforderung erfüllt.

F3: TTP-Anmeldung: In PEER gibt es diverse Authentifizierungsmechanismen und eine Accountverwaltung, wodurch diese Anforderung erfüllt wird.

F4: Zuständigkeit: Durch die Überprüfung der Kontrolle eines Administrators über die Domain, durch einen speziellen HTTP-Request oder DNS Einstellungen, wird sichergestellt, dass der Benutzer berechtigt ist, die Konfiguration des Systems zu übernehmen. Damit ist diese Anforderung erfüllt.

F5: Zugriffsberechtigungen: Über ein detailliertes System können Berechtigungen zum Modifizieren von Metadaten durch XPATH-Ausdrücke auf Ebene von einzelnen Elementen gemacht werden. Dadurch und eine generelle Benutzerverwaltung, wird diese Anforderung erfüllt.

F6: Dienstverwaltung: Es besteht die Möglichkeit die Zuständigkeit für einen Dienst an einen anderen Benutzer weiterzugeben. Das erfüllt diese Anforderung.

F7: Kompatibilität: Da das PEER-System nicht direkt in eine Föderation eingebunden wird, hat es keine Auswirkungen auf ansonsten verwendete Software. Diese Anforderung findet hier also keine Anwendung.

F8: Fehlertoleranz: Ähnlich wie die vorhergehende Anforderung zur Kompatibilität, findet die Fehlertoleranz hier keine weitere Anwendung, da das System unabhängig von den in einer Föderation eingesetzten Systemen ist.

F9: Metadatensynchronisation: PEER bietet nur eine Verwaltung von Metadaten, keine Synchronisation zwischen den Endsystemen, die die Metadaten verwenden. Daher wird diese Anforderung nicht erfüllt.

Die Anforderungen F10: Benutzerinitiierter Verbindungsaufbau, F11: IDP Auswahl und F12: Attributsidentifikation finden hier ebenfalls keine Anwendung, da das System nicht versucht diese Probleme zu lösen. Das Gleiche gilt für die Konvertierung von Attributen, wie sie in den Anforderungen F13: Konvertierungsregelverwaltung, F14: Konvertierungsregelbeschreibung, F15: Konvertierungsregelbewertung, F16: Konvertierungsregeloperationen und F17: Konvertierungsregelabgleich beschrieben ist.

F18: Fehlerbenachrichtigung: PEER beinhaltet ein Benachrichtigungssystem, das Benutzer über nicht länger gültige Metadaten informiert. Zusätzlich können über die Integration in Nagios Benachrichtigungen zu verschiedensten Ereignissen erzeugt werden. Damit wird diese Anforderung erfüllt.

Für eine reine Metadatenverwaltung sind die Anforderungen F19: Attribute Release Policies, F20: Access Control Lists und F21: Attributsfreigabe nicht relevant, und können daher nicht bestimmt werden.

F22: Auditierbarkeit: Durch die Nagios-Integration sind Änderungen in dem System einfach zu überwachen, wodurch diese Anforderung erfüllt wird.

Die letzten funktionalen Anforderungen F23: Testsysteme und F25: Schemaerweiterungen finden hier auch keine Anwendung.

Anforderung	**Erfüllt**	
	ja	nein
F1: Dynamischer Verbindungsaufbau		✓
F2: Trusted Third Party	✓	
F3: TTP-Anmeldung	✓	
F4: Zuständigkeit	✓	
F5: Zugriffsberechtigungen	✓	
F6: Dienstverwaltung	✓	
F9: Metadatensynchronisation		✓
F18: Fehlerbenachrichtigungen		✓
F22: Auditierbarkeit	✓	

Tabelle 4.7.: Vergleich der funktionalen Anforderungen mit PEER

4.4.2.2. Nichtfunktionale Anforderungen

NF1: Wartbarkeit: Da bei dem SP oder IDP keine zusätzlichen Softwarekomponenten installiert werden müssen, gibt es auch keinen zusätzlichen Wartungsaufwand und diese Anforderung ist erfüllt.

NF2: Kosten: Die Verwendung von PEER ist ohne weitere Anschaffungs- oder Lizenz-Kosten möglich. Damit wird diese Anforderung erfüllt.

NF3: Zukunftssicherheit: Durch die Unterstützung von TERENA ist eine große Vereinigung an der Entwicklung beteiligt, allerdings gibt es das Beta-System schon seit mehreren Jahren und keinen dokumentierten Produktiveinsatz. Daher ist diese Anforderung nicht erfüllt.

NF4: Open Source: PEER ist unter der Simplified BSD License veröffentlicht und erfüllt dadurch diese Anforderung.

NF5: Anwendbarkeit: Ziel von PEER ist es in Verbindung mit Föderationen eingesetzt zu werden, um dort Metadaten verwalten zu können. Es kann sowohl in großen Inter-Föderationen als auch in kleineren Projekt-Föderationen eingesetzt werden und erfüllt damit diese Anforderung.

Die Anforderung NF6: Protokoll ist für PEER nicht anwendbar, da für PEER keine direkte Interaktion mit anderen System vorgesehen ist und daher kein Protokoll definiert wird. Ebenso sind die Anwendungen NF7: Verlässlichkeit der SPs, NF8: Authentifizierungsverantwortung, NF9: Datenaktualität und NF10: User-Experience-Konsistenz nicht relevant.

NF11: Zugang zu einer TTP: Unter `http://beta.terena-peer.yaco.es` gibt es eine Testumgebung, an der jeder zum Test Metadaten registrieren kann. Einen Produktiveinsatz gibt es zur Zeit noch nicht. Damit wird diese Anforderung erfüllt.

NF12: Homeless IDP: Kann hier nicht Angewendet werden.

Anforderung	**Erfüllt**	
	ja	nein
NF1: Wartbarkeit	✓	
NF2: Kosten	✓	
NF3: Zukunftssicherheit		✓
NF4: Open Source	✓	
NF5: Anwendbarkeit	✓	
NF11: Zugang zu einer TTP		✓

Tabelle 4.8.: Vergleich der nichtfunktionalen Anforderungen mit Shibboleth

4.5. DiscoJuice

DiscoJuice (`http://discojuice.org`) ist ein Identity Provider Discovery Service, der von Andreas Åkre Solberg bei UNINETT entwickelt wird. Ziel ist es die Auswahl des IDPs besonders benutzerfreundlich zu gestalten und auf Seite des Service Providers weniger Aufwand

Listing 4.7: Beispiel für die DiscoJuice Konfiguration von [Sol13].

```
<script type="text/javascript"
    src="https://cdn.discojuice.org/discojuice-stable.min.js"></script>
<link rel="stylesheet" type="text/css"
    href="https://cdn.discojuice.org/css/discojuice.css" />
<script type="text/javascript">
  DiscoJuice.Hosted.setup({
    "target": "a.signin",
    "title": "Example showcase service",
    "spentityid": "https://bridge.uninett.no/saml2/entityid",
    "responseurl": "http://bridge.uninett.no/response.html",
    "redirectURL": "http://bridge.uninett.no/login?idp=",
    "feeds": ["edugain", "kalmar", "feide"]
  });
</script>
```

zum Implementieren des Discovery Services zu verursachen. Dokumentiert ist DiscoJuice auf seiner Website, die die Grundlage für den folgenden Überblick bildet [Sol13].

4.5.1. Übersicht

Prinzipiell funktioniert der Dienst ähnlich wie die Shibboleth Discovery Services. Er zeigt dem Benutzer eine Liste von IDPs, aus der dieser seinen IDP auswählen kann. Implementiert ist die bei dem Service Provider zu installierende Komponente allerdings in JavaScript. Dadurch, dass nicht ein extra Tomcat-Server betrieben werden muss, ist es deutlich einfacher DiscoJuice zu integrieren als einen Shibboleth DS. Die Konfiguration erfolgt über JSON-Dateien oder Datenstrukturen, die in die Webseite eingebunden werden, auf der der Login stattfinden soll.

DiscoJuice verwendet sowohl lokalen Speicher im Browser in der Form von Cookies als auch eine Datenbank auf der DiscoJuice Seite.

Erweiterte Funktionen von DiscoJuice sind zum Beispiel, dass basierend auf der IP-Adresse eines Benutzers über Geo-Lokalisierung versucht wird die initiale Liste der angezeigten IDPs auf das Ursprungsland einzuschränken.

Je nachdem in welcher Umgebung man DiscoJuice einsetzen möchte, gibt es grundsätzlich vier mögliche Optionen für eine DiscoJuice Installation:

1. **Embedded DiscoJuice**: Der Embedded DiscoJuice Discovery Service kann auf jeder Seite eingesetzt werden, die einen Login-Button hat. Dieser Button wird über etwas zusätzliches JavaScript konfiguriert und öffnet beim Klicken ein Menü, über das der Benutzer seinen IDP auswählen kann. Ein Beispiel für diese Konfiguration ist in Listing 4.7 abgebildet.

   Zunächst wird hier im ersten Script-Tag das DiscoJuice JavaScript eingebunden und danach ein CSS-Stylesheet, welches die Darstellung des DiscoJuice Overlays steuert. Danach wird in dem zweiten Script-Tag DiscoJuice initialisiert. Dazu wird über das `target` angegeben, welches Element auf der Webseite den Loginvorgang startet. Zudem über den `title` eine Überschrift für das DiscoJuice Popup festgelegt. Die Attribute `spentityid` und `responseurl` werden für das IDP Discovery Protokoll benötigt und

geben die EntityID des SPs und eine Seite, über die die Identität des SPs bestätigt werden kann an. Über das Attribut `redirectURL` wird angegeben wohin der Benutzer nach der Auswahl seines IDPs weitergeleitet werden soll. Letztlich kann über `feeds` angegeben werden, welche Föderationen für die IDP-Auswahl betrachtet werden sollen.

In dem Fall, dass es keinen Login-Button gibt, können die Installationsoptionen Service Provider Discovery oder Global Discovery Service eingesetzt werden.

2. **Service Provider Discovery**: Mit dieser Methode kann ein lokaler Discovery Service bei einem SP betrieben werden. Diese Installationsoption benötigt nur eine Webseite, die bei dem SP gehostet ist und an die der SP den normalen IDP Discovery Request sendet. Dies ist ähnlich zu dem Ablauf über den Shibboleth DS, der im Abschnitt 4.1.3 beschrieben wurde. Diese Installation ist daher eine sehr einfach zu installierende Alternative zu dem Shibboleth Embedded DS. Die Konfiguration erfolgt ähnlich wie bei Installationsoption 1, direkt in der Webseite über JavaScript. Bei dieser Methode gibt es allerdings eine zusätzliche Option, bei der durch eine ACL der Zugriff auf den DS eingeschränkt werden kann.

3. **Identity Federation Discovery**: Diese Methode ist nahezu identisch zu der vorhergehenden. Der einzige Unterschied besteht darin, dass die Installation nicht nur für einen SP vorgesehen ist, sondern als DS für eine Föderation betrieben wird. Diese Installationsoption erfüllt dadurch die selben Funktionen wie der Shibboleth DS, der in Abschnitt 4.1.3 beschrieben wurde. Die Installation und Konfiguration ist allerdings bedeutend einfacher, da sie nur das Hosten einer Webseite und das Eintragen einiger Konfigurationsparameter in dieser Seite benötigt.

4. **Global Discovery Service**: Dies ist die einfachste Möglichkeit DiscoJuice als Discovery Service zu verwenden. Sie verwendet einen von DiscoJuice gehosteten Dienst, der die Metadaten aller teilnehmenden Föderationen enthält und für SPs, die in diesen Metadaten vertreten sind, IDP-Discovery durchführt. Dazu muss bei einem SP nur für den Discovery Dienst eine URL von DiscoJuice angegeben werden, die in den Parametern einige wenige Konfigurationsmöglichkeiten wie den Namen des Dienstes und die Auswahl der Föderation aus der IDPs vorgeschlagen werden sollen. Eine solche URL für einen Discovery Service der IDPs aus der eduGAIN Föderation auflistet sieht folgendermaßen aus: `https://cdn.discojuice.org/discovery?b=%7B%22title%22%3A%22eduGAIN%20Discovery%22%2C%22feeds%22%3A%5B%22edugain%22%5D%7D`

4.5.2. Vergleich mit Anforderungen

DiscoJuice spezialisiert sich auf einen kleinen Teil der Aufgaben, die mit den Anforderungen abgedeckt werden. Daher macht es keinen Sinn, DiscoJuice explizit mit allen aufgestellten Anforderungen zu vergleichen. Im Folgenden werden desshalb nur die Menge an relevanten Anforderungen betrachtet.

F7: Kompatibilität: Dadurch, dass für DiscoJuice nur JavaScript in die Webseite eingebunden werden muss, ist das Programm mit sogut wie allen Webanwendungen kompatibel und erfüllt diese Anforderung.

F8: Fehlertoleranz: DiscoJuice ist darauf ausgelegt, in verschiedensten Umgebungen eingesetzt zu werden, daher erfüllt es auch diese Anforderung.

F10: Benutzerinitiierter Verbindungsaufbau: DiscoJuice ermöglicht es dem Benutzer nur seinen IDP aus einer Liste, schon in der Föderation vorhandener IDPs auszuwählen. Daher können keine neuen Verbindungen initiiert werden und diese Anforderung wird nicht erfüllt.

F11: IDP Auswahl: DiscoJuice ermöglicht es auf einfache Art und Weise einen Discovery Service zu betreiben, mit dem der Benutzer seinen IDP auswählen kann. Die Auswahl wird durch Geo-Lokalisierung und Cookies unterstützt. Damit erfüllt DiscoJuice diese Anforderung.

Anforderung	**Erfüllt**	
	ja	nein
F7: Kompatibilität	✓	
F8: Fehlertoleranz	✓	
F10: Benutzerinitiierter Verbindungsaufbau		✓
F11: IDP Auswahl	✓	

Tabelle 4.9.: Vergleich eines Teils der funktionalen Anforderungen mit DiscoJuice

NF1: Wartbarkeit: Dadurch, dass DiscoJuice nur aus einer JavaScript-Komponente besteht und innerhalb der Webseite über JavaScript-Strukturen konfiguriert wird, ist der Betrieb besonders einfach. Zusätzlich gibt es eine Dokumentation, die die meisten Funktionen beschreibt. Damit wird diese Anforderung erfüllt.

NF2: Kosten: Der DiscoJuice Programmcode ist frei verfügbar und auch der von DiscoJuice gehostete zentrale Service, der die Metadaten großer europäischer Föderationen zugänglich macht, kann frei verwendet werden. Daher ist diese Anforderung erfüllt.

NF3: Zukunftssicherheit: Innerhalb des offiziellen Git-Repositories, wurde seit einem Jahr keine Änderungen an DiscoJuice mehr vorgenommen. Es gibt noch eine Mailingliste, auf der äußerst unregelmäßig Fragen gestellt und beantwortet werden. Die Zukunftsentwicklung der Software ist daher unsicher und diese Anforderung nicht erfüllt.

NF4: Open Source: DiscoJuice ist unter der GNU Lesser General Public License Version 3.0 veröffentlicht und erfüllt damit diese Anforderung.

NF5: Anwendbarkeit: Durch die vier Installationsmodi ist DiscoJuice in einer Vielzahl von Fällen anwendbar und erfüllt diese Anforderung.

NF6: Protokoll: DiscoJuice definiert kein eigenes neues Protokoll, sondern verwendet das SAML Discovery Protokoll, um den ausgewählten IDP an den SP zu übermitteln. Für die Kommunikation mit dem Globalen IDP Discovery Service wird ein neues Protokoll spezifiziert. Damit ist diese Anforderung erfüllt.

NF10: User-Experience-Konsistenz: Das User-Interface ist in DiscoJuice besonders darauf abgestimmt worden, dass es flexibel in vorhandene UI-Strukturen integriert werden kann. Daher wird auch diese Anforderung erfüllt.

Anforderung	**Erfüllt**	
	ja	nein
NF1: Wartbarkeit	✓	
NF2: Kosten	✓	
NF3: Zukunftssicherheit		✓
NF4: Open Source	✓	
NF5: Anwendbarkeit	✓	
NF6: Protokoll	✓	
NF10: User-Experience-Konsistenz	✓	

Tabelle 4.10.: Vergleich eines Teils der nichtfunktionalen Anforderungen mit DiscoJuice

4.6. Account Chooser

Account Chooser (AC, `https://www.accountchooser.com`) ist ein, von der OpenID Foundation geführtes, Projekt, welches ein Open-Source Identitäts-Verwaltungs-System basierend auf Webtechnologie konzipiert und implementiert. AC ermöglicht es Benutzern auf einfache Weise einen Überblick über ihre Accounts zu haben, die sie bei verschiedenen Seiten im Web verwenden. Aus Sicht des Benutzers ist der Ablauf ähnlich zu dem von Shibboleth DS und DiscoJuice. AC ist dabei nicht auf eine bestimmte Authentifizierungstechnologie beschränkt, sondern kann flexibel sowohl mit klassischer lokaler Authentifizierung als auch mit OpenID (Connect), SAML oder anderen Protokollen verwendet werden [Ope13].

AC verwendet eine Kombination aus JavaScript und HTML5 LocalStorage, um einfach in ein Webprojekt integriert werden zu können. Auf der Seite des Service Providers muss ein JavaScript-Datei eingebunden werden, die den Benutzer zur Auswahl des Accounts auf die Seite `accountchooser.com` weiterleitet, in deren Kontext ist im Browser des Benutzers über HTML5-LocalStorage eine Liste von Accounts hinterlegt. Das heißt, dass die Account-Informationen nicht auf dem Server von AccountChooser gespeichert werden müssen und nur lokal im Browser vorhanden sind, jede Webseite aber über die Seite darauf zugreifen kann.

In Abbildung 4.1 ist der Ablauf einer Anmeldung mit einem beliebigen externen IDP über AC schematisch dargestellt. In der Abbildung wird davon ausgegangen, dass der vom Benutzer verwendete Account schon bei dem SP registriert und beim AC bekannt ist.

Der Benutzer, der eine geschützte Seite bei einem Service Provider aufruft, wird durch das dort eingefügte JavaScript auf die Account Chooser Seite weitergeleitet. Dort kann er einen

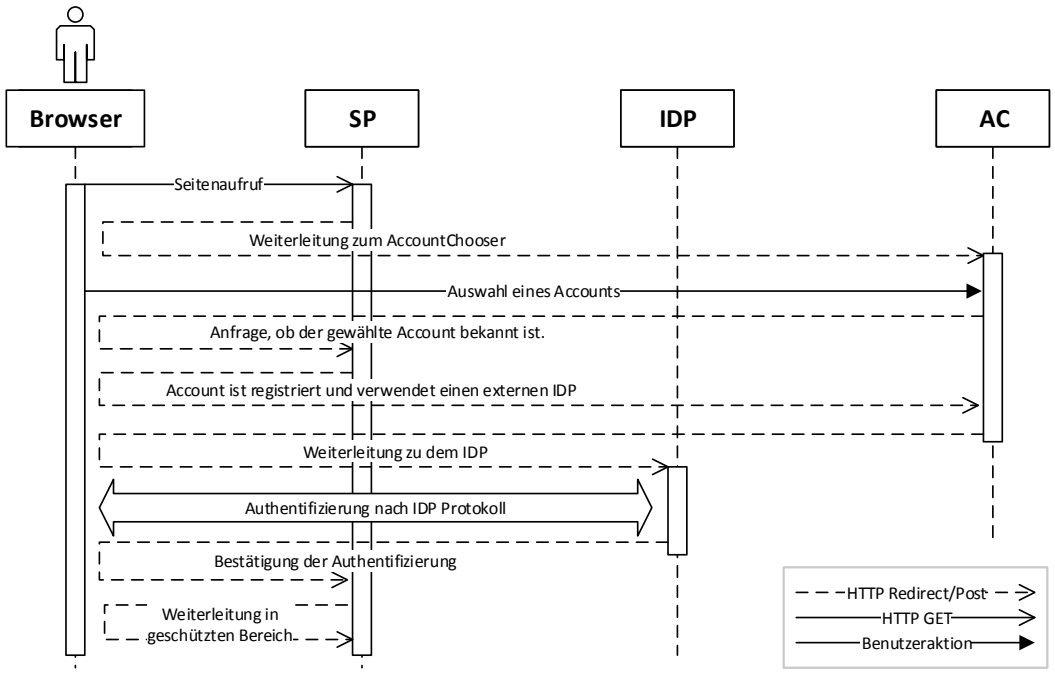

Abbildung 4.1.: Vereinfachtes Schema für die Authentifizierung unter Verwendung von Account Chooster mit einem externen IDP

Account, den er für den Login auf der Seite verwenden möchte, auswählen, woraufhin eine Anfrage an den SP gesendet wird, ob der gewählte Account dort registriert ist. Wenn der Account bekannt ist, wird der Benutzer zu der Login-Seite weitergeleitet. In diesem Fall, ist für den Login ein externer IDP konfiguriert worden und der Benutzer authentifiziert sich dort entsprechend dem IDP-Protokoll. Wenn die Authentifizierung erfolgreich durchgeführt wurde, wird der SP darüber informiert und kann den Benutzer in den geschützten Bereich weiterleiten.

Da diese Implementierung selbst keine Authentifizierung, Metadatensynchronisation oder Konvertierung von Attributen umsetzt, sondern hauptsächlich zur Verwaltung von Account-Informationen gedacht ist, erfüllt sie kaum Anforderungen aus dem in Kapitel 3 erstellten Katalog, weshalb diese jetzt nicht einzeln betrachtet werden. Allerdings ist die Methode zur Auswahl von Accounts in einer webbasierten Umgebung interessant und kann eventuell in Teilen für eine Erweiterung weiterverwendet werden.

4.7. Zusammenfassung

Die Analyse der verschiedenen Software-Lösungen in diesem Kapitel hat, keine Lösung gefunden, welche die aufgestellten Anforderungen vollständig erfüllt. Auch die Kombination von Lösungen wie simpleSAMLphp mit JANUS oder die Verwendung von PEER, lässt wichtige Anforderungen, besonders die automatisierte Konvertierung von Attributen und einen dynamischen Verbindungsaufbau, unbeachtet. Implementierungen wie DiscoJuice oder Ac-

countChooser erleichtern es einem Benutzer seinen IDP auszuwählen, ermöglichen ihm aber nicht, eine vorher nicht vorhandene Verbindung zwischen SP und IDP herzustellen. Daher muss eine neue Lösung erstellt werden.

Für die Auswahl der Komponenten, auf denen die neue Lösung basieren soll, ist es ausschlaggebend, wie aufwändig die Implementierung der nötigen Funktionen wäre. Daher wird zunächst betrachtet, welche Anforderungen in welcher der Lösungen, Shibboleth und simpleSAMLphp, fehlen. Da diese Betrachtung die Funktionen der Lösungen vergleicht, werden nur funktionale Anforderungen überprüft.

Besonders interessant sind die Anforderungen, die nur bei einem der beiden Systeme fehlen. Bei simpleSAMLphp sind dies F16 Konvertierungsregel-Operationen, welche als „grundlegende Anforderung“ eingeordnet ist und F19 Attribute Release Policies, welche aus der Kategorie “erweiterte Anforderung“ stammt. Shibboleth hingegen fehlt nur die Anforderung F23 Testsysteme, welche als „optionale Anforderung“ klassifiziert ist.

Mit diesem Vergleich liegt die Entscheidung Shibboleth als Basis zu verwenden nahe. Die dort fehlende Anforderung ist nicht nur weniger relevant, sondern ist auch die Implementierung von vollständigen Konvertierungsregeloperationen komplex. Dabei auf ein schon bei vielen Administratoren bekanntes System zurückgreifen zu können, erhöht die Chancen deutlich auch angewendet zu werden. Ähnlich verhält es sich mit der Anforderung zu ARPs.

Der Vollständigkeit halber fehlen folgende Anforderungen bei beiden Implementierungen:

F1: Dynamischer Verbindungsaufbau
F2: Trusted Third Party
F3: TTP-Anmeldung
F4: Zuständigkeit
F5: Zugriffsberechtigungen
F6: Dienstverwaltung
F9: Metadatensynchronisation
F10: Benutzerinitiierter Verbindungsaufbau
F13: Konv.-Regelverwaltung
F14: Konv.-Regelbeschreibung
F15: Konv.-Regelbewertung
F17: Konv.-Regelabgleich
F18: Fehlerbenachrichtigungen
F20: Access Control Lists

Die Implementierungen, zur Vereinfachung der Auswahl eines IDPs DiscoJuice und AccountChooser, bringen beide keine relevanten Anforderungen mit sich, sodass sie die Entscheidung nicht beeinflussen.

Unter Berücksichtigung der Erweiterungen JANUS und PEER, werden die Anforderungen F2 Trusted Third Party, F3 TTP-Anmeldung, F4 Zuständigkeit, F5 Zugriffsberechtigungen und F6 Dienstverwaltung erfüllt. Diese sind zwar wichtig aber für ein Webfrontend eher Standard-Funktionen und daher nicht ausschlaggebend.

JANUS, aufbauend auf simpleSAMLphp, erfüllt allerdings zusätzlich die Anforderungen F18 Fehlerbenachrichtigungen, F20 Access Control Lists und F19 Attribute Release Policies. Unter Berücksichtigung, dass diese Anforderungen für simpleSAMLphp mit JANUS erfüllbar sind, fehlen simpleSAMLphp nur noch die Anforderung F16 Konv.-Regeloperationen. Dahingegen fehlen Shibboleth die Anforderungen F23 Testsysteme, F18 Fehlerbenachrichtigungen und F20 Access Control Lists.

Da die ersten beiden bei Shibboleth fehlenden Anforderungen nicht von hoher Wichtigkeit und Access Control Lists im Vergleich zu Konvertierungsregeloperationen, die schon bekannt

sind und für die viele Beispiele existieren, leichter neu zu implementieren sind, fällt die Entscheidung für eine Erweiterung auf Shibboleth. Außerdem wird Shibboleth von ca. 90% aller Provider der DFN-AAI-Basic-Föderation und ca. 85% der Provider der DFN-AAI-Advanced-Föderation eingesetzt. Diese Prozentzahlen wurden aus den aktuellen Metadaten der DFN-AAI, über die von den Providern angegebenen EntityIDs, bestimmt. Für Shibboleth-Provider enthält die EntityID im Normalfall die Zeichenkette „shibboleth“.

5. Konzeptionelle Erweiterung

In diesem Kapitel wird ein Konzept entwickelt, welches den Betrieb von Föderationen dynamischer und besser skalierbar machen soll. Insbesondere werden dabei die in Kapitel 3 aufgestellten Anforderungen betrachtet. Zunächst wird im ersten Abschnitt dieses Kapitels eine Übersicht, über den Lösungsansatz gegeben. Danach werden die besonders wichtigen Punkte der Metadatensynchronisation (Abschnitt 5.2) und der Konvertierungsregeln (Abschnitt 5.3) betrachtet. Anschließend wird ein Konzept für das Verwaltungssystem (Abschnitt 5.4) beschrieben und abschließend das entstandene System mit den Anforderungen verglichen (Abschnitt 5.5).

Das zu entwickelnde System muss entsprechend den in Kapitel 3 aufgezeigten Problemen und den daraus abgeleiteten Anforderungen vor allem dazu geeignet sein, schnell und flexibel Metadaten auszutauschen und nicht verfügbare Attribute aus anderen verfügbaren abzuleiten. Für ein solches System gibt es, ausgehend von dem aktuell in Hochschulumgebungen verwendeten, auf SAML und Shibboleth basierenden, Föderationskonzept vor allem den Ansatz eines zentralen Kommunikationsknotens.

Innerhalb von klassischen nationalen Föderationen gibt es schon zentrale Dienste wie den Discovery Service und einen Dienst zum Verteilen der gesammelten Metadaten. Dieses System könnte erweitert werden, um nicht automatisch den kompletten Metadatensatz an alle Teilnehmern zu veröffentlichen, sondern den Teilnehmern die gerade eine Kommunikationsbeziehung benötigen, dynamisch nur die dazu nötigen Metadaten zu vermitteln. Der zentrale Punkt könnte auch die Verwaltung von Konvertierungsregeln übernehmen und deren Abgleich zwischen IDP und SP koordinieren.

Das System würde dadurch dynamische virtuelle Föderationen zwischen den Teilnehmern, die eine Verbindung benötigen, kreieren. Diese Föderationen werden in Anlehnung an die virtuellen Organisationen aus dem Grid-Computing „virtuell“ genannt, da sie nicht durch eine reale Föderation abgebildet werden und nur solange Bestand haben, wie sie benötigt werden.

Ein weiterer Ansatz wäre ein vollständig verteiltes System, bei dem die beteiligten Dienste, ähnlich dem Ansatz von OpenID Connect, direkt miteinander kommunizieren. Ohne einen konkreten zentralen Punkt hätte dieses System keinen single point of failure, wodurch es weniger fehleranfällig, besser skalierbar und nicht auf einen Betreiber der zentralen Instanz angewiesen wäre. Dies würde der Anforderung F2 an eine TTP nicht zwangsläufig wiedersprechen, da ein äquivalentes System auch verteilt Implementiert werden könnte. Die konsistente Synchronisation von bestehenden virtuellen Föderationen, Metadaten und Konvertierungsregeln, würde die Komplexität des Gesamtsystems allerdings deutlich erhöhen. Zudem müssten alle Einstellungen der Dienstadministratoren lokal für jeden Dienst vorgenommen werden, wodurch mehr zusätzliche Software installiert und verwaltet werden muss. Durch den erhöhten Administrationsaufwand, würde die Akzeptanz des Systems sinken, weshalb im Folgenden ein Konzept mit minimalen Änderungen auf Seite der Dienstbetreiber und einem zentralen Verwaltungssystem entwickelt wird.

Das zentrale System hat weiterhin die Eigenschaft, dass alle Teilnehmer den Aussagen des Systems vertrauen müssen. Dieses System ist der praktischen Verwendung von OpenID Connect ähnlich, da es dort wenige frei zugängliche und relevante Identity-Provider, wie zum Beispiel Google, gibt. Das dadurch entstehende Vertrauensverhältnis ist prinzipiell in Föderationen nichts Neues, da dort auch alle Dienste den von der Föderation veröffentlichten Metadaten vertrauen müssen. Aufgrund dieses Vertrauensverhältnisses wird die, in Anforderung F2 geforderte, zentrale Instanz als „Vertrauenswürdige dritte Partei“ oder Trusted Third Party (TTP) bezeichnet.

5.1. Allgemeiner Ablauf

In diesem Abschnitt wird zunächst ein genereller Überblick über den Austausch von Metadaten zwischen SP und IDP über die TTP gegeben. Daraufhin werden zuerst verschiedene Lösungsansätze präsentiert und verglichen.

Ziel ist es eine Lösung zu finden, die das Grundprinzip, dass durch dynamische virtuelle Föderationen, Metadaten zwischen den teilnehmenden Organisationen nicht prophylaktisch, sondern erst, durch einen Benutzer angestoßen, ausgetauscht werden, erfüllt.

5.1.1. Lösungsansatz 1: IDP-Proxy

Eine schon bestehende Lösung für das Verbinden von Service Providern mit Identity Providern, wenn die SPs und IDPs in unterschiedlichen Föderationen sind, ist ein IDP-Proxy. Dessen Funktion wird in der Wiki des Internet2 Projektes beschrieben [CS12].

Im Wesentlichen ist ein IDP-Proxy ähnlich zu einem Web-Proxy, in dem er für den SP und IDP, die über den Proxy kommunizieren, transparent ist. Aus Sicht des SPs ist der IDP-Proxy ein IDP und aus Sicht des IDPs ein SP. Der Proxy kann durch das Zwischenspeichern (caching) von Attributen die Effizienz steigern, sowie durch den zentralen Zugangspunkt den Zugriff auf die Föderation leichter kontrollierbar machen. Abbildung 5.1 zeigt in einem Sequenzdiagramm den Ablauf der Kommunikation, wenn als TTP ein IDP-Proxy verwendet wird.

Der Benutzer, der sich auf der Webseite eines Service Providers anmelden möchte und keine aktive Sitzung bei dem SP hat, kann in dem Login-Dialog des Dienstanbieters, die TTP zur Anmeldung auswählen oder wird, wenn es keine alternative Option gibt, direkt dort hin weitergeleitet. Bei der Weiterleitung von SP zu TTP wird vom SP ein Authentication Request (authReqSP) übertragen. Der Benutzer kann darauf, falls er auch bei der TTP keine aktive Session hat, aus einer Liste von IDPs auswählen und wird von der TTP mit einem neuen Authentication Request (authReqTTP) zu dem ausgewählten IDP weitergeleitet. Falls der Benutzer durch einen anderen Dienst schon eine gültige Sitzung an der TTP hat, kann der ursprüngliche Request des SPs sofort beantwortet werden. Nach der Übertragung des Authentication Requests von der TTP zum IDP, muss der Benutzer sich authentifizieren und wird, mit einer Assertion über das Ergebnis der Authentifizierung, an die TTP zurückgeleitet. Dort wird die Assertion verifiziert und wenn diese in Folge eines erfolgreichen Loginvorgangs ausgestellt wurde und gültig ist, wird eine Assertion, die den Login bestätigt für den SP ausgestellt. Der SP kann dies dann ebenfalls verifizieren und den Benutzer in den geschützten Bereich weiterleiten.

Dieser Ansatz hat den Vorteil, dass gar keine Metadaten dynamisch ausgetauscht werden müssen, nur der Proxy muss die Metadaten aller angeschlossenen Systeme kennen. Außerdem müssen SP und IDP nicht verändert werden, da diese Funktionalität über SAML verfügbar ist. Allerdings ist dieser Ansatz für den Einsatz als TTP nicht geeignet, da die Skalierbarkeit und Fehlertoleranz durch den zentralen Proxydienst, der für jede Authentifizierung benötigt wird, begrenzt wird. Die TTP sollte nur für einen initialen Metadatenaustausch zwischen SP und IDP nötig sein, nachfolgende Authentifizierungen sollten direkt zwischen den beteiligten Parteien durchgeführt werden.

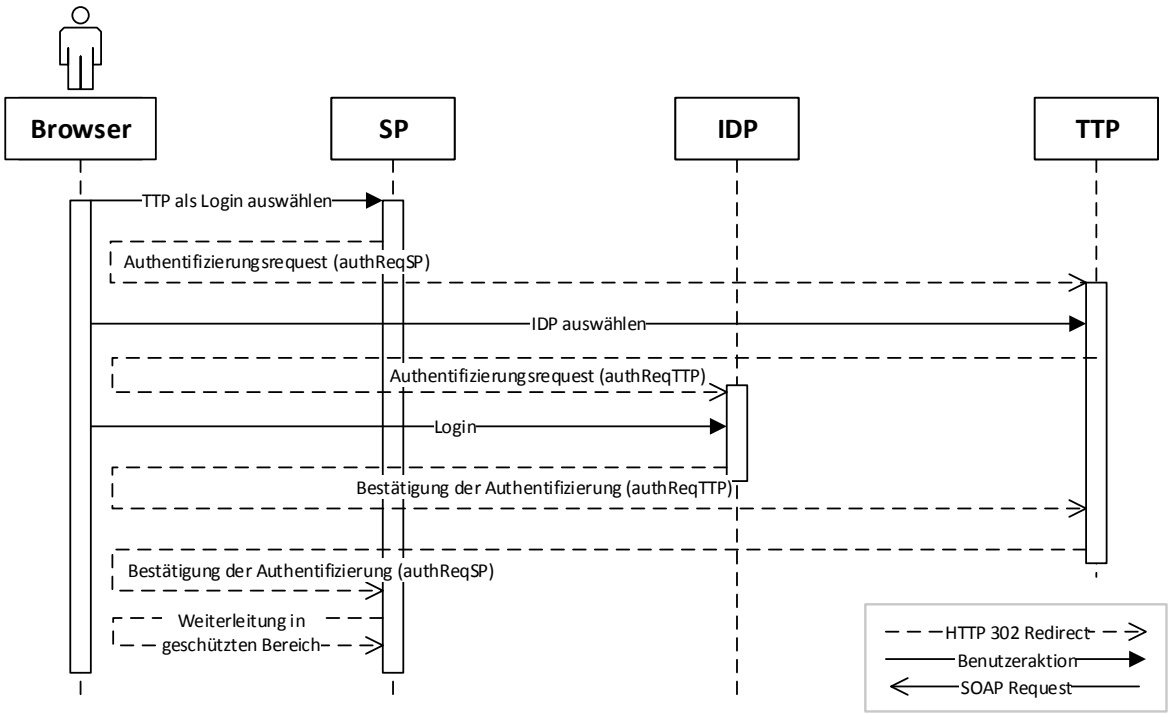

Abbildung 5.1.: Sequenzdiagramm für den Aufbau einer Vertrauensbeziehung durch eine TTP, die als IDP-Proxy implementiert ist.

5.1.2. Lösungsansatz 2: Modifizierter Centralized Discovery Service

Ein alternativer Ansatz basiert auf dem Shibboleth Centralized Discovery Service. Wie in Abschnitt 4.1.3 beschrieben, kann der CDS dazu verwendet werden innerhalb einer Föderation den passenden IDP auszuwählen. Dieser Ablauf kann erweitert werden, um eine dynamische virtuelle Föderation, durch den vom CDS angestoßenen Metadatenaustausch, aufzubauen. Abbildung 5.2 zeigt dazu einen möglichen Ablauf.

Der Ablauf in diesem Lösungsansatz beginnt wie bei dem des CDS. Der Benutzer greift auf eine geschützte Seite eines SPs zu und bekommt eine Menge von möglichen Authentifizierungsmethoden angeboten. Da der Benutzer FIM verwenden möchte, sein IDP aber nicht direkt mit dem SP verbunden ist, wählt er die TTP aus. Daraufhin wird er vom SP, entsprechend des DS-Protokolls, zur TTP weitergeleitet. Dort kann er seinen IDP, der wie der SP Teil des TTP-Netzwerks ist, auswählen. Um zu verifizieren, dass der Benutzer wirklich einen gültigen Account bei dem gewählten IDP hat, agiert die TTP im Folgenden als SP und stellt einen Authentication Request an den IDP. Der Benutzer wird mit dem Request zu dem IDP weitergeleitet, wo er sich authentifizieren muss oder eine schon gültige Sitzung hat. Nach der erfolgreichen Authentifizierung wird der Benutzer mit einer Authentication Response wieder zurück zur TTP geleitet, die die Response überprüft.

Die TTP kann nun zwischen IDP und SP den Metadatenaustausch anstoßen. Der genaue Ablauf hierfür ist hier nicht dargestellt, dieser wird in Abschnitt 5.2 näher beschrieben.

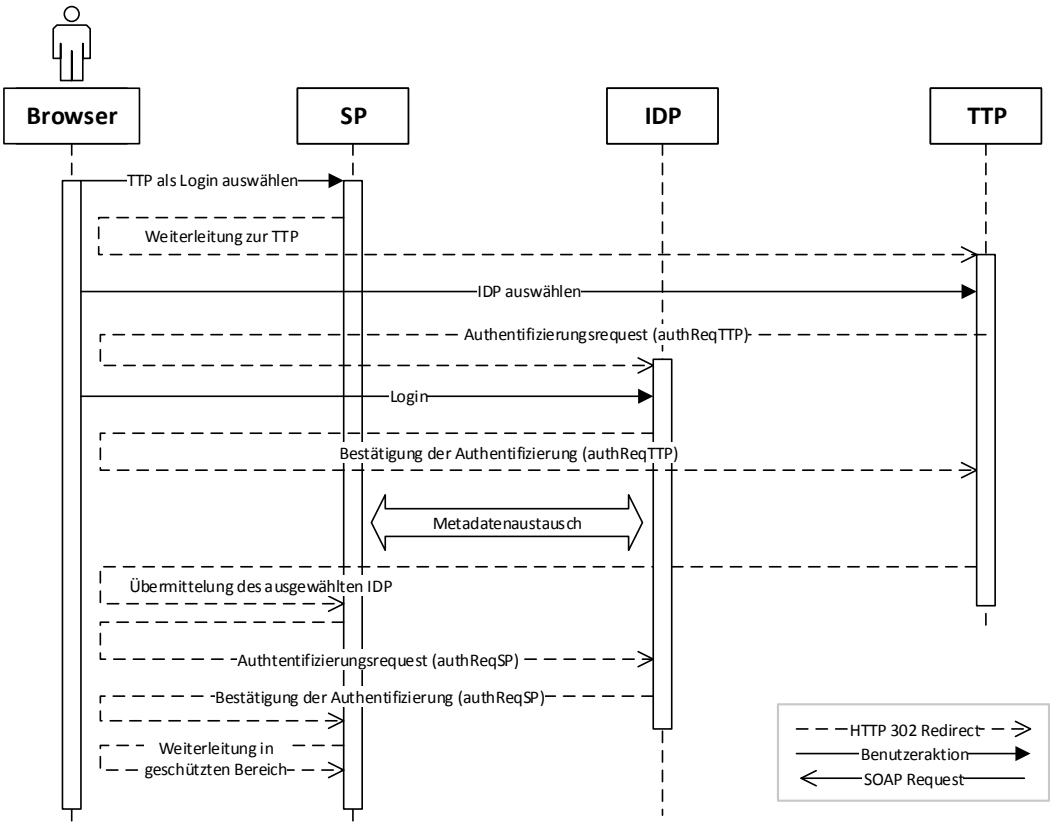

Abbildung 5.2.: Sequenzdiagramm für den Aufbau einer Vertrauensbeziehung durch einen modifizierten Centralized Discovery Service

Nachdem die Metadaten ausgetauscht wurden, existiert eine dynamische virtuelle Föderation zwischen SP und IDP und der SP kann wie gewohnt einen Authentication Request an den IDP schicken. Da der Benutzer sich erst vor kurzem dort eingeloggt hat, existiert diese Sitzung in der Regel noch und der Benutzer muss sich nicht noch einmal authentifizieren. Der IDP bestätigt die Identität des Benutzers und der SP kann den Benutzer nach erfolgreicher Validierung der Bestätigung in den geschützten Bereich leiten.

Diese Lösung hat allerdings das Problem, dass die TTP nicht sicher sein kann, dass der SP wirklich diese dynamische virtuelle Föderation aufbauen möchte. Die als HTTP-Redirect gestaltete Weiterleitung von SP zu TTP zur Auswahl des IDPs kann auf einfache Weise, direkt in der Adresszeile des Broswers von jedem abgesetzt werden. Dieser Fehler würde erst auffallen, wenn die TTP den Metadatenaustausch bei dem SP anstößt und dieser erkennen würde, dass er die Metadaten des fraglichen IDPs gar nicht angefragt hat. Dabei wird vorausgesetzt, dass der SP nachvollziehen kann, für welchen IDP er die Übermittlung von Metadaten erwarten kann. Sollte das nicht der Fall sein, könnte ein Dritter zwischen beliebigen Teilnehmern des TTP-Netzwerks Metadatenaustausche durchführen. Dieses Problem könnte potentiell für DOS-Angriffe verwendet werden.

Das selbe Problem tritt bei dem in Abschnitt 4.5 beschriebenen DiscoJuice in den Installationsoptionen „Service Provider Discovery“ und „Global Discovery Service“ auf. Hier wird das Problem dadurch gelöst, dass der SP eine Webseite betreibt, die der DS aufruft und von der er einen Bestätigungscode abfragt. Im nächsten Abschnitt wird ein Ansatz vorgestellt, der dieses Problem auf eine andere Art löst.

5.1.3. Lösungsansatz 3: Erweiterte Modifizierung des Centralized Discovery Service

Da der vorherige Lösungsansatz das Problem hatte, dass Verbindungsaufbau-Requests ohne Zutun des SP einfach durch den Nutzer im Browser abgesetzt werden können, wird hier eine Abwandlung der Herangehensweise beschrieben. Der grundsätzliche Ablauf ist sehr ähnlich zu dem des vorhergehenden Lösungsansatzes, greift aber auf den Prozess des Nachrichtenaustauschs wie bei dem im Abschnitt 5.1.1 vorgestellten IDP-Proxy zurück. Abbildung 5.3 zeigt diesen modifizierten Ablauf.

Im vorherigen Ablauf wurde nicht genau beschrieben, wie die Auswahl des IDPs für einen Benutzer aussehen soll. Hierfür wird in diesem Ansatz ein modifizierter Embedded Discovery Service, der in Abschnitt 4.1.3.1 vorgestellt wurde, verwendet. Der EDS kann lokal beim SP feststellen, welche Metadaten vorhanden sind und dadurch einen Loginvorgang mit allen schon bekannten IDPs initiieren. Der EDS benötigt nur eine weitere Option, die IDP-Auswahl an den CDS weiterzuleiten. Mit dieser Option wird der Benutzer, wie im oben beschrieben, zur Auswahl seines IDP, zur TTP weitergeleitet.

Die TTP, die auf einem CDS basiert, übermittelt die Auswahl, des Benutzers in diesem überarbeiteten Ablauf entspechend des normalen DS-Protokolls zurück an den SP. Dieser erstellt daraufhin einen Authentication Request für den vom Benutzer gewählten IDP und schickt diesen mit einer Signatur an die TTP, da er ohne die Metadaten des IDPs gar keinen Kommunikationsendpunkt an dem IDP kennt. Die TTP kann die Signatur prüfen und dadurch sicherstellen, dass der SP an dem Verbindungsaufbau beteiligt sein möchte.

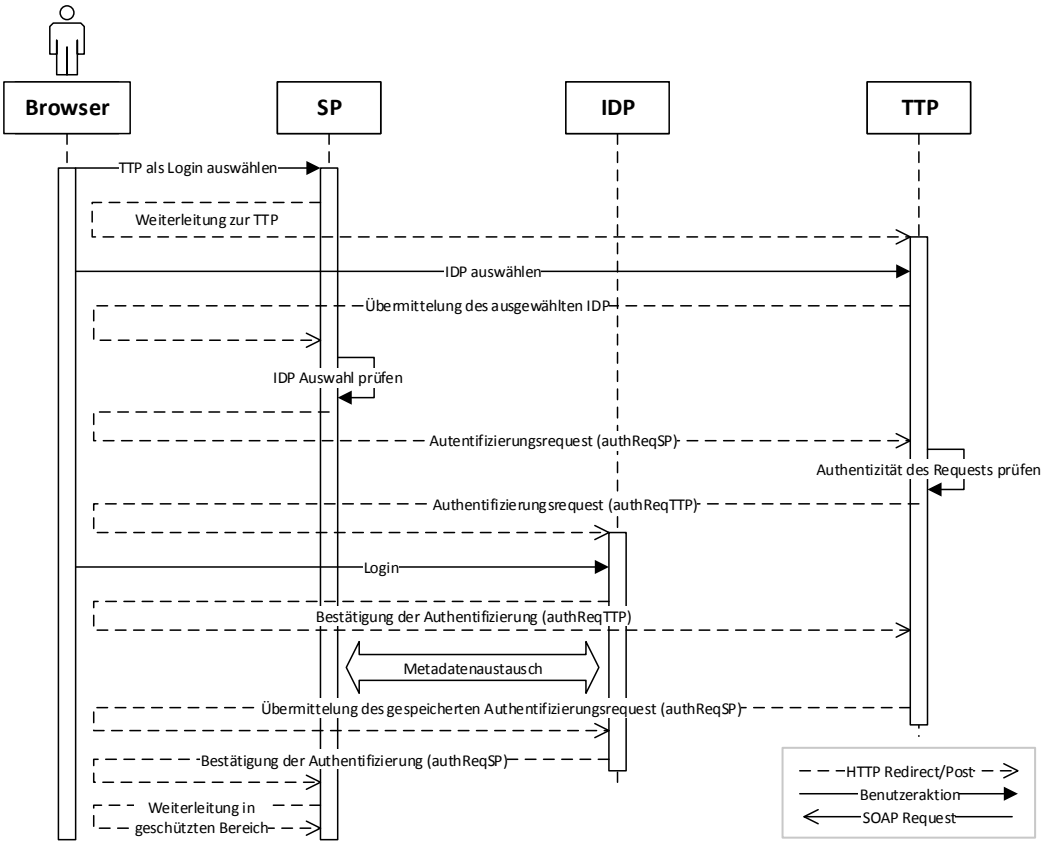

Abbildung 5.3.: Sequenzdiagramm für den Aufbau einer Vertrauensbeziehung durch modifizierten CDS mit Verifizierung des Requests durch die TTP

An dieser Stelle, da nun der beteiligte SP und IDP sicher bekannt sind, kann die TTP eine ACL implementieren, über die Verbindungen zwischen bestimmten Providern verboten oder erlaubt werden können.

Den Authentication Request des SP speichert die TTP zwischen, um vor der Initiierung des Metadatenaustausches festzustellen, dass der Benutzer einen IDP ausgewählt hat, bei dem er einen gültigen Account hat. Dazu erstellt die TTP, wie im vorhergehenden Ablauf, einen eigenen Authentication Request und sendet diesen an den IDP. Wenn der IDP bestätigt, dass der Benutzer sich authentifiziert hat, kann die TTP den Metadatenaustausch starten. Der Metadatenaustausch ist hier wieder nur abstrakt dargestellt, wird im folgenden Abschnitt näher beschrieben.

Nachdem der Metadatenaustausch erfolgreich abgeschlossen wurde, übermittelt die TTP den Benutzer mit dem zuvor gespeicherten Authentication Request des SP an den IDP. Da hier wie zuvor der Benutzer erst vor wenigen Sekunden authentifiziert wurde, existiert vermutlich noch eine gültige Sitzung und der IDP leitet den Benutzer mit einer Authentication Response an den SP weiter. Der SP kann diese Response dann validieren und bei einer gültigen Authentifizierung in den geschützten Bereich weiterleiten.

5.2. Synchronisation von Metadaten

In diesem Abschnitt wird beschrieben, wie die Metadaten zwischen Service Provider und Identity Provider über eine Trusted Third Party ausgetauscht werden können. Für den Austausch der Metadaten werden mehrere Komponenten benötigt. Zum einen muss es für SP und IDP einen neuen **MetadataProvider** geben, der die über die TTP synchronisierten Metadaten verwendet und den anderen Programmteilen von SP und IDP zur Verfügung stellt. Dieser MetadataProvider wird in Abschnitt 5.2.2 beschrieben. Der MetadataProvider benötigt eine Kommunikationsschnittstelle, über die der Metadatenaustausch initiiert werden kann und die Metadatendateien heruntergeladen werden können. Diese Schnittstelle wird als **MetadataSyncHandler** bezeichnet und in Abschnitt 5.2.3 beschrieben. Der grundlegende Ablauf der Kommunikation zur Metadatensynchronisation wird im folgenden Abschnitt vorgestellt.

5.2.1. Kommunikation

Für die Kommunikation zu dem Austausch gibt es prinzipiell zwei grundlegende Methoden. Die erste Methode besteht darin, eine Kopie der Metadaten bei der TTP zu hinterlegen und diese Kopie an die Teilnehmer herauszugeben. In der zweiten tritt die TTP nur als Vermittler auf, speichert daher selbst keine Metadaten und übermittelt nur URLs, unter denen sich zum Beispiel ein IDP direkt die Metadaten von dem SP (oder andersherum) herunterladen kann. Dieses Modell ist sehr flexibel und ermöglicht es, den Parteien, die Metadaten zur Verfügung zu stellen, sowie diese beliebig zu modifizieren. Shibboleth IDP und SP haben experimentelle Web-Schnittstellen, um Metadaten basierend auf der aktuellen Konfiguration zu generieren. Die Verwendung dieser Schnittstellen würde jegliche Latenz, die zwischen der Aktivierung von geänderten Metadaten eines Dienstes und der Verfügbarkeit der geänderten Metadaten bei der TTP eliminieren.

Etwas abstrahiert, sind beide Methoden allerdings miteinander kompatibel, bzw. von der Implementierung her identisch, so dass nicht explizit zwischen beiden unterschieden werden muss. In jedem Fall muss der Client, der Metadaten einer anderen Partei herunterladen möchte, über den Speicherort, der Metadaten, in Form einer URL, informiert werden. Ob diese URL nun zu einer Datei auf dem Host der TTP zeigt, oder direkt zu einem SP/IDP, muss nicht festgelegt werden. Dadurch ist auch ein Mischbetrieb möglich, bei dem SPs und IDPs entscheiden können, ob sie die Metadaten selber hosten möchten, und entsprechend bei der TTP nur eine URL hinterlegen oder der TTP die Speicherung einer Kopie der Metadaten überlassen. Grundvoraussetzung ist nur, dass die Metadaten eines Dienstes durch eine URL abrufbar sind.

Neben der Art der Speicherung der Metadaten ist ein wichtiger Aspekt der Metadatensynchronisation, ob diese nach einem Push- oder Pull-Modell konzipiert wird. Nach dem Pull-Modell fragen die Teilnehmer, zu einem von ihnen bestimmten Zeitpunkt, den Metadatenaustausch bei der TTP an. Für den im vorherigen Abschnitt vorgestellten Ablauf, ist dieses Modell ungeeignet. SP und IDP wissen nicht, zu welchem Zeitpunkt es relevant ist, den Metadatenaustausch zu beginnen. Der SP, weiß nicht, wie lange die TTP benötigt, um den Benutzer bei dem IDP zu authentifizieren. Diese Zeit hängt häuptsächlich davon ab, wie lange der Benutzer für den Loginvorgang benötigt, bzw. kann unbestimmt lange dauern, wenn der Benutzer den Vorgang abbricht.

Der IDP hat eine bessere Möglichkeit festzustellen, wann er die Metadaten des SPs durch das Pull-Modell anfordern muss, da er nach dem Übertragen der Authentifizierungsbestätigung für den Benutzer die Metadaten des SPs benötigt, um im nächsten Schritt dessen Request zu bestätigen. Allerdings weiß der IDP zu dem Zeitpunkt nicht, welcher SP den Verbindungsaufbau initiiert hat, also auch nicht welche Metadaten er anfordern soll. Auch hier müsste der Standard Authentifizierungsvorgang modifiziert und zusätzlich Statusinformationen bei der TTP hinterlegt werden.

Die Verwendung der Push-basierten Kommunikation ist deutlich einfacher. Nachdem die TTP bestätigt hat, dass der Benutzer sich an dem von ihm ausgewählten IDP anmelden kann, kann sie den Metadatenaustausch initiieren. Dabei muss nur an der TTP die Statusinformation gespeichert werden, welcher SP und IDP beteiligt sind. Das DS-Protokoll auf SP-Seite bzw. das Authentifizierungs-Protokoll auf IDP-Seite muss nicht modifiziert werden und SP/IDP müssen keine Statusinformationen speichern. Da SP und IDP bei dem Verbindungsaufbau darauf vertrauen müssen, dass die TTP die richtigen Metadaten übermittelt, können sie auch darauf vertrauen, dass die TTP nur dann einen Metadatenaustausch initiiert, wenn er notwendig ist. Dabei muss darauf geachtet werden, dass nur die TTP den Metadatenaustausch starten kann.

Für das Push-Modell benötigt die TTP allerdings zusätzlich die Information darüber, wo bei SP und IDP die Schnittstellen existieren, die einen Metadatenaustausch initiieren können. Diese Kommunikationsendpunkte können als Erweiterung in den Metadaten der Dienste hinterlegt werden. Alternativ könnten sie auch bei der Registrierung eines Dienstes separat bei der TTP hinterlegt oder unter einer standardisierten URL angegeben werden müssen. Die beiden letzteren Optionen sind allerdings recht unflexibel und eine separate Angabe der Endpunkte würde auch eine zusätzliche Methode zum Austausch der Endpunkte benötigen, wodurch die Angabe in der Erweiterungs-Sektion der Metadaten am praktikabelsten ist.

Abbildung 5.4 zeigt den ausgewählten Ablauf zur Metadatensynchronisation. Die TTP veranlasst den Metadatenaustausch dadurch, dass sie dem IDP einen HTTP-GET-Request an die Metadatensynchronisationsschnittstelle schickt, der die entityID des SPs und die URL, unter der die Metadaten gespeichert sind, als Parameter enthält. Der IDP kann dann die Metadaten anhand der URL herunterladen und speichern. Über den Return-Code des GET-Requests zeigt der IDP der TTP an, ob der Download erfolgreich war.

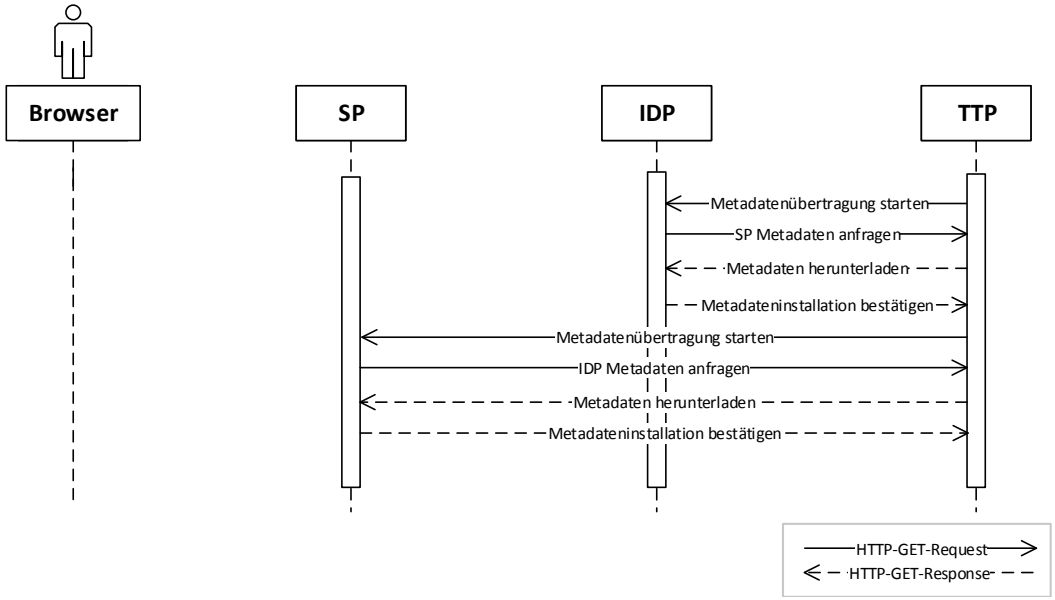

Abbildung 5.4.: Sequenzdiagramm für den Austausch von Metadaten zwischen SP und IDP über die TTP

Wenn der IDP die Metadaten erfolgreich gespeichert hat, beginnt die TTP den Metadatendownload auf Seite des SPs mit einem äquivalenten HTTP-GET-Request. Im Falle eines Fehlers, muss dieser Schritt nicht mehr durchgeführt werden und der Vorgang kann abgebrochen und der Benutzer über den Fehler informiert werden.

Nachdem IDP und SP die Metadaten des jeweils anderen Dienstes integriert haben, kann mit dem weiteren Ablauf zum Verbindungsvorgang fortgefahren werden.

5.2.2. MetadataProvider

Der MetadataProvider ist dafür zuständig, die von der TTP heruntergeladenen Metadatendateien, den anderen Prozessteilen, zum Beispiel zur Erstellung von AuthnRequests und Assertions, bereitzustellen.

Bei der Auswahl des Speichersystems für die Metadaten muss berücksichtigt werden, wie im Normalfall auf die Metadaten zugegriffen wird. Die häufigste Suchoperation, die ein MetadataProvider durchführen muss, ist es die Metadaten zu einer gegebenen entityID zu finden und wenn diese vorhanden sind, sie an die anfragende Prozedur zurückzugeben. Diese Anfrage muss im Prinzip jedes mal beantwortet werden, wenn auf Seite des IDPs eine Assertion bzw.

auf Seite des SPs eine Anfrage gestellt wird. Daher sollte diese Operation besonders performant sein. Eine weniger wichtige, weil seltener durchzuführende und nicht zeitkritische, Operation ist es neue Metadaten in den Speicher einzufügen oder zu entfernen.

Für die Speicherung der Metadaten gibt es daher zwei Optionen. Zum einen, kann eine Datenbank verwendet werden, die im Falle einer relationalen SQL basierten Datenbank die entityID als primären Schlüssel verwendet und die eigentlichen Metadaten als Binary Large Object (BLOB) speichert. Alternativ könnte eine dokumentenbasierte XML-Datenbank verwendet werden, da die Metadaten als XML vorliegen und so einfacher zu durchsuchen wären. Bei der vorherigen Betrachtung der notwendigen Operationen auf dem Speichersystem, ist es allerdings in keinem Fall notwendig die Metadaten direkt durchsuchen zu können, weshalb eine relationale Datenbank ausreichen würde. Zusätzlich können in der Datenbank Metainformationen gespeichert werden. Interessant könnte es sein, den Zeitpunkt, an dem die Metadaten eingefügt wurden, zu speichern, um festzustellen, wann sie veraltet sind und entfernt werden sollen. Außerdem könnte gespeichert werden, wann die Metadaten zuletzt verwendet wurden, woraus auch darauf geschlossen werden kann, ob die Metadaten noch benötigt werden oder entfernt werden können. Ein Nachteil des Datenbanken Ansatzes ist es, dass eine Datenbank installiert und eingerichtet werden muss.

Die zweite Option, verwendet daher keine Datenbank, sondern legt die Metadatendateien im Filesystem ab. Dabei kann die entityID als Dateiname verwendet werden und bietet so eine einfache Methode die Dateien performant auffindbar zu machen. Da die entityID normalerweise Zeichen wie „/“, „:“ oder „?“ enthält, die, je nach Betriebssystem und Filesystem, in einem Dateinamen nicht vorkommen dürfen, muss die entityID dafür transformiert werden. Es wäre denkbar diese Zeichen durch andere, wie zum Beispiel „_“, „,“und „#“ zu ersetzten, da diese nicht in Domainnamen vorkommen dürfen. Da allerdings unterschiedliche Betriebssysteme und Filesysteme unterschiedliche Regeln für Dateinamen haben, ist diese Option fehleranfällig. Alternativ kann die gesamte entityID encodiert werden, zum Beispiel als Base64 oder durch einen Hash. Ein Hash hat dabei den weiteren Vorteil, dass die Länge des Dateinamen bekannt und begrenzt ist und somit dort keine Komplikationen mit dem verwendeten Dateisystem auftreten können. Die verwendete Hashfunktion muss in erster Linie schnell und möglichst kollisionsfrei sein. Es ist nicht nötig, dass die Hashfunktion kryptografisch sicher ist. Der verwendete Hashalgorithmus muss nur bei dem Dienst selber bekannt sein, so dass verschiedene Dienste problemlos verschiedene Hashfunktionen verwenden können. Mögliche Hashfunktionen sind zum Beispiel MD5 [RFC7033] oder SHA1 [RFC7033]. Der größte Nachteil an der Verwendung von Hashfunktionen zur Generierung der Dateinamen tritt beim Reverse-Lookup auf. Aus der Liste der Dateien ist nicht direkt ersichtlich zu welchen entityIDs die gespeicherten Metadaten gehören. Diese Operation wird aber nicht benötigt. Für das Auflisten der verfügbaren entityIDs müssen vom MetadataProvider nur die Metadaten aller ihm bekannten Dienste zurückgegeben werden. Die Aufbereitung übernimmt ein anderer Programmteil. Der Ort an dem die Metadaten gespeichert werden sollen, kann über eine Konfigurationsdatei angegeben werden.

Um nicht für jeden Lookup auf das Dateisystem zugreifen zu müssen, kann der MetadataProvider einen Cache implementieren, in dem er die zuletzt benutzten Metadaten zwischenspeichert.

Der MetadataProvider benötigt des weiteren eine Schnittstelle zum, im nächsten Abschnitt beschrieben MetadataSyncHandler, über die er informiert wird, wenn neue Metadaten herun-

Listing 5.1: Initiierung des Metadaten-Downloads

```
http://sp.example.com/Shibboleth.sso/MetaSync
?entityID=https://idp.example.com/idp/shibboleth
&location=https://ttp.example.com/discovery/ttp/getMetadata/action/fetchMetadata
  ?entityID=https://idp.exampl.com/idp/shibboleth
```

tergeladen wurden. Dadurch kann er seinen Cache aktualisieren und andere Programmteile darüber informieren, dass sich die Metadaten, die dieser Provider anbieten kann, geändert haben.

5.2.3. MetadataSyncHandler

Dieser Handler wird, wie in den vorhergehenden Abschnitten beschrieben, durch die TTP sowohl bei SP als auch IDP durch einen HTTP-GET-Request, wie in Listing 5.1 dargestellt, aktiviert. Dieser Request enthält die Parameter `entityID` mit der EntityID des Dienstes, dessen Metadaten heruntergeladen werden und `location` eine URL, die angibt, wo diese gespeichert sind. In dem abgebildeten Beispiel wurden jegliche für die Parameter normalerweise nötige URL-Kodierung zur besseren Lesbarkeit entfernt. Zusätzlich benötigt der Handler, angegeben durch eine Konfigurationsdatei, den Ort, an dem die Metadaten gespeichert werden sollen.

Wenn der Handler einen Request empfängt, liest er die URL-Parameter `entityID` und `location` aus. Durch den entityID Parameter kann sofort der Speicherort bestimmt und überprüft werden, ob die dazugehörigen Metadaten eventuell schon existieren und erneuert werden sollen.

Der Handler erzeugt nun einen eigenen GET-Request an die übertragene URL und lädt die dort verlinkte Datei herunter. Dabei ist darauf zu achten, dass die angegebene URL eventuell nicht den eigentlichen Speicherort angibt, sondern eine Weiterleitung enthält. Nach einem erfolgreichen Download wird der MetadataProvider über die geänderten Metadaten informiert und der Request der TTP mit einem Status-Code `200 Ok` beantwortet.

Sollte bei der Übertragung oder Speicherung ein Fehler aufgetreten sein, wird der Request der TTP mit einem Status-Code `500 Internal Server Error` beantwortet. In dem Text der Response können im Fehlerfall Details zu dem Fehler übermittelt werden.

Wichtig für das Funktionieren dieses Ablaufs ist, dass der HTTP-GET-Request der TTP ein genügend großes Timout-Fenster hat, sodass dem Dienst ausreichend Zeit für den Download und das Speichern der Metadaten bleibt, bis der Request beantwortet wird. Normalerweise sind diese Timeout-Fenster sehr groß gewählt, zum Beispiel ist im Firefox Browser 31 über den Config-Parameter `network.http.response.timeout` der Timeout mit einem Standardwert von 300 Sekunden (5 Minuten) definiert. Ein Fenster, das für diese Vorgänge vollkommen ausreichend sein sollte, da die Dienste in der Regel über eine schnelle Internetanbindung verfügen und die Metadatendateien nicht sehr groß sind. Als Referenz für die durchschnittliche Größe einer Metadatendatei kann hier die in Tabelle 3.1 angegebenen Größen für Föderationsmetadatensätze, die jeweils die Metadaten von hunderten Diensten

enthalten, verwendet werden. In Tests wurden die Metadaten immer innerhalb von weniger als einer Sekunde übertragen.

Für den IDP gibt es in diesem Ablauf allerdings einen Spezialfall. Nach der Übertragung der SP Metadaten kann dieser daraus feststellen, welche Attribute der SP anfragen möchte. Sollte der IDP nicht in der Lage sein diese Attribute selber zu generieren, kann er die Konvertierunsregelsynchronisation, die im nächsten Abschnitt beschrieben wird, initiieren.

5.3. Synchronisation von Konvertierungsregeln

Dieser Abschnitt beschreibt eine Methode zur Speicherung, Verwaltung und Übertragung von Konvertierungsregeln. Zunächst wird ein Überblick über die verschiedenen Möglichkeiten Konvertierungsregeln zu implementieren gegeben. Daraufhin werden für die geeignetste Methode die einzelnen Komponenten detailliert beschrieben. Das Suchen nach Konvertierungsregeln wird in Abschnitt 5.3.1, das Übertragen in Abschnitt 5.3.2 sowie das Anwenden und die Struktur der Regeln in Abschnitt 5.3.3 behandelt.

Es gibt prinzipiell zwei Möglichkeiten Konvertierungsregeln in den bekannten SAML-Ablauf einzubringen. Zum einen können sie beim Service Provider angewendet werden, zum anderen beim Identity Provider. Für die Seite des SPs würde sprechen, dass dieser ein bestimmtes Attribut benötigt, welches aus anderen Attributen ableitbar ist und es daher einfacher ist die Konvertierung einmal bei dem SP zu konfigurieren, als sie bei jedem IDP, der mit dem SP kommunizieren möchte, zu installieren. Der IDP wäre allerdings eine gute Wahl, da dieser für die Verwaltung von Attributen zuständig ist und somit erwartet wird, dass er alle Attribute, die einen seiner Benutzer beschreiben, kennt. Außerdem bietet Shibboleth für IDPs bereits, wie in Abschnitt 4.1.2 beschrieben, eine Möglichkeit Attribute zu extrahieren und zu konvertieren. Es ist daher sinnvoller die neuen Attribute auf der Seite des IDPs abzuleiten und so viel wie möglich schon vorhandene Mechaniken zur Verwaltung von Attributen zu verwenden.

Da durch die Synchronisation der Metadaten, welche in Abschnitt 5.2 beschrieben wurde, schon ein Grundgerüst für die Kommunikation zwischen TTP und IDP gegeben ist, muss diese in Kombination mit der Synchronisation der Konvertierungsregeln funktionieren. Abbildung 5.5 zeigt, wie der im Folgenden beschriebene Austausch dort integriert werden kann.

Nachdem der IDP die Metadaten des SP erhalten hat, kann er anhand der darin, über das `AttributeConsumingService`-Element geforderten Attribute, feststellen, welche Attribute der SP benötigt und ob er diese zur Verfügung stellen kann. Da dieses Element in vielen Fällen in den Standardeinstellungen nicht in die Metadaten des SPs integriert wird, muss festgelegt werden, dass ein SP, der möchte, dass der IDP versucht alle Attribute zu generieren, dieses Element führen muss.

Der IDP kann dann für die Attribute, die benötigt werden, ihm aber nicht zur Verfügung stehen, bei der TTP anfragen, ob diese über passende Konvertierungsregeln verfügt. Anhand einer Konvertierungsregeldatenbank kann die TTP nun eine Reihe von Konvertierungsregeln finden, mit denen die verfügbaren Attribute in die gesuchten umgewandelt werden können.

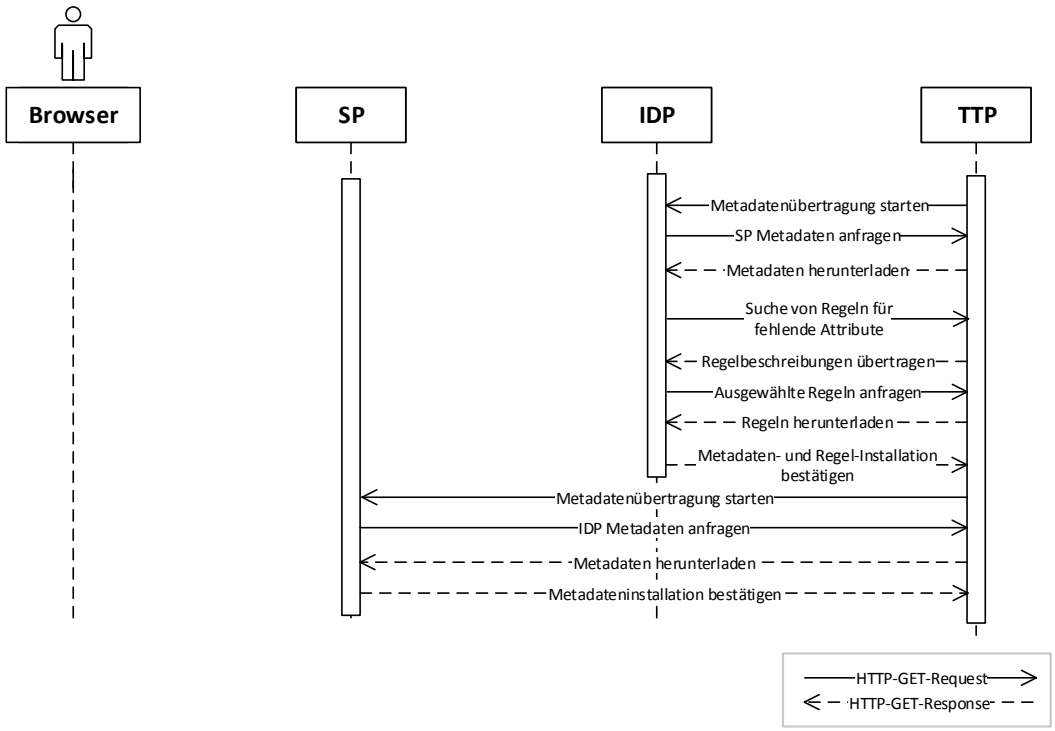

Abbildung 5.5.: Sequenzdiagramm für den Austausch von Konvertierungs-Regeln zwischen TTP und IDP

Sollten mehrere Regeln zu der gleichen Umwandlung gefunden werden, können diese entweder direkt bei der TTP anhand eines Rankings der Regeln gefiltert oder die Auswahl der geeignetsten Regel dem IDP überlassen werden. Daher werden nicht sofort die eigentlichen Regeln übertragen, sondern nur die Metadaten der Regel.

Falls keine passenden Regeln gefunden werden, besteht die Möglichkeit die Administratoren der betroffenen Dienste zu informieren, die dann eventuell selbst Regeln erstellen können. Der aktuelle Verbindungsversuch des Benutzers muss aber unvollständig abgebrochen, Konfigurationsänderungen bei dem IDP rückgängig und der Benutzer über den Abbruch informiert werden.

Nach der Auswahl passender Regeln durch den IDP, kann dieser dann anhand der eindeutigen ID einer Regel deren Download von der TTP initiieren und die Regel lokal installieren. Wenn die Integration der Regeln abgeschlossen ist, kann der IDP der TTP zurückmelden, dass er den Vorgang beendet wurde und diese, im Erfolgsfall, dann mit der Metadatensynchronisation des SPs fortfahren.

5.3.1. Suche

Bei der Suche nach Konvertierungsregeln ist die einfachste Suchoperation, alle Regeln zu finden, die ein bestimmtes Attribut erzeugen, da dabei nur ein Attribut angegeben werden muss. Allerdings werden viele Regeln zurückgegeben, die vorraussichtlich gar nicht anwend-

Listing 5.2: Suchanfrage für Konvertierungsregeln

```
http://example.com/convRuleSearch
?source_oids=givenName
&source_oids=sn
&target_oid=cn
&sourceEntityID=https://idp.example.com/idp/shibboleth
&destEntityID=https://sp.example.com/shibboleth
```

bar sind, weil die Quellattribute nicht vorhanden sind. Daher sollten die vorhandenen Attribute in die Suche einfließen, wodurch die Menge der Attribute, die zurückgegeben wird, auch möglichst klein bleibt und der Großteil der Filterung schon durch die Datenbank übernommen wird, welche für diese Operation optimiert ist. Eine Suche nach Attributen, die sich aus einer gegebenen Menge von Quellattributen ableiten lässt, ist für einen Dienstadministrator auch interessant, um abschätzen zu können, wie viele Informationen aus seinen Attributen abgeleitet werden können.

Ein weiterer Punkt, der bei der Suche nach Konvertierungsregeln beachtet werden muss, sind eventuelle Attribute Release Policies, wie sie in Anforderung F19 beschrieben wurden. Es ist daher möglich, dass die Attribute eines IDPs ausreichen ein weiteres Attribut abzuleiten, dieser das jedoch nicht erlaubt.

Aus obigen Überlegungen ergeben sich für die Such-Schnittstelle folgende Parameter:

- **Quellattribute**: Ein Array von mehreren Attributen, welches die Menge der verfügbaren Attribute darstellt. Im Folgenden als Parameter `source_oids` bezeichnet.
- **Zielattribut**: Ein Parameter, welcher das zu erzeugende Attribut benennt. Im Folgenden als als Parameter `target_oid` bezeichnet.
- **Identity Provider**: IDP, der das neue Attribut erzeugen soll. Im Folgenden als als Parameter `sourceEntityID` bezeichnet.
- **Service Provider**: SP, der das neue Attribut abfragen möchte. Im Folgenden als als Parameter `destEntityID` bezeichnet.

Die Suche kann dabei entweder Quellattribute, ein Zielattribut oder beides enthalten. Zusätzlich kann ein Paar IDP–SP angegeben werden, um die Resultate entsprechend der Attribute Release Policies dieser Verbindung zu filtern. Als flexibler Dienst für die automatische Suche eignet sich ein Web-Interface, bei dem die Suchanfrage über die URL-Parameter spezifiziert wird. Listing 5.2 zeigt ein Beispiel für einen solchen Request, der ausgehend von den Attributen `givenName` und `sn` (surname) das Attribut `cn` (common name) sucht. Dabei werden zusätzlich der Quell-IDP sowie der Ziel-SP angegeben.

Das Ergebnis der Suche wird als Antwort der Anfrage zurückgegeben. Der HTTP-Statuscode zeigt dabei zunächst an, ob die Suche erfolgreich war (Status 200), ob ein Fehler aufgetreten ist (Status 500) oder ob dieses Attribut wegen einer ARP nicht freigegeben werden kann (Status 403). Im Erfolgsfall wird in der Antwort nicht direkt der Inhalt der Konvertierungsregel zurückgegeben, sondern eine Liste der Regeln mit ihren Eigenschaften. Für diese Liste bietet es sich an, eine XML-Repräsentation zu wählen, da diese sowohl von anderen Programmen verarbeitet sowie auch direkt von Menschen gelesen werden kann. Zudem

Listing 5.3: Antwort auf eine Konvertierungsregel-Suchanfrage

```
<ruleDescriptions>
  <ruleDescription id="2" name="givenName+surname=commonName">
    <target friendlyName="cn"
    nameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri"
    >urn:oid:2.5.4.3</target>
    <dependencies>
      <dependency friendlyName="sn"
    nameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri"
    >urn:oid:2.5.4.4</dependency>
      <dependency friendlyName="givenName"
    nameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri"
    >urn:oid:2.5.4.42</dependency>
    </dependencies>
  </ruleDescription>
</ruleDescriptions>
```

Listing 5.4: Formale Definition einer Konvertierungsregel-Suchanfrage

```
<!ELEMENT ruleDescriptions (ruleDescription*)>

<!ELEMENT ruleDescription (target, dependencies?)>
<!ATTLIST ruleDescription
  id    CDATA      #REQUIRED
  name  CDATA      #REQUIRED
>

<!ELEMENT target (#PCDATA)>
<!ATTLIST target
  friendlyName CDATA      #REQUIRED
  nameFormat  CDATA      #REQUIRED
>

<!ELEMENT dependencies (dependency+)>
<!ELEMENT dependency (#PCDATA)>
<!ATTLIST dependency
friendlyName CDATA      #REQUIRED
nameFormat  CDATA      #REQUIRED
>
```

werden im Shibboleth-Umfeld alle Nachrichten in XML formatiert, wodurch die nötigen Parser schon vorhanden sind. Dazu wird der entsprechende HTTP-Content-Type „application/xml“verwendet. Listing 5.3 zeigt eine solche Antwort beispielhaft.

Eine formale Definition der Antwort, ist durch die in Listing 5.4 dargestellte Schema-Definition (DTD) gegeben. Die Antwort enthält das Wurzel-Element `ruleDescriptions`, das für jede gefundene Regel jeweils ein `ruleDescription`-Element enthält. Für jede Regel wird dabei eine eindeutige ID sowie der Name der Regel angegeben. Als Kindelemente enthält eine Regel mindestens das `target`-Element, welches das Attribut, das durch die Regel erzeugt wird, beschreibt. Dabei geben die Attribute des Elements den menschenlesbaren Namen `friendlyName` sowie das für den Attributsnamen verwendete Format `nameFormat` an. Der eigentliche Name des Zielattributs wird als Inhalt des Elements übertragen. Falls die Regel Abhängigkeiten zu anderen Attributen hat, enthält sie das Element `dependencies`, welches für jedes benötigte Attribut ein Element `dependency` enthält. Die `dependency`-Elemente sind identisch zu dem `target`-Element aufgebaut.

Listing 5.5: Downloadanfrage für eine Konvertierungsregel

```
1 http://example.com/convRuleDownload?id=21
```

5.3.2. Übertragung

Die Übertragung der eigentlichen Konvertierungsregeln wird in Anlehnung an den zuvor beschriebenen Suchmechanismus ebenfalls über HTTP durchgeführt. Der Download einer bestimmten Regel wird initiiert durch das Aufrufen einer Download-URL und der Angabe einer Regel-ID, wie sie in den Suchergebnissen spezifiziert ist. Listing 5.5 zeigt ein Beispiel für einen Download der Regel mit der ID 21.

Der Server sucht nun in seinem Speichersystem nach der entsprechenden Regel und gibt sie mit dem Content-Type „application/xml“und Status `200 Ok` zurück. Sollte die Regel-ID ungültig sein, da keine Regel zu der ID gefunden werden kann, wird nur der Status `404 Not found` zurückgegeben. Tritt bei der Verarbeitung der Anfrage ein weiterer Fehler auf, so wird der Status `500 Internal Server Error` mit einer optionalen Beschreibung des Fehlers als Text zurückgegeben.

5.3.3. Anwendung & Struktur

Nach einer erfolgreichen Übertragung, kann die Regel angewendet werden. Dazu muss zunächst festgelegt werden, welche Form und Struktur die Beschreibung einer Konvertierungsregel überhaupt hat. Da zuvor in Abschnitt 5.3 festgelegt wurde, dass die Konvertierungsregeln auf Seite des IDP integriert werden sollen und dieser durch die in Abschnitt 4.1.2 beschriebene und in Listing 4.2 beispielhaft dargestellte Datei `attribute-resolver.xml` schon über die Konvertierungsfähigkeit lokaler Attribute verfügt, müssen neue Regeln ebenfalls in diese Datei integriert werden. Außerdem müssen die Regeln so allgemein gehalten sein, dass sie unabhängig von den lokal vergebenen IDs für `AttributeDefinition`- und `DataConnector`-Elemente funktionieren.

Neben der `attribute-resolver.xml` muss auch die Datei `attribute-filter.xml` modifiziert werden. Diese Datei regelt, wie in Abschnitt 4.1.2 beschrieben, welche Attribute an einen SP herausgegeben werden dürfen. Solange das Attribut nur definiert nicht aber auch freigegeben ist, kann es nicht verwendet werden. Für das Modifizieren von XML-Dateien bietet sich die Verwendung von Extensible Stylesheet Language Transformations (XSLT) an, was bedeutet, dass die Regelbeschreibungen als XSLT zu definieren sind oder eine automatisierte Übersetzung von XML-Regel-Definitionen zu einer entsprechenden XSLT entwickelt werden muss.

Die Freigaben sind dabei über `AttributeFilterPolicy`-Elemente, die über eine `id` identifiziert werden können, gruppiert. Innerhalb dieser Gruppierungen, können Bedingungen definiert werden, die zum Beispiel festlegen, für welchen SP diese Gruppe an Freigaben gilt. Dies ist essentiell für die Umsetzung von Attribute Release Policies. Es muss verhindert werden, dass ein SP, der explizit Freigegeben ist, ein spezielles Attribut über eine Konvertierungsregel erstellt und das Attribut dadurch für alle anderen SPs zugänglich macht. Für

jeden SP wird daher eine solche Gruppe erstellt, die über EntityID des SPs als `id` identifiziert werden kann.

Die eigentliche Freigabe erfolgt durch ein `AttributeRule`-Element, welches über das Attribut `attributeID` die ID des freizugebenden Attributs angibt.

Vor der Modifizierung der aktuellen `attribute-resolver.xml` muss allerdings von dieser eine Kopie erstellt werden, um im Fehlerfall die vorherige Konfiguration wiederherstellen zu können und generell Änderungen an der Datei für Administratoren nachvollziebar zu machen. Hierbei gibt es verschiedene Möglichkeiten. Die einfachste ist es von der zu modifizierenden Datei eine Kopie zu erstellen und deren Dateinamen mit einer Versionsnummer, zum Beispiel dem aktuellen Datum nach dem Unix-Timestamp, zu erweitern. Dazu kann in dem `conf`-Verzeichnis, das bei einer Standard Shibboleth IDP Installation alle Konfigurationsdateien enthält, ein neues Verzeichnis `conf-versions` angelegt werden, in dem die Kopien gespeichert werden.

Eine andere Methode wäre es ein Versionsverwaltungssystem wie CVS, SVN oder GIT zu verwenden. Der größte Vorteil gegenüber dem zuvor vorgeschlagenen Kopieren von Dateien ist es, dass die alten Versionen der Dateien übersichtlicher organisiert sind. Dieser Vorteil spielt allerdings nur dann wirklich eine Rolle, wenn viele Versionen der Dateien angelegt werden, dh. wenn häufig neue Attribute durch Konvertierungsregeln erzeugt werden sollen. Außerdem erhöht die Verwendung einer Versionsverwaltung die Anzahl der Abhängigkeiten der Implementierung, was eventuell Administratoren von der Installation der Erweiterung abschrecken könnte.

Die Entscheidung, welches System für die Versionierung verwendet werden soll, kann abhängig von der jeweiligen Implementierung getroffen werden. Für den Einsatz zum Testen des Systems ist eine dateibasierte Lösung ausreichend, da sie die Anzahl der Abhängigkeiten minimiert und weniger Konfiguration benötigt. Für eine Produktivumgebung, die mehrere Jahre in Betrieb sein soll, ist es auf Dauer vermutlich übersichtlicher, wenn eine Versionsverwaltung verwendet werden kann.

5.4. Gesamtsystem Trusted Third Party

Nachdem in den vorherigen Abschnitten die Synchronisation von Metadaten und Konvertierungsregeln sowie deren Installation auf Seiten von SP und IDP beschrieben wurden, wird in diesem Abschnitt auf die dazu nötigen Systeme an der Trusted Third Party eingegangen. Dazu gehören die Benutzerverwaltung (Abschnitt 5.4.1), Providerverwaltung (Abschnitt 5.4.2), Organisationsverwaltung (Abschnitt 5.4.3) und die Konvertierungsregelverwaltung (Abschnitt 5.4.4).

Für die Organisation der Daten dieser Verwaltungsschnittstellen wird eine Datenbank benötigt. Im Gegensatz zu SP und IDP, wo, aus Gründen weiterer Abhängigkeiten und zusätzlichem Konfigurationsaufwand für die Betreiber, jeweils gegen die Verwendung einer Datenbank argumentiert wurde, lohnt es sich bei der Installation der TTP diese Nachteile in Kauf zu nehmen, um die deutlich größere Menge an Daten, die hier anfallen, zu verwalten. Die für die einzelnen Verwaltungsfunktionen nötigen Datenbanktabellen werden zunächst

in den jeweiligen Abschnitten beschrieben und abschließend, in Abschnitt 5.4.5, zu einem kompletten Datenbankschema zusammengesetzt.

Die Verwaltung der einzelnen Komponenten wird über ein Webinterface gesteuert. Dies hat den Vorteil, dass es universell, von jedem potentiellen Benutzer über einen Standard-Browser verwendet werden kann und keine zusätzliche Hürde durch den Einsatz eigener Software schafft. Ein Webinterface ermöglicht es neuen Benutzern, die das System ausprobieren und einen schnellen Überblick über dessen Funktionen bekommen wollen, sowie erfahrenen Benutzern ihre Konfigurationen durchzuführen. Die Alternative, eigene Software, eventuell in der Form von Skripten zu verwenden, würde „power usern“, die ihre Konfiguration aus der Kommandozeile oder durch eigene Skripte automatisiert durchführen möchten, entgegenkommen und könnte als ein Application Programming Interface (API) zur Verfügung gestellt werden. Diese Beschreibung beschränkt sich allerdings auf das initial wichtigere Webinterface.

5.4.1. Benutzerverwaltung

Die Benutzerverwaltung der TTP benötigt die Möglichkeit, dass sich neue Benutzer bzw. Dienstadministratoren registrieren können und, wie für Webplattformen üblich, ihr Profil verwalten können. Ebenfalls ist es allgemein üblich, dass ein neuer Benutzeraccount zunächst verifiziert werden muss, bevor er Änderungen an dem System vornehmen kann. Eine solche Verifizierung erfolgt häufig darüber, dass überprüft wird, ob die E-Mail-Adresse des Benutzers existiert und der Benutzer darauf zugreifen kann, indem eine Mail mit einem Aktivierungs-Code oder Link an diese geschickt wird. Alternativ kann eine Aktivierung auch durch einen Administrator des Dienstes geschehen. Der Benutzer muss dafür bei der Registrierung angeben, weshalb er den Dienst benutzen möchte und darauf warten, dass der Administrator entscheidet, ob der Benutzer den Dienst verwenden darf.

Eine weitere Möglichkeit, wäre es, dass sich Benutzer über ihren IDP authentifizieren. Dazu würde ein neuer Benutzer bei der Registrierung entweder einen schon vorhandenen IDP auswählen oder gleichzeitig einen neuen IDP registrieren und sich über diesen authentifizieren. Diese Methode hätte den Vorteil, dass für die Verwaltung eines FIM-Systems, dessen Ziel es unter anderem ist, weniger einzelne Accounts verwalten zu müssen, kein neuer Account nötig ist. Allerdings kann diese Methode nur eine Option sein, da eventuell ein Service Provider seinen Dienst registrieren will, ohne selbst einen IDP zu betreiben bzw. einen Account bei einem IDP zu haben.

Da sich die Benutzer also mit einem neuen Account bei der TTP registrieren können müssen, werden folgende Datenbankattribute für einen Benutzer benötigt.

- **id**: Eindeutige ID zur Identifizierung des Benutzers.
- **loginname**: Ein eindeutiger Name, über den der Benutzer beim Login identifiziert wird. Dieser Name kann nachträglich nicht geändert werden, ist nur beim Login zu verwenden und nicht für andere Benutzer sichtbar.
- **username**: Der Name unter dem der Benutzer für andere Benutzer sichtbar ist. Dieser Name muss ebenfalls eindeutig sein, kann jedoch geändert werden.

- **password**: Das Passwort, mit dem der Benutzer sich authentifiziert. Bei der Speicherung ist darauf zu achten, dass das Password nicht im Klartext, sondern durch einen geeigneten Hash oder eine Verschlüsselung geschützt wird.
- **salt**: Um den Aufwand zu erhöhen, aus der Datenbank gestohlene Passwörter zu knacken, sollten diese individuell mit einem Salt geschützt werden.
- **givenName**: Der Vorname des Benutzers kann innerhalb des Webfrontends oder E-Mails zur Anrede des Benutzers verwendet werden.
- **surname**: Wie der Vorname, kann der Nachname dazu verwendet werden das Webfrontend oder E-Mails zu personalisieren.
- **email**: Eine E-Mail-Adresse, unter der der Benutzer erreichbar ist.
- **validated**: Ein Flag, das angibt ob der Benutzer durch die Validierung seiner E-Mail-Adresse oder einer anderen Methode überprüft wurde.
- **created**: Das Datum, an dem der Benutzeraccount erstellt wurde. Dies kann hilfreich sein, um massenhaft angelegte, nicht validierte, Accounts zu entfernen.

5.4.2. Providerverwaltung

Die Verwaltung eines Providers ermöglicht es einem Benutzer neue Provider bei der TTP zu registrieren, sowie Einstellungen für vorhandene Provider zu ändern. Damit sichergestellt ist, dass ein Benutzer nur Provider modifiziert, bei denen er dazu berechtigt ist, wird ein System benötigt, mit dem ein Benutzer nachweisen kann, dass er für einen Provider zuständig ist. Dieses System wird im ersten Teilabschnitt betrachtet. Darauf wird in Abschnitt 5.4.2.2 beschrieben, wie ein neuer Provider hinzugefügt werden kann und in Abschnitt 5.4.2.3, wie die Metadaten für einen Provider verwaltet werden. Abschließend wird in Abschnitt 5.4.2.4 gezeigt, welche Datenbank-Attribute daher für einen Provider benötigt werden.

5.4.2.1. Verbinden eines Providers mit einem Benutzer

Sowohl nach dem Erstellen eines neuen Providers als auch für das Zuordnen eines bestehenden Providers zu einem Benutzer, muss der Benutzer in der Lage sein nachzuweisen, dass er legitimiert ist, Einstellungen für einen Provider vorzunehmen.

Die einfachste Möglichkeit hierfür ist es aus der EntityID des neuen Providers die Domain des Providers auszulesen und von dem Benutzer zu verlangen, dass er eine Datei mit einem zufällig generierten Namen auf der Domain erstellt. Dazu sollte nur ein für diese Domain und dadurch auch den Provider zuständiger Administrator fähig sein. Dieses Konzept kann alternativ zu der Abfrage einer bestimmten Datei per HTTP über einen DNS TXT-Record in der Domain implementiert werden. Eine weitere Möglichkeit ist es von dem Benutzer zu verlangen, dass er einen zufällig generierten Text mit dem privaten Schlüssel des Providers signiert. Da der öffentliche Schüssel in den Metadaten enthalten ist, kann die TTP damit überprüfen, ob der Benutzer Zugriff auf den privaten Schlüssel des Providers hat. Diese Art der Überprüfung funktioniert allerdings nur, wenn der öffentliche Schlüssel in den Metadaten des Providers, Teil eines Zertifikates ist, dass von einer Partei unterschrieben wurde, der die

TTP vertraut. Ansonsten könnte ein Benutzer ein eigenes Schlüsselpaar für eine beliebige Domain erstellen.

Die sicherste Methode wäre die Überprüfung der Signatur eines zufälligen Textes in Verbindung mit einem von einer vertrauenswürdigen Certificate Authroity (CA) signierten Zertifikat. Da das System allerdings für möglichst viele Provider offen sein soll und das Erstellen eines Zertifikates häufig mit hohen Kosten und Verwaltungsaufwand verbunden ist, sollte die einfachere Domainvalidierung ausreichend sein.

Neben diesen Methoden, bei denen der Benutzer direkt nachweist, dass er Kontrolle über die Domain hat, ist es auch möglich, dass ein Benutzer, der schon für einen Provider registriert ist, seine Rechte an einen anderen Benutzer delegiert.

5.4.2.2. Hinzufügen eines Providers

Mit der Providerverwaltung kann ein registrierter und validierter Benutzer unter anderem einen neuen Identity oder Service Provider erstellen. Abbildung 5.6 zeigt den dafür vorgesehenen Workflow. Entsprechend der SAML-Semantik wird ein Provider über seine EntityID identifiziert, wodurch diese eindeutig zu einem bestimmten Provider gehört. Bei der Registrierung eines neuen Providers muss daher überprüft werden, ob ein Provider mit der EntityID des neuen Providers schon existiert.

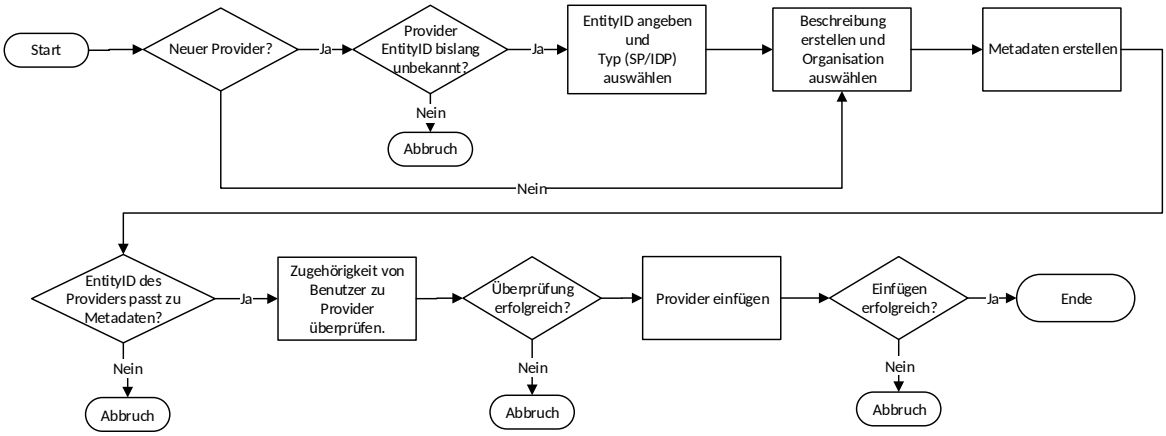

Abbildung 5.6.: Workflow für die Registrierung eines neuen Providers

Neben der EntityID muss für einen Provider angegeben werden, ob es sich um einen Identity Provider oder Service Provider handelt. Zusätzlich kann der Benutzer den Dienst durch einen kurzen Text beschreiben und auswählen, welcher Organisation der Provider angehören soll.

Nachdem der Provider mit diesen grundlegenden Informationen angelegt wurde, muss der Benutzer, wie im vorherigen Abschnitt beschrieben, nachweisen, dass er die Kontrolle über den Provider hat. Daraufhin kann er die Eigenschaften des Providers, mit Ausnahme der eindeutigen und unveränderbaren EntityID, editieren. Das Hinzufügen und Editieren von Metadaten wird im folgenden Abschnitt beschrieben.

5.4.2.3. Hinzufügen & Editieren von Metadaten

Die Verwaltung von Metadaten soll es einem Benutzer ermöglichen, neue Metadaten hochzuladen sowie vorhandene Metadaten zu modifizieren. Dazu sollen die verschiedenen Versionen der Metadaten gespeichert werden, so dass Änderungen an den Metadaten vorgenommen werden können, ohne dass diese sofort nach außen sichtbar sind. Außerdem ist es dadurch möglich nach dem Aktualisieren von Metadaten mit einer fehlerhaften Version schnell zu der zuvor funktionierenden Version zurückzukehren.

Ähnlich zu der Betrachtung für die Speicherung der Metadaten auf Seite von SP und IDP gibt es für die TTP drei Möglichkeiten die Metadaten zu speichern. Zum einen, da für die TTP eine Datenbank vorgesehen ist, könnten die Metadaten als Binary Large Object (BLOB) direkt mit den zu den Metadaten gespeicherten Eigenschaften in der Datenbank gespeichert werden. Der Nachteil dieser Lösung ist, dass für jeden Abruf der Metadaten das BLOB ausgelesen werden muss. Bei der zweiten Methode, der dateibasierten Speicherung, ergibt sich der Vorteil, dass in der Datenbank nur ein Feld für den Speicherort angegeben werden muss. Wie in Abschnitt 5.2.2 beschrieben, könnte dieser Speicherort sowohl eine lokale Datei als auch eine URL enthalten und wäre daher flexibler. Die Metadatendateien können auch direkt in einem Verzeichnis abgelegt werden, auf das aus dem Internet über HTTP zugegriffen werden kann. Eine Erweiterung der dateibasierten Methode, wäre es auch hier für die Verwaltung der Versionen eine entsprechende Versionsverwaltungssoftware einzusetzen. Im Gegensatz zu der Betrachtung für Konvertierungsregeln in Abschnitt 5.3.3, würde eine Versionsverwaltung allerdings keinen großen Vorteil bieten, da die Datenbank schon als Managementinterface Referenzen auf die einzelnen Versionen der Metadaten hält und somit eine Übersicht für die Dateien bildet. Im Folgenden soll daher mit einer einfachen dateibasierten Speicherung gearbeitet werden.

Die Dateinamen werden dabei wie bei SP und IDP aus dem SHA1-Hash der EntityID generiert. Da es mehrere Versionen einer Metadatendatei geben kann, wird jeder Datei das aktuelle Datum als Unix-Timestamp angehängt.

Der Workflow, der zum Hinzufügen und Modifizieren von Metadaten verwendet wird, ist in Abbildung 5.7 dargestellt. Beim Hinzufügen bzw. Modifizieren der Metadaten muss zunächst durch die TTP sichergestellt werden, dass die Metadaten nach wie vor nur die EntityID des dazugehörigen Providers enthalten und der Benutzer berechtigt ist, für den Provider Änderungen vorzunehmen. Falls es sich nicht um komplett neue Metadaten handelt, sondern um eine Modifikation bestehender Metadaten, muss die Beziehung der Metadatenversionen in der Datenbank dargestellt werden. Hierzu ist es am einfachsten, jedem Metadaten-Eintrag ein `parent`-Attribut zuzuweisen, das sofern es nicht leer ist, angibt, von welcher Metadatenversion diese Version abgeleitet wurde. Durch diese Struktur kann ein Versionsbaum aufgebaut werden. Zusätzlich kann jede Version der Metadaten mit einem Kommentar versehen werden, der beschreibt, warum eine bestimmte Änderung durchgeführt wurde.

Nachdem Metadaten für einen Provider erstellt wurden, können diese aktiviert werden. Die Aktivierung findet nicht automatisch statt, um es Administratoren zu ermöglichen geänderte Metadaten erst zu testen, bevor sie von der TTP verwendet werden.

In der Datenbank benötigen Metadaten daher folgende Attribute:

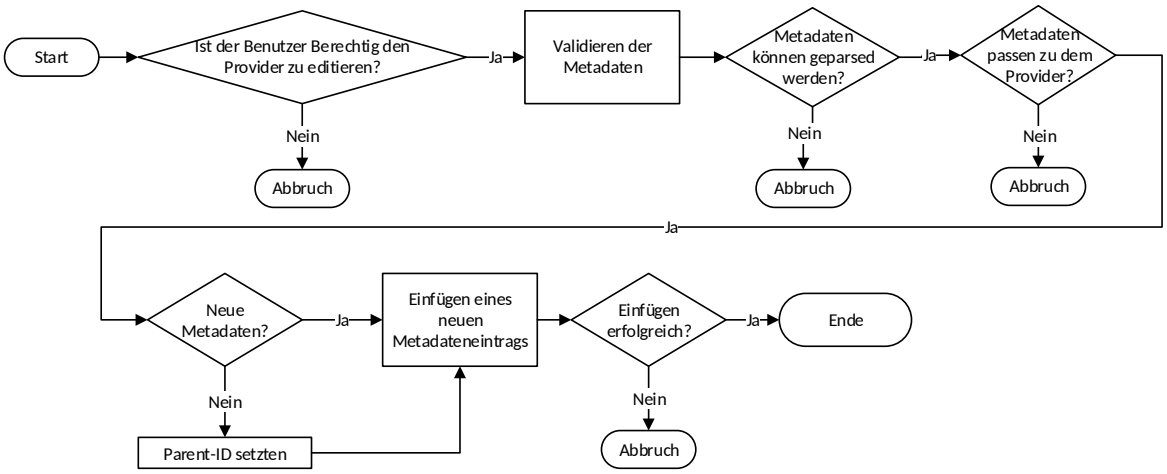

Abbildung 5.7.: Workflow für das Hinzufügen neuer/geänderter Metadaten

- **id**: Zur eindeutigen Identifizierung von Metadaten.
- **entityID**: Die EntityID des durch diese Metadaten beschriebenen Dienstes.
- **location**: Speicherort der Metadaten. Wie in Abschnitt 5.2.1 beschrieben, kann dies ein lokaler Pfad oder eine URL sein.
- **comment**: Ein optionaler Kommentar, der die Version bzw. die Änderung an den Metadaten beschreibt.
- **owner**: Verweis auf den Benutzer, der die Metadaten angelegt hat.
- **parent**: Verweis auf die Metadaten, von denen diese abgeleitet sind.
- **created**: Datum, zu dem die Metadaten erstellt wurden. Anhand dieses Datums kann festgestellt werden, wann Änderungen an den Metadaten durchgeführt wurden.

5.4.2.4. Provider-Modell

Aus den vorherigen Abschnitten ergibt sich, dass ein Provider über die folgenden Attribute verfügen muss. Da es mehrere Benutzer geben kann, die für einen Provider eine bestimmte Rolle ausführen, gibt es eine zusätzliche Auflösungstabelle, die Provider und die für den Provider registrierten Benutzer und ihre Rollen verbindet.

- **id**: Eine eindeutige ID, die den Provider identifiziert.
- **type**: Ein Flag zur Unterscheidung zwischen SPs und IDPs.
- **entityID**: Die EntityID des Providers, die eindeutig sein muss. Es gibt kein Szenario bei dem es erwünscht wäre, dass mehrere Providereinträge zu einer EntityID existieren.
- **description**: Eine kurze Beschreibung des Providers.
- **organization**: Eine Referenz zu einer Organisation, der der Provider angehört.

- **metadata**: Eine Referenz auf die Metadaten, die gerade für diesen Dienst aktiv sein sollen.
- **created**: Ein Datum, das angibt, wann dieser Provider erstellt wurde.

5.4.3. Organisationsverwaltung

Die Organisationsverwaltung ist sehr ähnlich zu der Verwaltung der Provider. Allerdings ist es für Organisationen nicht möglich, einfach über die Domain nachzuweisen, dass ein Benutzer berechtigt ist, Aussagen über diese Organisation zu treffen. Daher bleibt für die Organisationen nur der Weg, dass sie sich über ein Formular bei dem Betreiber der TTP anmelden müssen und mit diesem über andere Kommunikationskanäle beweisen müssen, dass sie für eine bestimmte Organisation tätig sind. Zum Beispiel gibt es die Möglichkeit für den TTP-Betreiber zu überprüfen, ob ein offizieller Mailaccount der Organisation oder einer ihrer Ansprechpartner verwendet wurde. Diese an sich simple Überprüfung lässt sich allerdings nur schwer automatisieren.

Für die Zuordnung von Benutzern, die für eine Organisation zuständig sind, wird wie für die Provider eine Auflösungstabelle verwendet, die angibt welcher Benutzer welche Rolle bei welcher Organisation hat.

- **id**: Eine eindeutige ID, die die Organisation identifiziert.
- **name**: Der vollständige Name der Organisation.
- **displayName**: Eine kurze Version des Organisationsnamens, der verwendet wird, falls der vollständige Name zu viel Platz in Anspruch nehmen würde.
- **description**: Eine kurze Beschreibung der Organisation.
- **url**: Ein Link zu der Webseite der Organisation.

5.4.4. Konvertierungsregelverwaltung

Zur Speicherung der Konvertierungsregeln bei der TTP wird, wie für die Metadaten im vorhergehenden Abschnitt beschrieben, eine Datenbank für die Eigenschaften der Konvertierungsregeln und das Dateisystem für die Speicherung der eigentlichen Regeldateien verwendet. Anstatt des SHA1-Hashes der EntityID wird hier der Hash des Namens der Regel verwendet. Das Anhängen der aktuellen Zeit als Unix-Timestamp wird für die verschiedenen Versionen beibehalten.

Die Verwaltung der Konvertierungsregeln erfolgt über ein Webinterface. Dort können Administratoren Regeln hochladen sowie nach bestehenden Regeln suchen und diese verbessern und bewerten. Der in Abbildung 5.8 dargestellte Ablauf orientiert sich dabei stark an dem Ablauf für das Hinzufügen und Modifizieren von Metadaten.

Beim Hinzufügen einer neuen Regel muss zunächst ein Name für die Regel vergeben werden sowie die benötigten Quellattribute und das Zielattribut ausgewählt werden. Diese drei Parameter sollen nachträglich nicht mehr geändert werden können. Das Ändern des Namens könnte zu Verwirrungen darüber führen, welche Regel nun gemeint ist und Änderungen an

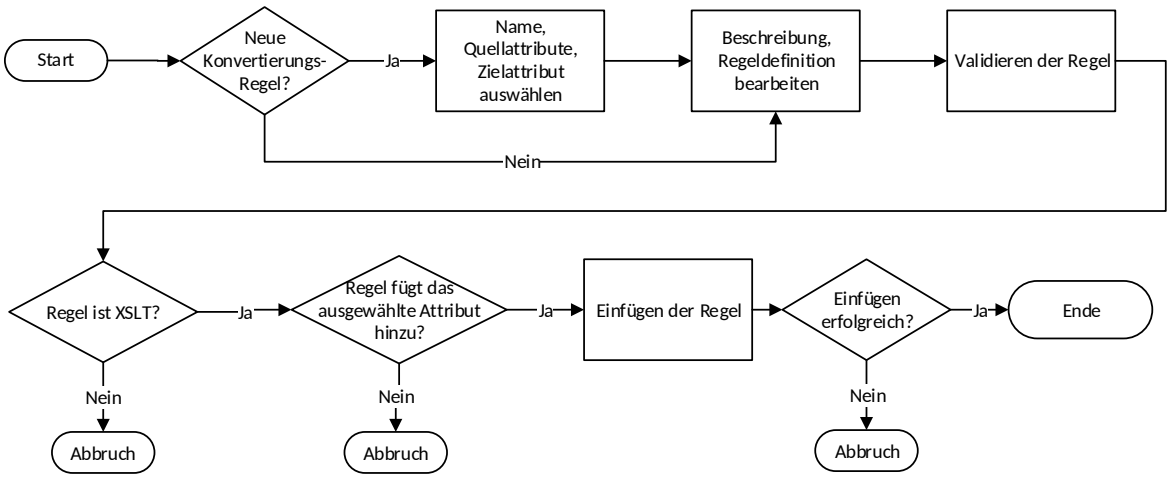

Abbildung 5.8.: Workflow für das Hinzufügen einer neuen/geänderten Konvertierungsregel

den Quellattributen, eine neuere Version einer Regel, wegen danach fehlender Attribute, eventuell nicht mehr für einen IDP anwendbar machen. Auch das Zielattribut muss konstant bleiben, um beim Aktualisieren von Regeln, nicht deren ursprüngliches Ziel zu verfehlen.

Danach können die Beschreibung der Regel sowie die Regeldefinition erstellt bzw. bearbeitet werden. Hier macht es Sinn, für neue Versionen nachträglich Änderungen zuzulassen. So kann der Beschreibungstext über die Regel an sich und die durchgeführten Aktualisierungen informieren und eventuelle Fehler in der Konvertierungsregel beseitigt werden.

Daraus ergeben sich für eine Konvertierungsregel folgende Datenbankattribute:

- **id**: Eine eindeutige ID, die die Konvertierungsregel identifiziert.
- **name**: Ein die Regel beschreibender Name.
- **description**: Eine kurze Beschreibung für die Regel, die gegebenenfalls Notizen für durchgeführte Aktualisierungen enthalten kann.
- **target**: Das Attribut, das durch diese Regel erstellt wird.
- **owner**: Der Benutzer, der die Regel erstellt hat. Eine Regel ist immer genau einem Benutzer zugeordnet, so dass hier keine Auflösungstabelle wie bei Providern und Organisationen benötigt wird. Jeder andere Benutzer kann eine Modifikation einer Regel erstellen.
- **location**: Ein Verweis auf den Speicherort der Regel.
- **parent**: Ein Verweis auf die Regel, von der diese Regel abgeleitet wurde oder null.
- **status**: Ein Flag, dass den Status der Regel, angibt. Hierbei kann angegeben werden, ob diese Regel verwendet werden soll, oder ob sie nur zu Dokumentationszwecken existiert oder unvollständig ist.
- **created**: Das Datum, an dem die Regel erstellt wurde.

Um verschiedene Regeln, die identische Attribute erstellen, zu vergleichen, soll eine Bewertung der Regeln durch den Benutzer möglich sein. Dazu wird eine weitere Tabelle erstellt, die Paare aus Benutzern und Regeln mit der dazugehörigen Bewertung versieht.

5.4.5. Resultierendes Datenbankschema

Aus den in den vorhergehenden Abschnitten beschriebenen Schnittstellen und deren Anforderungen ergibt sich das in Abbildung 5.9 dargestellte Datenbankschema. In der Abbildung sind die Primärschlüssel jeder Tabelle fett gedruckt. Außerdem geben die eingezeichneten Pfeile an, welches Attribut sich auf welche ID bezieht.

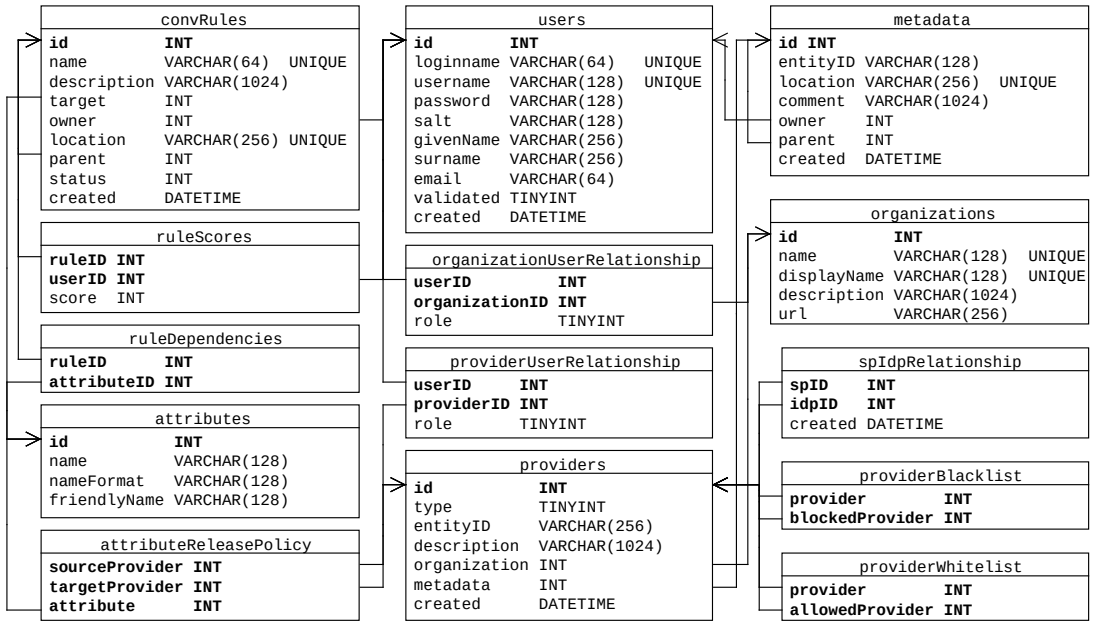

Abbildung 5.9.: Übersicht über das Datenbankschema

5.5. Vergleich mit den Anforderungen

Das in diesem Kapitel entwickelte Konzept wird nun mit den in Kapitel 3 aufgestellten Anforderungen verglichen. Zunächst werden dazu im nächsten Abschnitt die funktionalen Anforderungen und im darauffolgenden Abschnitt 5.5.2 die nichtfunktionalen Anforderungen betrachtet.

5.5.1. Funktionale Anforderungen

F1: Dynamischer Verbindungsaufbau: Die Erweiterung des Centralized Discovery Services erlaubt es, sofern nicht durch Access Control Lists eingeschränkt, dynamisch Verbindungen,

zwischen den an der TTP teilnehmenden Parteien, aufzubauen. Diese Anforderung wird daher erfüllt. (Abschnitt 5.1.3)

F2: Trusted Third Party: Der erweiterte Centralized Discovery Service dient in dem System als Trusted Third Party, wodurch diese Anforderung erfüllt wird. (Abschnitt 5.1.3)

F3: TTP-Anmeldung: Das System enthält eine eigenständige Benutzerverwaltung, die es Administratoren erlaubt selbstständig einen neuen Account zu Registrieren und zu Modifizieren. Dadurch wird diese Anforderung erfüllt. (Abschnitt 5.4.1)

F4: Zuständigkeit: Das Konzept stellt mehrere Methoden (Domainvalidierung, Zertifikatsvalidierung) vor, über die ein Administrator seine Zuständigkeit für einen Dienst nachweisen kann. (Abschnitt 5.4.2.1)

F5: Zugriffsberechtigungen: Die Zugriffsberechtigung für das Modifizieren von zum Beispiel Providerinformationen ist an den vorherigen Nachweis der Zuständigkeit gekoppelt, wodurch diese Anforderung erfüllt wird. (Abschnitt 5.4.2.1)

F6: Dienstverwaltung: Ein registrierter Benutzer kann sowohl neue Dienste zu dem System hinzufügen, als auch, nachdem er bewiesen hat, dass er für einen Dienst zuständig ist, diesen Modifizieren und auch Löschen. Eine Übertragung an einen Nachfolger, wie sie in der Anforderung verlangt wird, ist in diesem Konzept nicht notwendig, da mehrere Benutzer gleichzeitig für einen Dienst zuständig sein können. Der Nachfolger kann daher jederzeit die gleiche Kontrolle über den Dienst bekommen. Daher ist diese Anforderung erfüllt. (Abschnitte 5.4.2.2, 5.4.2.3 und 5.4.2.1)

F7: Kompatibilität: Das Konzept baut auf weit verbreiteten Shibboleth-Implementierungen auf, wodurch die Basisfunktionen kompatibel bleiben und diese Anforderung erfüllt wird. (Abschnitt 5.1.3)

F8: Fehlertoleranz: Das entworfene System setzt mit der Verwendung von HTTP als Transportmechanismus und XML für die Nachrichten auf die gleichen Technologien wie Shibboleth. Da es für diese Techniken robuste Implementierungen gibt, erfüllt das Konzept diese Anforderung. (Abschnitt 5.1.3)

F9: Metadatensynchronisation: Das Konzept beinhaltet die Synchronisation von Metadaten zwischen sich vorher unbekannten Diensten, wodurch diese Anforderung erfüllt wird. (Abschnitte 5.2.1 und 5.2.3)

F10: Benutzerinitiierter Verbindungsaufbau: Der Verbindungsaufbau wird durch das Konzept dann initiiert, wenn ein Benutzer einen Dienst verwenden will, der keine direkte Verbindung zu dem IDP des Benutzers hat, aber Teil des TTP-Netzwerkes ist. Dadurch ist diese Anforderung erfüllt. (Abschnitt 5.1.3)

F11: IDP Auswahl: Die Auswahl des IDP geschiet in dem Konzept durch einen modifizierten EDS, der alle aktuell verbundenen IDPs anzeigt und eine Option hat, die IDP-Suche bei Bedarf zur TTP zu eskalieren. (Abschnitt 5.1.3)

F12: Attributsidentifikation: Für die Identifikation von Attributen werden die selben Verfahren wie bei Shibboleth verwendet, wodurch diese Anforderung erfüllt wird.

F13: Konvertierungsregelverwaltung: Das Konzept sieht eine Schnittstelle vor, durch die Benutzer Konvertierungsregeln suchen, anzeigen, erstellen, modifizieren sowie bewerten können. Daher ist diese Anforderung erfüllt. (Abschnitt 5.4.4)

F14: Konvertierungsregelbeschreibung: Die Konvertierungsregeln werden mit Metainformationen wie Namen, Beschreibung, erstellender Benutzer, Versionshirarchie sowie Quell- und Ziel-Attributen in der Datenbank gespeichert. Daher ist diese Anfoderung erfüllt. (Abschnitt 5.4.4)

F15: Konvertierungsregelbewertung: Über einen Score kann jeder Benutzer die Konvertierungsregeln bewerten. Ein einzelner Score unterscheidet nicht zwischen den in der Anforderung genannten Werten Vollständigkeit, Korrektheit und Zuverlässigkeit, ist allerdings weniger komplex und einfacher zu interpretieren und zu vergleichen. Daher ist diese Anforderung erfüllt. (Abschnitt 5.4.4)

F16: Konvertierungsregeloperationen: Die Konvertierungsregeln basieren auf den von Shibboleth verwendeten Attributs-Definitionen, die alle nötigen Operationen unterstützen. Diese Anforderung ist daher erfüllt. (Abschnitt 5.3.3)

F17: Konvertierungsregelabgleich: Das Konzept definiert einen Ablauf, mit dem ein IDP Konvertierungsregeln automatisiert suchen kann und diese sofort installieren kann. Dadurch wird diese Anforderung erfüllt. (Abschnitt 5.3.2)

F18: Fehlerbenachrichtigung: Eine explizite Fehlerbenachrichtigung wird durch das Konzept nicht spezifiziert. Die Art der Fehlerbenachrichtigung wird dadurch implementationsabhängig gemacht und ist nicht erfüllt.

F19: Attribute Release Policies: Bei dem Abgleich von Konvertierungsregeln, sieht das Konzept die Integration von ARPs vor, um zu verhindern, dass ein IDP Regeln installiert, die Attribute erzeugen, die der IDP-Administrator nicht freigeben möchte. (Abschnitt 5.3.3)

F20: Access Control Lists: Durch eine Filterung vor der Übertragung der Metadaten können einzelne Providerverbindungen verboten oder erlaubt werden. Dadurch wird diese Anforderung erfüllt. (Abschnitt 5.1.3)

F21: Attributsfreigabe: Da das System in Shibboleth integriert ist, funktioniert die dort vorhandene Lösung uApprove auch in dem durch dieses Konzept vorgeschlagenen Szenario. Die Anforderung wird daher durch ein Plugin erfüllt.

F22: Auditierbarkeit: Die Auditierbarkeit von Shibboleth SP und IDP wird durch diese Erweiterung nicht eingeschränkt. Alle vorher möglichen Logeinträge können weiterhin generiert werden, zusätzlich werden implementierungsabhängig neue Logeinträge für die Synchronisation von Metadaten und Konvertierungsregeln erstellt. Diese Anforderung wird daher erfüllt.

F23: Testsysteme: Das Konzept sieht keine konkreten Testsysteme vor, so dass das Betreiben solcher Systeme durch die jeweiligen Betreiber der Föderation organisiert werden muss. Diese Anforderung wird daher nicht erfüllt.

F24: Kompatibilitätstests: Das Konzept sieht keine theoretischen Kompatibilitätstest vor, so dass die einzige Methode zu überprüfen, ob ein IDP die von einem SP geforderten Attribute (unter Berücksichtigung eventuell zu installierender Konvertierungsregeln) erbringen kann, ist, den Verbindungsaufbau auszuprobieren. Diese Anforderung wird daher nicht erfüllt.

F25: Schemaerweiterungen: Durch die Verwendung von Shibboleth als Grundlage der Erweiterung bleibt es möglich, eigene Schemata zu definieren und zu verwenden. Diese Anforderung wird daher erfüllt.

5.5.2. Nichtfunktionale Anforderungen

Da es sich bei dem hier Betrachteten um ein Konzept ohne Implementierung handelt, sind einige Anforderungen, die von der Implementierung abhängig sind, wie zum Beispiel Kosten und OpenSource nicht anwendbar und werden daher nicht betrachtet.

NF5: Anwendbarkeit: Das Konzept baut auf Shibboleth auf und verwendet dessen Funktionen als Grundlage. Da Shibboleth diese Anforderung, wie in 4.1.4.2 dargestellt, erfüllt, wird sie auch von dieser Erweiterung erfüllt.

Anforderung	Erfüllt			Relevanz
	ja	mit Plug-in	nein	
F1: Dynamischer Verbindungsaufbau	✓			1
F2: Trusted Third Party	✓			1
F3: TTP-Anmeldung	✓			1
F4: Zuständigkeit	✓			1
F5: Zugriffsberechtigungen	✓			1
F6: Dienstverwaltung	✓			1
F7: Kompatibilität	✓			2
F8: Fehlertoleranz	✓			2
F9: Metadatensynchronisation	✓			1
F10: Benutzerinitiierter Verbindungsaufbau	✓			1
F11: IDP Auswahl	✓			1
F12: Attributsidentifikation	✓			1
F13: Konv.-Regelverwaltung	✓			1
F14: Konv.-Regelbeschreibung	✓			1
F15: Konv.-Regelbewertung	✓			2
F16: Konv.-Regeloperationen	✓			1
F17: Konv.-Regelabgleich	✓			1
F18: Fehlerbenachrichtigungen			✓	3
F19: Attribute Release Policies:	✓			2
F20: Access Control Lists	✓			2
F21: Attributsfreigabe		✓		2
F22: Auditierbarkeit	✓			3
F23: Testsysteme			✓	3
F24: Kompatibilitätstests			✓	3
F25: Schemaerweiterungen	✓			2

Tabelle 5.1.: Vergleich der funktionalen Anforderungen mit dem TTP-Konzept

NF6: Protokoll: Das Konzept spezifiziert für die Schnittstellen der TTP die jeweils erwarteten und gesendeten Nachrichten, so dass diese Anforderung erfüllt wird.

NF7: Verlässlichkeit der SPs: Ebenso, wie bei der zugrundeliegenden Shibboleth-Implementierung, lässt sich die Verlässlichkeit der SPs nicht durch das Konzept an sich bestimmen.

NF8: Authentifizierungsverantwortung: An der Übertragung der Authentifizierungsverantwortung ändert sich durch die Erweiterung nichts, so dass diese Anforderung weiterhin erfüllt bleibt.

NF9: Datenaktualität: Wie bei Shibboleth selbst ist die Aktualität der Daten nicht durch die Implementierung, sondern nur von organisatorischen Maßnahmen abhängig, so dass diese Anforderung nicht bestimmbar ist.

NF10: User-Experience-Konsistenz: Das User-Interface der Erweiterung ist primär implementierungsabhängig, unterscheidet sich aber nicht wesentlich von der, der Shibboleth

Implementierung. Der Benutzer muss nun zunächst beim SP seinen IDP suchen und wenn er dort nicht vorhanden ist, die Suche bei der TTP wiederholen. Der zweite Schritt war bisher nicht vorgesehen, stellt aber keine große Abweichung von dem bekannten Ablauf dar. Daher wird diese Anforderung erfüllt.

NF11: Zugang zu einer TTP: Der Zugang zu einer TTP ist primär abhängig von dem Betreiber der TTP, das Konzept schlägt, in Anlehnung an das offene Konzept von OpenID Connect, ein möglichst offenes System vor. Daher ist diese Anforderung erfüllt.

Anforderung	**Erfüllt**		
	ja	nein	**Relevanz**
NF5: Anwendbarkeit	✓		2
NF6: Protokoll	✓		1
NF7: Verlässlichkeit der SPs	–	–	1
NF8: Auth.-Verantwortung	✓		1
NF9: Datenaktualität	–	–	2
NF10: User-Experience-Konsistenz	✓		3
NF11: Zugang zu einer TTP	✓		1

Tabelle 5.2.: Vergleich der nichtfunktionalen Anforderungen mit dem TTP-Konzept

6. Implementierung

In diesem Kapitel wird die konkrete Implementierung eines Prototypen entsprechend des im vorherigen Kapitels entwickelten Konzeptes beschrieben. Dabei wird in fünf Blöcken auf die Implementierung der einzelnen Teilkomponenten für Service Provider (Abschnitt 6.1), Identity Provider (Abschnitt 6.2), Centralized Discovery Services (Abschnitt 6.3) sowie des Embedded Discovery Services (Abschnitt 6.4) und des TTP-Web-Services (Abschnitt 6.5) eingegangen.

Die Teilmodule sind so konzipiert und entwickelt worden, dass sie als Erweiterungen in bestehende Installationen eingefügt werden können. Besonders für SP und IDP ist diese Eigenschaft wichtig, da es den Installationsaufwand der Erweiterung minimiert und erlaubt, eine TTP einzusetzen, ohne schon bestehende Konfigurationen für einzelne IDPs/SPs bzw. Föderationen verändern oder migrieren zu müssen.

In abgebildeten Funktionen sind ausgelassene Codesegmente durch einen Kommentar mit dem Inhalt `[...]` gekennzeichnet. Außerdem sind häufig Kommentare und Fehlerbehandlungen nicht abgebildet. Der komplette Code ist auf der, gedruckten Arbeit beiligenden CD enthalten sowie unter der Adresse `http://gntb08.srv.lrz.de` veröffentlicht.

6.1. Service Provider

Der Shibboleth Service Provider ist in C/C++ implementiert und kann als Authentifizierungs-Modul in den Apache Webserver oder den Microsoft IIS integriert werden. Die hier implementierte Version ist allerdings nur auf dem weit verbreiteten Apache2 Webserver entwickelt und getestet worden. Die Implementierung des SPs besteht im Prinzip aus zwei Teilen. Zum einen dem „Shibboleth SP Lite“-Teil, der in den jeweiligen Webserver als Shared Library integriert wird und zum anderen, dem Programm `shibd`, das den Großteil der Verarbeitung von Anfragen übernimmt. Die Erweiterung ist ebenfalls eine Shared Library, die durch das Shibboleth-Apache-Modul in den Webserver und von dem Shibd-Prozess geladen wird.

Diese Aufteilung hat zum einen den Vorteil, dass bei dem Webserver eingehende Requests dort vorbehandelt, zum Beispiel unzulässige Anfragen direkt geblockt werden können, und reduziert die Anzahl der Abhängigkeiten, die ebenfalls in den Prozess des Webservers geladen werden müssen. Zum anderen erhöht diese Aufteilung die Komplexität des Systems und fördert durch die gewählte Architektur die Unübersichtlichkeit. Beide Teilmodule teilen sich einen Großteil der verwendeten Quell-Dateien, so dass dort häufig innerhalb von Funktionen über Präprozessoranweisungen wie `#ifdef` und regulären If-Abfragen gesteuert wird, welcher Programmcode in welcher kompilierten Library existiert und ausgeführt wird. Zusätzlich muss, für verschiedene Betriebssystem-Umgebungen, eine Kommunikation zwischen den beiden Programmteilen hergestellt werden. Der Shibboelth SP implementiert hierbei die folgenden Methoden der Interprozesskommunikation (IPC): Berkley Sockets, Linux Sockets und TCP/IP.

Zur Übersicht zeigt Abbildung 6.1 den abstrakten Aufbau des Systems. Die Namen der Teilprozesse, in der Grafik mit doppelten „Seiten“ dargestellt, entsprechen den C++ Klassennamen der Implementierung. In den Apache2 Prozess wird das Modul `mod_shib`, welches in der Datei `mod_apache.cpp` implementiert wird, eingebunden. Durch die Konfiguration einer geschützten Seite, wird diesem Modul die Behandlung eines eingehenden Requests auf diesen geschützten Bereich überlassen. Das Modul stellt daraufhin durch den `ServiceProvider` fest, welcher Shibboleth-Handler die Anfrage beantworten soll. In diesem Fall ist hier der `SAML2SessionInitiator` gezeigt, der für das Erstellen eines neuen AuthenticationRequestes zuständig ist. Dieser Handler ist ein `RemotedHandler`, was bedeutet, dass er Teile der Verarbeitung eventuell nicht selbst durchführen kann und diese an den Shibboleth Prozess `shibd` weiterleitet. Wie im vorherigen Absatz beschrieben, gibt es verschiedene Methoden, d.h. Implementierungen des `ListenerService`, die verschiedene IPC-Methoden verwenden, um Nachrichten zwischen den Prozessen auszutauschen.

Die `run()`-Methode des `SAML2SessionInitiators` bestimmt, ob eine Nachricht an den `shibd`-Prozess geschickt werden muss. Für die Erstellung eines neuen AuthenticationRequestes ist dies der Fall, so dass eine entsprechende Nachricht beim `shibd`-Prozess eingeht, die dort in einem eigenen Thread verarbeitet wird. Im `shibd`-Prozess wird durch den `ListenerService` zunächst festgestellt, welcher Handler für die Bearbeitung zuständig ist und die Nachicht dann über die `receive()`-Methode an diesen übermittelt. Die `receive()`-Methode des `SAML2SessionInitiators` modifiziert dann das als Pointer übergebene Response-Objekt, welches nach der Bearbeitung durch den `ListenerService` wieder zurück an den Apache2-Prozess geschickt wird. Dort wird die Response dann zur Anzeige einer Seite oder einer Weiterleitung ausgewertet.

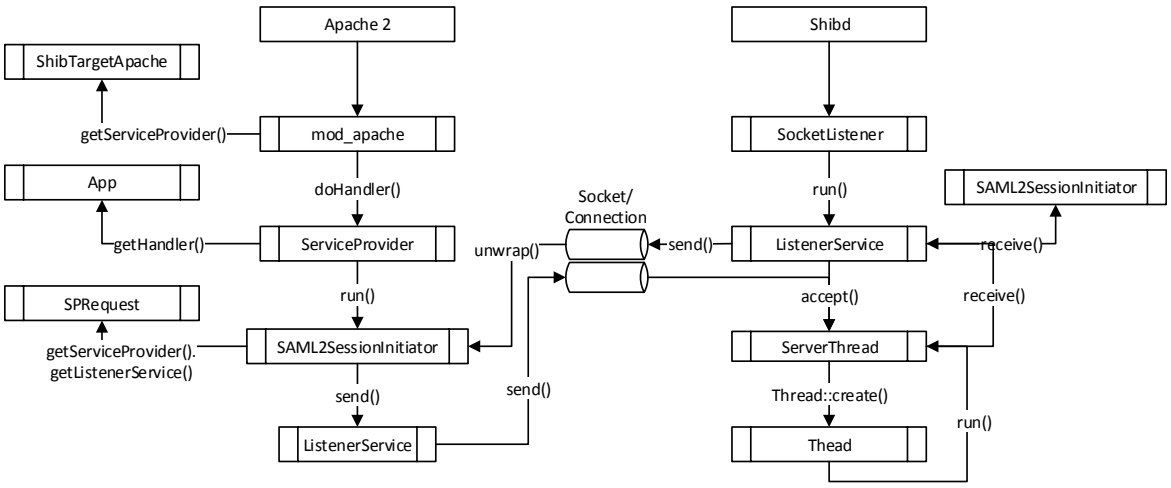

Abbildung 6.1.: Vereinfachte Übersicht eines typischen Ablaufs, zur Behandlung einer Anfrage, der C/C++ Implementierung des Service Providers

Für die Erweiterung des Service Providers, entsprechend des im vorherigen Kapitels entworfenen Konzeptes, muss ein MetadataProvider, der die von der TTP synchronisierten Metadaten verwaltet, ein entsprechender MetadataSyncHandler, der die Synchronisation der Metadaten mit der TTP abstimmt, sowie ein SessionInitiator, der eine neue Session über die TTP startet, implementiert werden. Sowohl der MetadataSyncHandler als auch der SessionInitiator werden, paralell zu dem SAML2SessionInitiator eingefügt. Der MetadataProvider, ist in Abbildung 6.1 nicht abgebildet und wird von den Handlern, wie dem SAML2SessionInitiator, aufgerufen. Diese Teilprogramme werden in dieser Reihenfolge den nächsten Abschnitten beschrieben.

6.1.1. MetadataProvider

Der MetadataProvider für die Verwendung mit einer TTP wird TTPMetadataProvider genannt und basiert auf der Implementierung des DynamicMetadataProviders. Die komplette Implementation des MetadataProviders ist in den mit zusätzlicher Dokumentation versehenen Dateien `src/TTPMetadataProvider.h` und `src/TTPMetadataProvider.cpp` zu finden. Im Folgenden werden nur einige Schlüsselstellen beschrieben. Die vollständige Vererbungshierarchie zeigt Abbildung 6.2. Darin ist zu sehen, wie verschiedene Funktionen für einen MetadataProvider durch die Vererbungskette definiert werden. Der Basis `MetadataProvider` muss nur Funktionen implementieren, die das Hinzufügen und Entfernen von Filtern, sowie das Abrufen der Metadaten bzw. einzelner EntityDescriptoren bzw. EntitiesDescriptoren ermöglicht. Darauf aufbauend definiert der `ObservableMetadataProvider` Funktionen, die es anderen Programmteilen ermöglichen, sich als Beobachter (Observer) bei dem MetadataProvider zu registrieren und bei Änderungen an den Metadaten benachrichtigt zu werden. Der davon abgeleitete `AbstractMetadataProvider` implementiert davon einige Funktionen und definiert zwei neue `resolve()` Funktionen, mit unterschiedlichen Parametern. Diese Funktionen übernehmen das Suchen der Metadaten nach verschiedenen Kriterien, wie zum Beispiel der EntityID. Der darauf aufbauende `DynamicMetadataProvider` führt einen

Locking-Mechanismus ein, bei dem ein anderer Prozessteil ein Lese-Lock setzen kann, um konsistente Ergebnisse bei den Anfragen zu erhalten, da der MetadataProvider dynamisch die ihm bekannten Metadaten ändern kann. Dieser `DynamicMetadataProvider` bietet den idealen Ausgangspunkt der eigenen Implementierung eines `TTPMetadataProviders`, der alle genannten Funktionen der Superklassen benötigt.

Abbildung 6.2.: Vererbungshierarchie des TTPMetadataProviders

Die wichtigste Funktion, die überschrieben werden muss, ist die `resolve` Funktion, die dazu verwendet wird, den EntityDescriptor eines bestimmten Dienstes zu finden. Diese Funktion ist zum Teil in Listing 6.1 abgebildet. Übergeben wird ihr ein Objekt vom Typ `Criteria`, in dem festgelegt ist, welche entityID gesucht wird. Die entityID kann dabei entweder als ASCII- oder Unicode-String angegeben werden. In dieser prototypischen Implementierung wird auf eine Auflösung von Artifacts verzichtet. Die Funktion sucht dann, durch den mit der `TTPUtils::getMetadataFilePath`-Funktion generierten Dateinamen, nach der Datei, die zu der entityID gehören müsste. Diese Funktion erstellt durch eine weitere Hilfsfunktion, die den SHA1-Hash der entityID berechnet, unter Verwendung des im Konstruktor aus der Konfigurationsdatei gelesenen Basispfads, den kompletten Pfad, unter der die Datei zu finden sein müsste. Wenn die Datei existiert, wird diese durch einen Unmarshaller

Listing 6.1: Funktion zur Auflösung von Metadaten

```
saml2md::EntityDescriptor* TTPMetadataProvider::resolve(const
    saml2md::MetadataProvider::Criteria& criteria) const
{
    //[...]
    string filePath = TTPUtils::getMetadataFilePath(m_ttpMetadataStorageDirectory,
    name);
    //[...]
    auto_ptr<XMLObject> xmlObject = getXMLObjectFromFile(filePath);

    EntityDescriptor* entityDescriptor =
    dynamic_cast<EntityDescriptor*>(xmlObject.get());
    if (!entityDescriptor) {
      throw MetadataException("Root␣of␣metadata␣instance␣not␣recognized:␣$1",
    params(1,xmlObject->getElementQName().toString().c_str()));
    }
    xmlObject.release();
    m_log.debug("Parsed␣metadata␣file␣for␣entityID␣%s", name.c_str());
    return entityDescriptor;
    //[...]
}
```

aus dem XML-Dokument in ein EntityDescriptor-Objekt gewandelt. Wenn dieser Vorgang erfolgreich durchgeführt wurde, wird ein Zeiger auf das erstellte EntityDescriptor-Objekt zurückgegeben.

Eine weitere essentielle Funktion für den `TTPMetadataProvider` ist die Implementierung der Funktion `getMetadata()`. Diese Funktion ist in Listing 6.2 dargestellt und gibt die momentan dem MetadataProvider bekannten Entities als `XMLObject` mit `EntityDescriptor`-Elementen innerhalb eines `EntitiesDescriptor`-Elements zurück. Diese Funktion wird von dem Embedded Discovery Service benötigt, um festzustellen, von welchen Providern der SP Metadaten besitzt. Dazu iteriert die Funktion über die von der TTP heruntergeladenen Metadatendateien und baut so das resultierende `XMLDokument` zusammen.

Da der TTPMetadataProvider eine Erweiterung des DynamicMetadataProviders ist, der unter anderem auch von dem ObservableMetadataProvider abgeleitet ist, können bei dem TTPMetadataProvider so genannte Observer registriert werden, die bei Änderungen an den Metadaten informiert werden. Damit der MetadataSyncHandler nach dem Download einer neuen Metadatendatei den Benachrichtigungsprozess anstoßen kann, wird eine neue öffentliche Funktion `void metadataDownloaded()` implementiert.

Die privaten Funktionen `getXMLObjectFromFile()` und `getDOMDocumentFromFile()` werden, wie oben dargestellt, von den anderen Funktionen benötigt, um aus den Metadaten-Dateien der TTP Objekte zu erstellen, die weiterverarbeitet werden können.

Konfiguriert wird der neue MetadataProvider in der Konfiguration des Shibboleth SPs `shibboleth2.xml`, über ein MetadataProvider-Element mit dem Typ TTPMetadataProvider. Listing 6.3 zeigt ein Beispiel für diese Konfiguration, die im Konstruktor des `TTPMetadataProviders` eingelesen werden.

Listing 6.2: Funktion, die die gesammten dem `TTPMetadataProvider` bekannten Entities in einem XMLObjekt zurückgibt

```
const XMLObject* TTPMetadataProvider::getMetadata() const
{
  DOMImplementation* impl =
    DOMImplementationRegistry::getDOMImplementation(XMLString::transcode("XML 1.0"));
  DOMDocument* doc =
    impl->createDocument(XMLString::transcode("urn:oasis:names:tc:SAML:2.0:metadata"),
    XMLString::transcode("EntitiesDescriptor"), 0);
  DOMElement* root = doc->getDocumentElement();

  DIR *dir;
  struct dirent *ent;
  if ((dir = opendir(m_ttpMetadataStorageDirectory.c_str())) != NULL) {
    while ((ent = readdir(dir)) != NULL) {
      m_log.info("Found metadata file '%s'", ent->d_name);
      if(ent && strncmp(ent->d_name,".",1) != 0){
        m_log.info("Processing file '%s'", ent->d_name);
        ostringstream file;
        file << m_ttpMetadataStorageDirectory << ent->d_name;
        DOMDocument* doc2 = getDOMDocumentFromFile(file.str());
        if (doc2) {
          m_log.info("1");
          DOMElement* e = doc2->getDocumentElement();
          DOMNode* import = doc->importNode(e, true);
          m_log.info("2");
          root->appendChild(import);
        }
        doc2->release();
      }
    }
    closedir (dir);
  } else {
    m_log.error("Could not open directory %s",
    m_ttpMetadataStorageDirectory.c_str());
  }

  XMLObject* xmlObject =
    XMLObjectBuilder::buildOneFromElement(doc->getDocumentElement(), true);
  doc->release();
  return xmlObject;
}
```

Listing 6.3: TTPMetadataProvider Konfiguration

```
<SPConfig [...]>
  [...]
  <MetadataProvider type="TTPMetadataProvider"
    metadataStorageDirectory="/usr/local/etc/shibboleth/metadata/ttp"/>
  [...]
</SPConfig>
```

6.1.2. MetadataSyncHandler

Der MetadataSyncHandler ist eine Erweiterung der Klassen `AbstractHandler` und `Remoted Handler`. Der `AbstractHandler` ist eine Basisklasse für die Implementierung eines `Handlers`, der unter einer URL bei dem Webserver registriert wird und beim Aufruf dieser URL ausgeführt wird. Der `RemotedHandler` erweitert diesen `Handler`, so dass Nachrichten vom Webserver-Kontext an den Shibd-Prozess weitergeleitet und dort verarbeitet werden können. Abbildung 6.3 zeigt die Vererbungshierarchie des TTPMetadataSyncHandlers.

Die zentrale Funktion eines Handlers ist die in Listing 6.4 angegebene `run()`-Methode, in der zwischen der Ausführung `InProcess`, also im Prozess des Webservers und `OutOfProcess`, dem Shibd-Prozess unterschieden wird. Im Kontext des Webservers wird dabei die konfigurierte ACL überprüft, um sicher zu stellen, dass nur Metadatensynchronisations-Anfragen von explizit zugelassenen Systemen beantwortet werden. Wenn ein anfragendes System nicht in der Access Control List (ACL) enthalten ist, wird der Request direkt im Webserver abgelehnt.

Wenn der Request zulässig ist, wird (weiterhin im Kontext des Webservers `InProcess`) der Aufruf an den Shibd-Prozess weitergeleitet. Dieser Vorgang wurde zu Beginn des Abschnittes in Abbildung 6.1 schematisch beschrieben. Der Request wird in ein DDF-Objekt verpackt, das per IPC an den Shibd-Prozess geschickt wird. Innerhalb des Shibd-Prozesses wird das DDF-Objekt in der `receive`-Funktion entpackt und an die `processMessage`-Funktion weitergeleitet. Die Process-Funktion führt dann den eigentlichen Download durch und gibt das Ergebnis zurück. Das Ergebnis wird dann wieder per IPC an den im Webserver-Kontext laufenden Prozess zurückgesendet, wo es in der `out`-Variable gespeichert wird. Die Antwort wird dann ausgepackt und an den Webserver zurückgegeben, der dann die entsprechende Statusmeldung ausgibt. Wenn die `run`-Methode direkt durch den Shibd-Prozess ausgeführt wird, kann sofort die `processMessage`-Methode aufgerufen werden.

Die eigentliche Behandlung des Download-Requests wird in der `processMessage`-Funktion durchgeführt. In dem Quellcode sind für die Parameternamen Konstanten definiert worden, um eventuelle Änderungen einfach und konsistent umsetzen zu können. Die Funktion extrahiert nun die Parameter `entityID` (angegeben durch die Konstante `DOWNLOAD_ENTITY_ PARAM_NAME`) und `location` (`DOWNLOAD_URL_PARAM_NAME`) aus dem Request und speichert sie in den Variablen `entityID` und `url`.

Wenn beide Variablen vorhanden sind, wird, wie bei dem TTPMetadataProvider, der Pfad zu der lokalen Metadatendatei berechnet und durch die `downloadFile`-Funktion der Download der Datei gestartet. Diese Funktion verwendet die Curl-Bibliothek für den eigentlichen Download. Nach dem Download wird der TTPMetadataProvider über die neue Metadatendatei informiert.

Dieser Handler wird, wie der MetadataProvider, ebenfalls in der `shibboleth2.xml`-Konfigurationsdatei definiert. Ein Beispiel für die Konfiguration ist in Listing 6.5 angegeben. Die IPs für die ACL werden dabei einzeln durch Leerzeichen getrennt angegeben. Der Parameter `Location` gibt an, unter welcher URL dieser Handler erreicht wird. Der `metadata StorageDirectory`-Parameter gibt an, wo die heruntergeladenen Metadatendateien gespeichert werden. Dieser Pfad muss mit dem bei dem MetadataProvider angegebenen Pfad übereinstimmen, um eine funktionierende Konfiguration von TTPMetdataProvider und TTPMe-

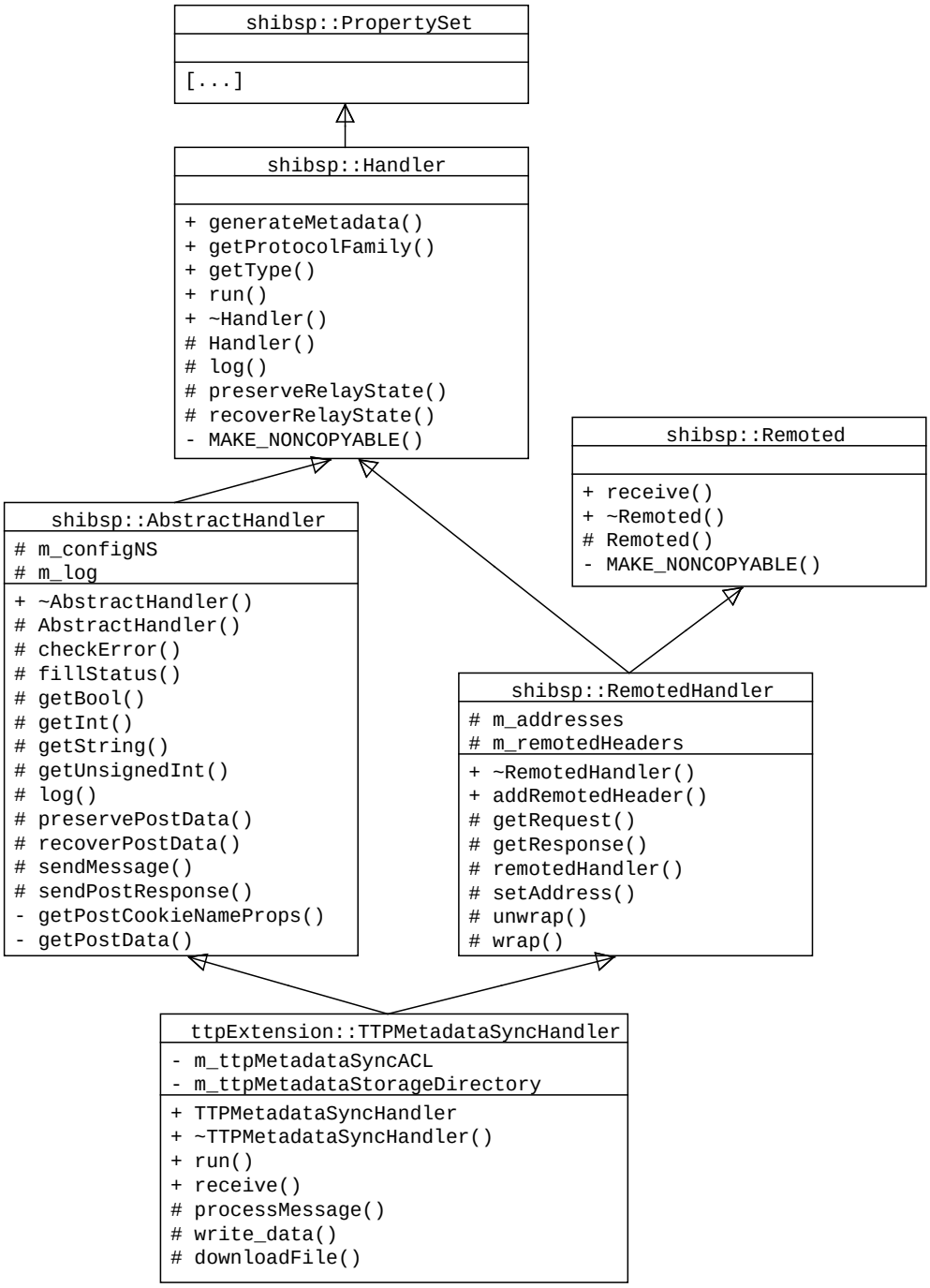

Abbildung 6.3.: Vererbungshierarchie des TTPMetadataSyncHandlers

Listing 6.4: Run-Funktion des TTPMetadataSyncHandlers

```
pair<bool,long> TTPMetadataSyncHandler::run(SPRequest& request, bool isHandler) const
{
  SPConfig& conf = SPConfig::getConfig();

  if (conf.isEnabled(SPConfig::InProcess)) {
    if (m_ttpMetadataSyncACL.empty() ||
    m_ttpMetadataSyncACL.count(request.getRemoteAddr()) == 0) {
      m_log.error("status␣handler␣request␣blocked␣from␣invalid␣address␣(%s)",
    request.getRemoteAddr().c_str());
      istringstream msg("Status␣Handler␣Blocked");
      return make_pair(true,request.sendResponse(msg,
    HTTPResponse::XMLTOOLING_HTTP_STATUS_FORBIDDEN));
    }
  }

  if (conf.isEnabled(SPConfig::OutOfProcess)) {
    return processMessage(request.getApplication(), request, request);
  }
  else {
    DDF out,in = wrap(request);
    DDFJanitor jin(in), jout(out);
    out=request.getServiceProvider().getListenerService()->send(in);
    return unwrap(request, out);
  }
}
```

Listing 6.5: TTPMetadataSyncHandler Konfiguration

```
<SPConfig [...]>
  [...]
  <Sessions>
    [...]
    <Handler type="TTPMetadataSyncHandler" Location="/MetaSync"
      metadataStorageDirectory="/usr/local/etc/shibboleth/metadata/ttp"
      initiatorACL="129.187.163.168␣79.225.220.203" />
  </Sessions>
  [...]
</SPConfig>
```

tadataSyncHandler herzustellen. Die unabhängige Konfiguration der beiden Teile erlaubt es, mehrere TTPMetadataSyncHandler und TTPMetadataProvider für unterschiedliche TTP-Netzwerke zu konfigurieren.

6.1.3. SessionInitiator

Der TTPSessionInitiator ist abgeleitet von den Klassen `SessionInitiator`, `AbstractHandler` und `RemotedHandler`. `AbstractHandler` und `RemotedHandler` wurden bereits im vorhergehenden Abschnitt beschrieben. Der `SessionInitiator` erweitert die `run`-Methode des `AbstractHandlers` um einen weiteren Parameter `entityID`. Diese EntityID gibt an, über welchen Provider versucht werden soll, eine neue Session zu erstellen. Das Vererbungsdiagramm hierzu ist in Abbildung 6.4 abgebildet.

Die Aufgabe des TTPSessionInitators ist es, wenn die TTP dem SP mitteilt, welchen IDP der

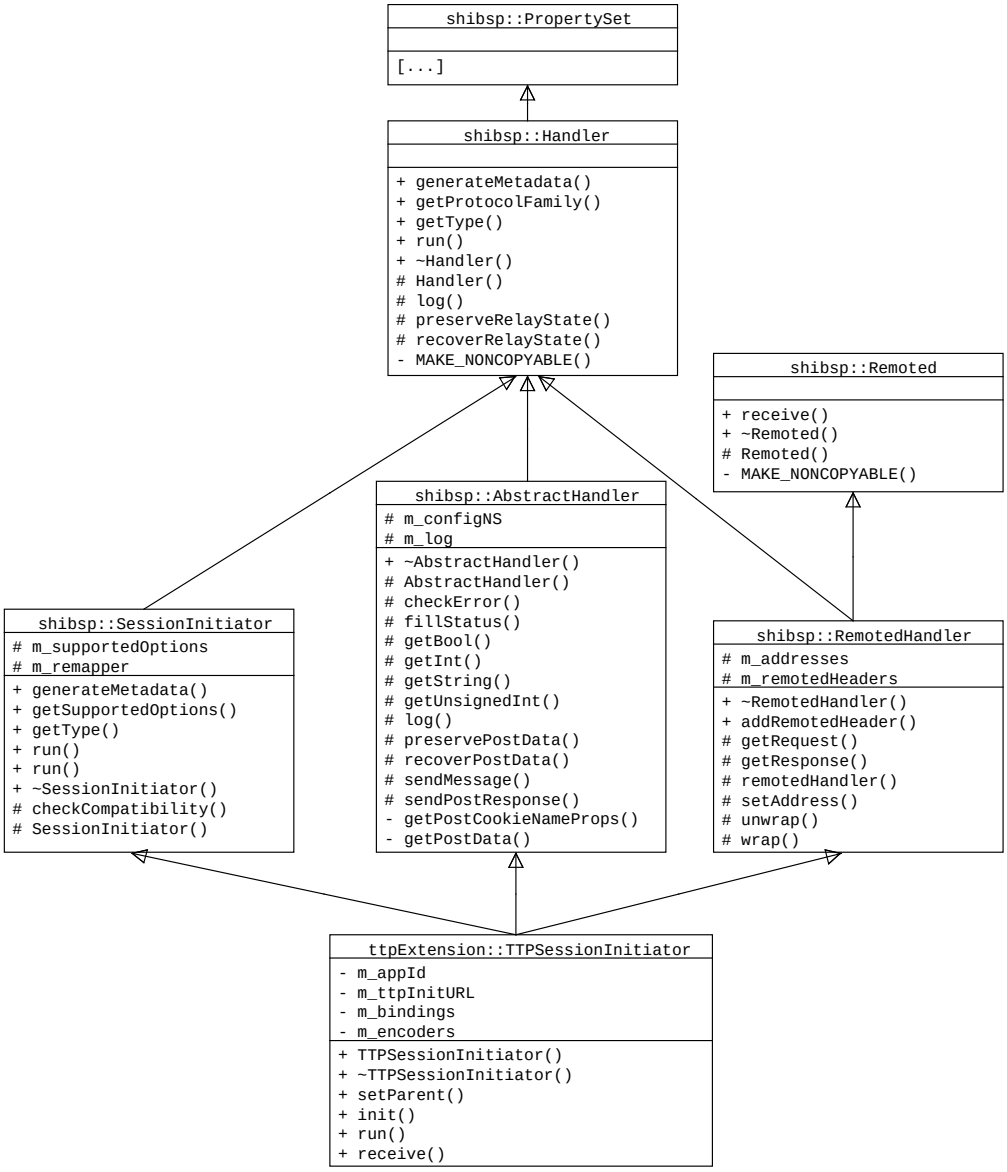

Abbildung 6.4.: Vererbungshierarchie des TTPSessionInitiators

Benutzer gewählt hat, ohne die konkreten Metadaten des IDPs zu kennen, einen Authentifizierungs-Request an diesen zu erstellen und den Request dann an die TTP zu schicken. Dazu wird zunächst, wie bei dem TTPMetadataSyncHandler, ein Request an die `run`-Funktion übergeben. Diese Methode hat allerdings einen zusätzlichen Parameter `entityID`, der angibt, an welchen Dienst ein Request erstellt werden soll. Der Request wird darauf wieder verpackt und an den Shibd-Prozess geschickt, wo er verarbeitet wird und ein Authentication-Request als Antwort erstellt wird. Die Funktion, die den Request erstellt, ist in Listing 6.6 dargestellt.

Zunächst wird dort überprüft, ob die gesuchte EntityID in den Metadaten vorhanden ist. Wenn das der Fall ist, sollte dieser SessionInitiator gar nicht aufgerufen worden sein, da der SP über den normalen SAML2SessionInitiator einen Authentication-Request erstellen könnte. Daher wird dazu eine Warnung ausgegeben, der Vorgang aber trotzdem weitergeführt, da eventuell auf Seite des SPs die Metadaten vorhanden sind, aber nicht auf Seite des IDPs und deshalb das TTP-Protokoll weiter abgearbeitet wird.

Dazu wird ein neuer EntityDescriptor erstellt, dem die übermittelte EntityID zugewiesen wird. Zusätzlich wird ein RoleDescriptor mit einem SingleSignOn-Descriptor benötigt, welcher danach erstellt wird. Als Binding für diesen SSO-Descriptor wird entweder das von der TTP übergebene Binding verwendet oder ein HTTP-Redirect-Binding. Im Normalfall sollte das HTTP-Redirect-Binding ausreichend sein, so dass die TTP kein Binding explizit angeben muss. Ähnlich kann die TTP eine EndpointLocation angeben, die eventuell für andere Bindings benötigt wird, der Basisfall HTTP-Redirect, benötigt diese Angabe allerdings nicht.

Danach wird anhand der verfügbaren Bindings für ausgehende Requests ein Binding und entsprechender Encoder ausgewählt. Beide werden in der `sendMessage`-Methode benötigt, um den Authentication-Request zu verschicken. Der Authentication-Request wird nun aus den vorher bestimmten Einzelteilen zusammengesetzt und dann verschickt. Da die Nachricht an die TTP gesendet werden muss, wird die Zieladdresse vorher noch aus der Konfiguration bestimmt und um den notwendigen `entityID`-Parameter erweitert.

Die Konfiguration des SessionInitiators erfolgt, wie bei den anderen Teilen auch, über die `shibboleth2.xml` Konfigurationsdatei. Listing 6.7 zeigt ein Beispiel dieser Konfiguration. Durch den Parameter `ttpInitURL` wird angegeben, zu welcher URL ein vom SP erstellter Authentication-Request geschickt wird. Der korrekte Wert für diese Konfiguration muss von der TTP vorgegeben werden.

6.2. Identity Provider

Die Erweiterung für den Shibboleth Identity Provider ist ähnlich der für den Service Provider. Prinzipiell wird hier der gleiche MetadataProvider, zum Lesen der Metadaten aus den Dateien und Aufbereiten für den nach einer EntityID suchenden Prozess und Handler, für die Synchronisation von Metadaten, wie beim Service Provider, verwendet. Zusätzlich werden hier aber auch noch Methoden benötigt, die die Synchronisation von Konvertierungsregeln und deren Einsatz ermöglichen. Auf den Teil des SessionInitators kann verzichtet werden, da im TTP-Modell der IDP keinen Verbindungsaufbau initiieren kann. Der Aufbau des grund-

Listing 6.6: Funktion zur Erstellung eines Authentication-Requests

```
pair<bool,long> TTPSessionInitiator::doRequest(const Application& app, const
    HTTPRequest* httpRequest, HTTPResponse& httpResponse, const char* entityID,
    const char* binding, const char* epLocation, const char* acsLocation, const
    XMLCh* acsBinding, bool isPassive, bool forceAuthn, string& relayState) const
{
#ifndef SHIBSP_LITE
  pair<const EntityDescriptor*,const RoleDescriptor*> entity = pair<const
    EntityDescriptor*,const RoleDescriptor*>(nullptr,nullptr);
  const IDPSSODescriptor* role = nullptr;
  const EndpointType* ep = nullptr;
  const MessageEncoder* encoder = nullptr;

  MetadataProvider* m = app.getMetadataProvider();
  Locker locker(m);
  MetadataProvider::Criteria mc(entityID);
  entity = m->getEntityDescriptor(mc);
  //[...]
  EntityDescriptor* dummyIdPEntity =
    EntityDescriptorBuilder::buildEntityDescriptor();
  dummyIdPEntity->setEntityID(XMLString::transcode(entityID));
  entity.first = dummyIdPEntity;

  xmltooling::QName qName = xmltooling::QName(samlconstants::SAML20MD_NS,
    XMLString::transcode("IDPSSODescriptor"));
  IDPSSODescriptor* dummyIdPRole = IDPSSODescriptorBuilder::buildIDPSSODescriptor();
  xmltooling::XMLObjectChildrenList<std::vector<SingleSignOnService*> >
    ssoDescriptions =
    dummyIdPRole->getSingleSignOnServices();

  SingleSignOnService* sso = SingleSignOnServiceBuilder::buildSingleSignOnService();
  //[...]
  ssoDescriptions.push_back(sso);
  entity.second = dummyIdPRole;
  //[...]
  role = dynamic_cast<const IDPSSODescriptor*>(entity.second);
  //[...]
  auto_ptr<AuthnRequest> req(AuthnRequestBuilder::buildAuthnRequest());
  if (ep)
    req->setDestination(ep->getLocation());
  if (acsLocation) {
    auto_ptr_XMLCh wideloc(acsLocation);
    req->setAssertionConsumerServiceURL(wideloc.get());
  }
  //[...]
  if (!req->getIssuer()) {
    Issuer* issuer = IssuerBuilder::buildIssuer();
    req->setIssuer(issuer);
    issuer->setName(
      app.getRelyingParty(entity.first)->getXMLString("entityID").second);
  }
  //[...]
  XMLCh* genid = SAMLConfig::getConfig().generateIdentifier();
  req->setID(genid);
  XMLString::release(&genid);
  req->setIssueInstant(time(nullptr));

  ostringstream dest;
  dest << m_ttpInitURL << "&idpEntityID=" << entityID;
  long ret = sendMessage(*encoder, req.get(), relayState.c_str(),
    dest.str().c_str(), role, app, httpResponse, false);
  req.release(); // freed by encoder
  return make_pair(true, ret);
#else
  return make_pair(false, 0L);
#endif
}
```

Listing 6.7: TTPSessionInitiator Konfiguration

```
<SPConfig [...]>
  [...]
  <Sessions>
    [...]
      <SessionInitiator type="Chaining" Location="/Login" isDefault="true"
    id="Login">
        <SessionInitiator type="SAML2" template="bindingTemplate.html" />
        <SessionInitiator type="SAMLDS"
    URL="http://gntb03.srv.lrz.de/shibboleth-ds/index.html"/>
      </SessionInitiator>

      <SessionInitiator type="TTPSessionInitiator" Location="/TTP" id="TTP"

    ttpInitURL="http://gntb08.srv.lrz.de:8080/discovery/TTP?action=authenticate" />
    [...]
  </Sessions>
  [...]
</SPConfig>
```

legenden Konzeptes der IDP-Implementation ist ebenfalls ähnlich zu dem des SPs, d.h. die Klassen, die abgeleitet werden, haben die selben Namen und Funktionen wie bei dem SP. Allerdings ist der IDP als Tomcat-Web-Anwendung implementiert und wird durch einen Tomcat-Server ausgeführt. Daher müssen die relevanten Teile des SPs nochmals für diese Architektur implementiert werden.

Die Implementierung des IDPs ist dabei übersichtlicher als die des SPs, da alle Funktionen innerhalb eines Tomcat-Servlets implementiert und nicht auf mehrere Prozesse verteilt sind. Eine Übersicht über die Implementierung des IDPs gibt Abbildung 6.5. Die Abbildung zeigt, die standardmäßig vorhandenen Servlets sowie das neu hinzugefügte `TTPMetadataSyncServlet`.

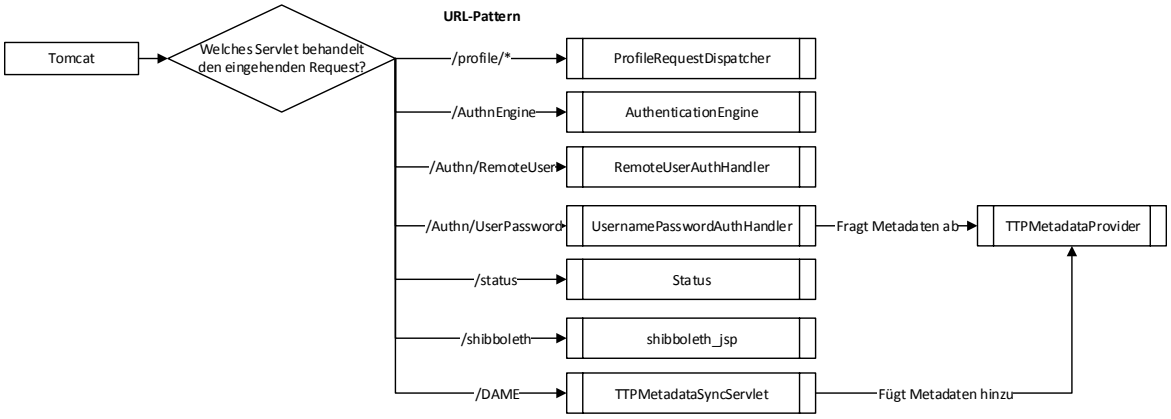

Abbildung 6.5.: Übersicht über die Architektur der IDP-Implementierung

6.2.1. MetadataProvider

Für die Erstellung eines eigenen MetadataProvider-Elements für die IDP-Konfigurationsdatei muss in der Java-Implementierung nicht nur eine entsprechende Klasse abgeleitet und imple-

Listing 6.8: Schema für das TTPMetadataProvider-Element

```
<?xml version="1.0" encoding="UTF-8"?>

<schema targetNamespace="urn:mace:gntb.lrz.de:ttpextension:ttpmetadataprovider"
    xmlns="http://www.w3.org/2001/XMLSchema"
    xmlns:md="urn:mace:shibboleth:2.0:metadata"
    elementFormDefault="qualified">

  <import namespace="urn:mace:shibboleth:2.0:metadata"
    schemaLocation="classpath:/schema/shibboleth-2.0-metadata.xsd" />

  <complexType name="TTPMetadataProvider">
    <complexContent>
      <extension base="md:MetadataProviderType">
        <attribute name="metadataStorageDirectory" type="string" />
      </extension>
    </complexContent>
  </complexType>

</schema>
```

mentiert werden, es muss auch eine Schemadefiniton erstellt werden. Das von dem `MetadataProviderType` abgeleitete Element ist in Listing 6.8 dargestellt. Dort wird durch den `targetNamespace` ein neuer Namespace für das erweiterte Element angegeben sowie ein neuer komplexer Typ definiert. In diesem Fall wird für das Element nur ein Attribut `metadataStorageDirectory`, zur Angabe des Speicherortes, erstellt. Um diese Schema-Datei zu verwenden, müssen zusätzlich Klassen für einen NamespaceHandler und BeanDefinitionParser erstellt werden, die festlegen mit welchen Parametern der Konstruktor der TTPMetadataProvider-Klasse aufgerufen wird. Da es hier nur einen Konfigurationsparameter gibt, wird nur dieser eine Parameter übergeben.

Implementiert wird der TTPMetadataProivder durch das Ableiten des BaseMetadataProviders, der die grundlegende Struktur für einen MetadataProvider liefert und dem Interface ObservableMetadataProvider, zur Benachrichtigung von Observern bei Änderungen an den Metadaten.

Die Funktion `getEntityDescriptor` wird bei der Suche nach den Metadaten zu einer EntityID von anderen Programmteilen aufgerufen. Dies ist die wichtigste Funktion, die neu implementiert werden muss, da sie bestimmt, wie die Suche nach den Metadaten zu der gegeben EntityID abläuft. Listing 6.9 zeigt diese Funktion. Die Implementierung auf Seiten des IDPs verwendet einen Cache bei der Suche nach den Metadaten, der direkt zu Anfang durchsucht wird. Wenn der gesuchte EntityDescriptor in diesem Cache enthalten ist, kann er sofort zurückgegeben werden. Der EntityDescriptor wird dabei in einem Objekt der Klasse `ttpEntityDescriptor` gespeichert, das den gecacheten Eintrag mit einigen Metainformationen, wie dem Zeitpunkt der letzten Verwendung, verbindet. Anhand dieser Informationen kann entschieden werden, welche Einträge eventuell aus dem Cache entfernt werden sollten.

Kann die EntityID nicht im Cache gefunden werden, wird zunächst der SHA1-Hash des Dateinamen berechnet und nach einer Datei mit dem entsprechenden Namen in dem `metadataStorageDirectory` gesucht. Wenn diese Datei existiert und lesbar ist, wird ein Unmarshaller, wie bei der SP-Implementierung, verwendet, um die XML-Datei in ein Objekt vom Typ Enti-

Listing 6.9: Ausschnitte aus der Funktion, die zu einer gegebenen EntityID den passenden EntityDescriptor sucht

```
1  public EntityDescriptor getEntityDescriptor(String entityID) throws
       MetadataProviderException {
2    TTPEntityDescriptor ttpEntityDescriptor = indexedDescriptors.get(entityID);
3    if(ttpEntityDescriptor != null){
4      log.debug("Found␣EntityDescriptor␣for␣{}␣in␣cache", entityID);
5      ttpEntityDescriptor.setLastUsed(System.currentTimeMillis());
6      return ttpEntityDescriptor.getEntityDescriptor();
7    } else {
8      String entityHash = generateHash(entityID);
9      File metadataFile = new File(this.metadataStorageDirectory+entityHash);
10     FileInputStream in = new FileInputStream(metadataFile);
11
12     //[...]
13
14     Document mdDoc = ppMgr.parse(in);
15     Element metadataRoot = mdDoc.getDocumentElement();
16
17     UnmarshallerFactory unmarshallerFactory = Configuration.getUnmarshallerFactory();
18     Unmarshaller unmarshaller = unmarshallerFactory.getUnmarshaller(metadataRoot);
19
20     EntityDescriptor entityDescriptor = (EntityDescriptor)
       unmarshaller.unmarshall(metadataRoot);;
21
22     if(entityDescriptor != null){
23       this.indexedDescriptors.put(entityID, new
       TTPEntityDescriptor(entityDescriptor));
24       return entityDescriptor;
25     }
26     //[...]
27   }
28 }
```

tyDescriptor zu parsen. Neben dieser Funktion, müssen auch noch Funktionen implementiert werden, die den RoleDescriptor, der zu einem EntityDescriptor gehört, zurückgeben.

Die Konfiguration des MetadataProvider-Elements ist beinahe identisch zu der auf Seite des SPs, benötigt aber die Angabe des vorher definierten Namespaces. Zudem können bei dem IDP mehrere MetadataProvider mit Hilfe des `ChainingMetadataProvider`-Elements kombiniert werden. Dies ermöglicht es sowohl dateibasierte MetadataProvider als auch den TTPMetadataProvider zu verwenden.

6.2.2. MetadataSyncHandler

Die Implementierung des Handlers, der für die Synchronisation der Metadaten verantwortlich ist, ist ebenfalls sehr ähnlich zu der des SPs. Für einen eigenen Handler müssen auch keine Schemata angelegt werden. Die Implementierung beschränkt sich daher auf eine Klasse, die über die IDP-Konfiguration als Handler aktiviert wird. Diese Klasse wird von der HttpServlet-Klasse abgeleitet, die für Tomcat-Webschnittstellen verwendet wird.

Aus der HttpServlet-Klasse wird die Methode `doGet` neu implementiert, die aufgerufen wird, wenn der Handler einen HTTP-GET-Request empfängt. Diese Methode ruft dann die Funktion `handleDownload` auf, die in Listing 6.11 dargestellt ist. Dort wird, wie bei der SP-

Listing 6.10: TTPMetadataProvider Konfiguration für den IDP

```
<rp:RelyingPartyGroup ...
           xmlns:ttp="urn:mace:gntb.lrz.de:ttpextension:ttpmetadataprovider"
           xsi:schemaLocation="...
                    urn:mace:gntb.lrz.de:ttpextension:ttpmetadataprovider␣
    classpath:/schema/ttp-metadata-provider.xsd">
  ...
  <metadata:MetadataProvider id="ShibbolethMetadata"
    xsi:type="metadata:ChainingMetadataProvider">

    ...

    <metadata:MetadataProvider id="TTP" xsi:type="ttp:TTPMetadataProvider"
      location="/opt/shibboleth-idp/metadata/ttp">
    </metadata:MetadataProvider>
  </metadata:MetadataProvider>
  ...
</rp:RelyingPartyGroup>
```

Implementierung, zunächst überprüft, ob die Anfrage von einem System ist, das auf der ACL steht. Danach werden die Parameter für die `entityID` und die URL, unter der die Metadaten gehostet werden, ausgelesen und die Datei an den aus der Konfiguration und dem SHA1-Hash der EntityID berechneten, Ort gespeichert.

6.2.2.1. Konvertierungsregel Synchronisation

Der Installationsvorgang besteht dabei aus zwei Schritten. Zum einen muss das neue Attribut in die `attribute-resolver.xml` übernommen und die Datei neu eingelesen werden. Zum anderen muss das neue Attribut freigegeben werden, in dem es in die `attribute-filter.xml` Datei eingefügt und auch diese Datei neu geladen wird. Die beiden Dateien werden vom IDP, wie in Abschnitt 4.1.2 beschrieben, auch für die Definition und Freigabe aller anderen Attribute verwendet.

Nach einer erfolgreichen Übertragung, kann die Regel angewendet werden. Dazu muss zunächst festgelegt werden, welche Form und Struktur eine Konvertierungsregel überhaupt hat. Da zuvor in Abschnitt 5.3 festgelegt wurde, dass die Konvertierungsregeln auf Seite des IDP integriert werden sollen und dieser durch die in Abschnitt 4.1.2 beschriebene und in Listing 4.2 beispielhaft dargestellte Datei `attribute-resolver.xml` schon über die Konvertierungsfähigkeit lokaler Attribute verfügt, müssen neue Regeln ebenfalls in diese Datei integriert werden. Außerdem müssen die Regeln so allgemein gehalten sein, dass sie unabhängig von den lokal vergebenen IDs für `AttributeDefinition`- und `DataConnector`-Elemente funktionieren.

Bevor die neue Regel eingefügt werden kann, müsste daher zunächst die aktuelle Konfiguration eingelesen, die relevanten Abhängigkeiten und deren IDs ausgelesen und in der Regel ersetzt werden.

Da es sich um eine XML-Datei handelt, die modifiziert werden muss, braucht kein Parser für diese Änderungen geschrieben zu werden. Stattdessen können die Regeln in der Form von Extensible Stylesheet Language Transformations (XSLT) definiert werden. XSLT erlaubt es

Listing 6.11: Funktion zum Download von Metadaten

```
private void handleDownload(HttpServletRequest req, HttpServletResponse res) {

  if(!ttpMetadataSyncACL.contains(req.getRemoteAddr())
      || !ttpMetadataSyncACL.contains(req.getRemoteHost())){
    res.setStatus(HttpServletResponse.SC_FORBIDDEN);
    return;
  }

  String entityID = req.getParameter(DOWNLOAD_ENTITY_PARAM_NAME);
  String location = req.getParameter(DOWNLOAD_URL_PARAM_NAME);
  //[...]
  String entityHash = generateHash(entityID);

  URL url = new URL(location);
  log.info("Downloading metadata from {}", url.getQuery());

  HttpURLConnection conn = (HttpURLConnection) url.openConnection();
  int responseCode = conn.getResponseCode();
  if (responseCode == HttpURLConnection.HTTP_OK) {
    InputStream inputStream = conn.getInputStream();
    String metadataFilePath = ttpMetadataStorageDirectory+entityHash;
    FileOutputStream outputStream = new FileOutputStream(metadataFilePath);

    int bytesRead = -1;
    byte[] buffer = new byte[BUFFER_SIZE];
    while ((bytesRead = inputStream.read(buffer)) != -1) {
      outputStream.write(buffer, 0, bytesRead);
    }

    outputStream.close();
    inputStream.close();

    res.setStatus(HttpServletResponse.SC_OK);
    return;

  } else {
    log.warn("Could not download metadata file. Server replied with HTTP code: {}",
    responseCode);
  }
  //[...]
  res.setStatus(HttpServletResponse.SC_INTERNAL_SERVER_ERROR);
}
```

XML-Dateien zu transformieren. Durch die Transformation in eine neue XML-Datei können auch neue Elemente, unter Berücksichtigung schon vorhandener Elemente, eingefügt werden.

Listing 6.12 zeigt eine XSLT-Datei die, wenn sie auf die `attribute-resolver.xml` angewendet wird, eine Konvertierungsregel, die das Attribut `norEduPersonBirthDate` in das Attribut `schacDateOfBirth` umbenennt, einfügt.

In dem `stylesheet`-Element werden zunächst die benötigten XML-Namespaces (XMLNS) spezifiziert. Das erste `template`-Element in Zeile 12 sorgt dafür, dass das Root-Element sowie Kommentare außerhalb der Wurzel kopiert werden. Das zweite `template`-Element in Zeile 15 sucht nach dem `AttributeResolver`-Element, welches das Wurzel-Element der `attribute-resolver.xml` ist. Dessen Inhalt wird dann durch die folgende Copy und Template Anweisung ebenfalls kopiert.

In Zeile 22 wird nun eine Variable über einen XPath-Ausdruck definiert. Dieser Ausdruck sucht anhand des `friendlyNames` das Element der Attributsdefinition für das Attribut `norEduPersonBirthDate`. In dem `if`-Element ab Zeile 26 wird darauf überprüft, ob die Variable belegt werden konnte, d.h. dass das benötigte Element gefunden wurde. Falls das Element nicht gefunden werden konnte, wird der Transformationsvorgang mit einer Fehlermeldung abgebrochen.

In Zeile 32 wird ein neues `AttributeDefinition`-Element, für das generierte `schacDateOfBirth`-Attribut, erzeugt. Dem Element werden in den darauffolgenden Zeilen die Attribute `id` und `xsi:type` sowie `sourceAttributeID` zugewiesen. Letzteres ist abhängig von der ID des Quellattributes `norEduPersonBirthDate` und wird daher über die zuvor definierte Variable bestimmt.

Zum Schluss werden noch zwei weitere Kind-Elemente `AttributeEncoder` erzeugt, die jeweils einmal für SAML 1 und SAML 2 den Namen des neuen Attributs und dessen Kodierung bestimmen. In diesem Beispiel hat der Attribut-Encoder den Typen `SAML1String` bzw. `SAML2String` und erzeugt die Attribute mit den Namen `urn:mace:dir:attribute-def:schacDateOfBirth` bzw. `urn:oid:1.3.6.1.4.1.1466.115.121.1.36`. Für die SAML 2 Version wird zusätzlich ein Attribut für den menschenlesbaren `friendlyName` angegeben.

Neben der `attribute-resolver.xml` muss auch die Datei `attribute-filter.xml` modifiziert werden. Diese Datei regelt, wie in Abschnitt 4.1.2 beschrieben, welche Attribute an einen SP herausgegeben werden dürfen. Solange das Attribut nur definiert nicht aber auch freigegeben ist, kann es nicht verwendet werden. Die Freigaben sind dabei über `AttributeFilterPolicy`-Elemente gruppiert. Diese Elemente werden über eine `id` identifiziert. Um alle durch die TTP generierten Freigaben zusammenzufassen, wird hier ein neues `AttributeFilterPolicy` Element erstellt, das die ID `ttp-attribute-releases` hat.

Für das Einfügen einer neuen Freigabe wird auch hier eine XSLT verwendet, die in Listing 6.13 dargestellt ist. Diese XSLT muss nicht für jede Regel neu erstellt werden, sondern bekommt bei der Ausführung über einen Parameter mitgeteilt, welche ID das neue freizugebende Attribut hat.

Der Aufbau der XSLT-Datei ist sehr ähnlich zu der, die eine neue Attributs-Definition erstellt. Zunächst werden alle benötigten XML-Namespaces definiert und die Ausgabemethode

Listing 6.12: Beispiel für eine Konvertierungsregel

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:resolver="urn:mace:shibboleth:2.0:resolver"
  xmlns:ad="urn:mace:shibboleth:2.0:resolver:ad"
  xmlns:dc="urn:mace:shibboleth:2.0:resolver:dc">

  <xsl:output method="xml" indent="yes"
    cdata-section-elements="ad:Script␣dc:FilterTemplate"/>

  <xsl:template match="@*|node()">
    <xsl:copy>
      <xsl:apply-templates/>
    </xsl:copy>
  </xsl:template>

  <xsl:template match="resolver:AttributeResolver">
    <xsl:copy>
      <xsl:apply-templates/>

      <xsl:variable name="norEduPersonBirthDate"
        select="//resolver:AttributeDefinition
          [resolver:AttributeEncoder/@friendlyName='norEduPersonBirthDate']" />

      <xsl:if test="not(string($norEduPersonBirthDate))">
        <xsl:message terminate="yes">
          Error: Could not find all dependencies!
        </xsl:message>
      </xsl:if>

      <xsl:element name="resolver:AttributeDefinition">
        <xsl:attribute name="id">schacDateOfBirth</xsl:attribute>
        <xsl:attribute name="xsi:type">ad:Simple</xsl:attribute>
        <xsl:attribute name="sourceAttributeID">
          <xsl:value-of select="$norEduPersonBirthDate/@id"/>
        </xsl:attribute>

        <xsl:element name="resolver:Dependency">
          <xsl:attribute name="ref">
            <xsl:value-of select="$norEduPersonBirthDate/resolver:Dependency/@ref"/>
          </xsl:attribute>
        </xsl:element>
        <xsl:element name="resolver:AttributeEncoder">
          <xsl:attribute name="xsi:type">enc:SAML1String</xsl:attribute>
          <xsl:attribute
    name="name">urn:mace:dir:attribute-def:schacDateOfBirth</xsl:attribute>
        </xsl:element>
        <xsl:element name="resolver:AttributeEncoder">
          <xsl:attribute name="xsi:type">enc:SAML2String</xsl:attribute>
          <xsl:attribute
    name="name">urn:oid:1.3.6.1.4.1.1466.115.121.1.36</xsl:attribute>
          <xsl:attribute name="friendlyName">schacDateOfBirth</xsl:attribute>
        </xsl:element>
      </xsl:element>
    </xsl:copy>
  </xsl:template>
</xsl:stylesheet>
```

`XML` ausgewählt. In Zeilen 8 und 9 werden die Parameter `attributeID` und `spEntityID` definiert, welche bei der Ausführung der Transformation übergeben werden müssen und angeben, welches Attribut an welchen SP freigegeben werden soll.

Der Hauptteil der XSLT besteht aus zwei Template-Definitionen. Das erste Template in Zeile 17 kopiert das Root-Element `AttributeFilterPolicyGroup` und dessen kompletten Inhalt. In der `If`-Bedingung in Zeile 21 wird überprüft, ob es ein `AttributeFilterPolicy`-Element mit der ID des SPs gibt, für den das Attribut freigegeben werden soll. Wenn es ein solches Element nicht gibt, wird es angelegt. Um die Freigabe der Attribute pro SP steuern zu können, muss für jeden SP ein solches Element angelegt werden. Über dessen Kind-Element `PolicyRequirementRule` kann dann mit dem Attribut `typ`, welcher als `basic:Attribute RequesterString` angegeben wird, sowie dem Attribut `value` mit der EntityID des SPs als Wert, definiert werden, das alle Attribute, die in dieser Policy freigegeben werden nur für den SP gelten.

In Zeile 32 wird dann durch das `AttributeRule`-Element das eigentliche Attribut freigegeben. Das freizugebende Attribut wird dabei über den Wert des Attributs `attributeID` identifiziert. Das Kind-Element der `AttributeRule PermitValueRule` erlaubt für das Attribut alle Werte.

Da die XSLT nur auf dem Orginaldokument arbeitet, wird durch das zweite Template nicht eine zweite, identische `AttributeRule` hinzugefügt, nachdem das erste Template bereits ein passendes `AttributeFilterPolicy`-Element mit der `AttributeRule` erstellt hat.

Das zweite Template, in Zeile 45, durchsucht alle `AttributeFilterPolicy`-Elemente und kopiert deren Inhalt. Wenn das Template dabei allerdings durch die `If`-Bedingung in Zeile 48 ein `AttributeFilterPolicy`-Element findet, das eine ID gleich der EntityID des SPs hat, fügt es dort die neue `AttributeRule` ein.

Vor der Modifizierung der aktuellen `attribute-resolver.xml` muss allerdings von dieser eine Kopie erstellt werden, um im Fehlerfall die vorherige Konfiguration wiederherstellen zu können und generell Änderungen an der Datei für Administratoren nachvollziehbar zu machen. Hierbei gibt es verschiedene Möglichkeiten. Die einfachste ist es von der zu modifizierenden Datei eine Kopie zu erstellen und deren Dateinamen mit einer Versionsnummer, zum Beispiel dem aktuellen Datum nach dem Unix-Timestamp, zu erweitern. Dazu kann in dem `conf`-Verzeichnis, das bei einer Standard Shibboleth IDP Installation alle Konfigurationsdateien enthält, ein neues Verzeichnis `conf-versions` angelegt werden, in dem die Kopien gespeichert werden.

Eine andere Methode wäre es ein Versionsverwaltungssystem wie CVS, SVN oder GIT zu verwenden. Der größte Vorteil gegenüber dem zuvor vorgeschlagenen Kopieren von Dateien ist es, dass die alten Versionen der Dateien übersichtlicher organisiert sind. Dieser Vorteil spielt allerdings nur dann wirklich eine Rolle, wenn viele Versionen der Dateien angelegt werden, d.h. wenn häufig neue Attribute durch Konvertierungsregeln erzeugt werden sollen. Außerdem erhöht die Verwendung einer Versionsverwaltung die Anzahl der Abhängigkeiten der Implementierung, was eventuell Administratoren von der Installation der Erweiterung abschrecken könnte.

Die Entscheidung, welches System für die Versionierung verwendet werden soll, kann abhängig von der jeweiligen Implementierung getroffen werden. Für den Einsatz zum Testen des

Listing 6.13: XSLT zur Erweiterung der Datei `attribute-filter.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:afp="urn:mace:shibboleth:2.0:afp"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

  <xsl:output method="xml" indent="yes"/>

  <xsl:param name="attributeID" />
  <xsl:param name="spEntityID" />

  <xsl:template match="@*|node()">
    <xsl:copy>
      <xsl:apply-templates/>
    </xsl:copy>
  </xsl:template>

  <xsl:template match="afp:AttributeFilterPolicyGroup">
    <xsl:copy>
      <xsl:apply-templates/>

      <xsl:if test="not(//afp:AttributeFilterPolicy[@id=$spEntityID])">
        <xsl:element name="afp:AttributeFilterPolicy">
          <xsl:attribute name="id">
            <xsl:value-of select="$spEntityID"/>
          </xsl:attribute>
          <xsl:element name="afp:PolicyRequirementRule">
            <xsl:attribute
    name="xsi:type">basic:AttributeRequesterString</xsl:attribute>
            <xsl:attribute name="value">
              <xsl:value-of select="$spEntityID"/>
            </xsl:attribute>
          </xsl:element>
          <xsl:element name="afp:AttributeRule">
            <xsl:attribute name="attributeID">
              <xsl:value-of select="$attributeID"/>
            </xsl:attribute>
            <xsl:element name="afp:PermitValueRule">
              <xsl:attribute name="xsi:type">basic:ANY</xsl:attribute>
            </xsl:element>
          </xsl:element>
        </xsl:element>
      </xsl:if>
    </xsl:copy>
  </xsl:template>

  <xsl:template match="afp:AttributeFilterPolicy">
    <xsl:copy>
      <xsl:apply-templates/>
      <xsl:if test="@id=$spEntityID␣and␣
    not(afp:AttributeRule[@attributeID=$attributeID])">
        <xsl:element name="afp:AttributeRule">
          <xsl:attribute name="attributeID">
            <xsl:value-of select="$attributeID"/>
          </xsl:attribute>
          <xsl:element name="afp:PermitValueRule">
            <xsl:attribute name="xsi:type">basic:ANY</xsl:attribute>
          </xsl:element>
        </xsl:element>
      </xsl:if>
    </xsl:copy>
  </xsl:template>
</xsl:stylesheet>
```

Systems ist eine dateibasierte Lösung vermutlich am besten geeignet, da sie die Anzahl der Abhängigkeiten minimiert und einen insgesamt geringeren Konfigurationsaufwand hat. Für eine Produktivumgebung, die mehrere Jahre in Betrieb sein soll, ist es auf Dauer vermutlich übersichtlicher, wenn eine Versionsverwaltung verwendet werden kann.

6.3. Trusted Third Party als CDS-Erweiterung

Der Centralized Discovery Service, der von der Architektur sehr ähnlich zu dem IDP ist, kann ebenfalls um einen eigenen weiteren Handler erweitert werden, der in Verbindung mit dem schon vorhandenen Handler für DS-Requests verwendet werden kann. Dafür werden die neuen Klassen `TTPServerServlet`, zum Entgegennehmen der Anfragen und `TTPServer MetadataSyncHandler` sowie `TTPServerConversionRuleSyncHandler` zur Synchronisation von Metadaten bzw. Konvertierungsregeln erstellt. Das Servlet dient dabei als Einstiegspunkt, der die Konfiguration lädt und eingehende Anfragen an die Handler, von denen es mehrere geben kann, weiterleitet. Mehrere Handler des gleichen Typs sind dann von Interesse, wenn verschiedene TTP-Netzwerke von einem CDS betrieben werden sollen. Die Integration von DS- und TTP-Funktionen ist in Abbildung 6.6 dargestellt. Es ist zu sehen, dass der DS-Teil nicht modifiziert wird, wodurch es möglich ist, diesen getrennt von der TTP-Entwicklung zu verwenden und zu aktualisieren.

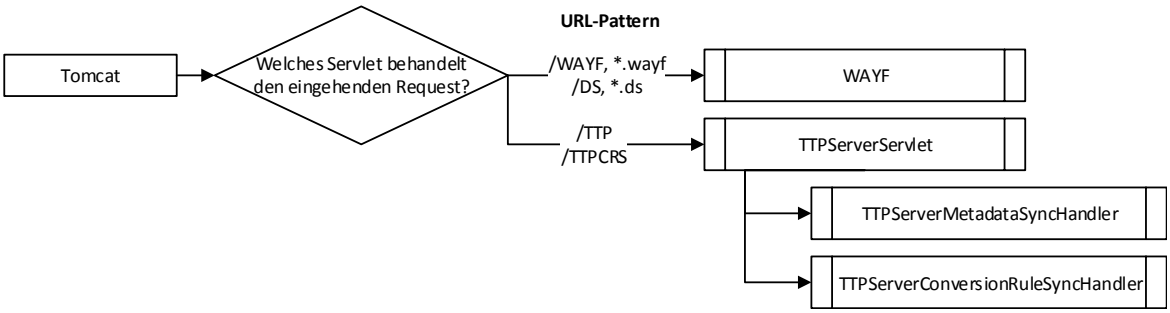

Abbildung 6.6.: Übersicht über die Architektur der CDS-TTP-Implementierung

Um alle Funktionen, die für die TTP benötigt werden, zu verwalten, wird eine Datenbank benötigt. In Abbildung 6.7 ist das konkret implementierte Schema zu sehen. Grau hinterlegte Tabellen wurden, für eine generelle Demonstration des Konzeptes als am wenigsten wichtig betrachtet und daher auf eine nachfolgende Version der Implementierung verschoben.

Die folgenden beiden Listings 6.14 und 6.15 zeigen den wichtigsten Teilprozess der TTP bei der Synchronisation von Metadaten. Der von dem SP kommende Request enthält einen Authentication Request, der zunächst überprüft werden muss. Dazu müssen die entsprechenden Parameter ausgelesen und der Request dekodiert werden. Ein per HTTP-Redirect-Binding mit einem GET-Request verschickter Authentication Request ist per gzip komprimiert und dann base64 kodiert. Um an den eigentlichen Request zu kommen, muss daher dieser Prozess umgekehrt werden. Nach der Spezifikation von SAML muss der Request nicht unbedingt komprimiert werden, bei Anfragen von Shibboleth-Systemen, kann allerdings davon ausgegangen werden, dass er es ist. Bei der Entwicklung ist aufgefallen, dass der Shibboleth IDP, unkomprimierte Authentication Request überhaupt nicht verarbeiten kann.

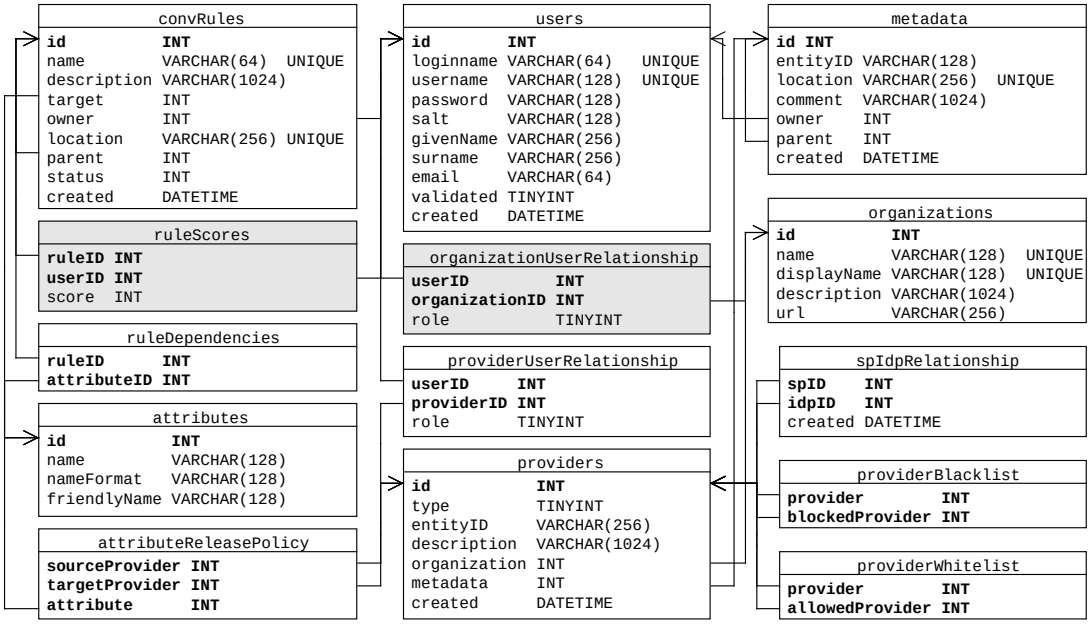

Abbildung 6.7.: Datenbankschema der Implementierung

Nachdem der Authentication Request ausgelesen wurde, wird zunächst mit Hilfe der Datenbank überprüft, ob die Verbindung zwischen dem SP und dem ausgewählten IDP erlaubt ist. Wie in Abbildung 6.7 dargestellt, enthält die Datenbank dafür zwei Tabellen `providerBlacklist` und `providerWhitelist`, die jeweils ein Paar aus verbotenen bzw. erlaubten Quell-Provider und Ziel-Provider definieren. Da diese Tabellen jeweils nur eine Richtung der Verbindung angeben, müssen beide Richtungen überprüft werden. Die Funktion `isProviderConnectionAllowed(sourceProvider, destinationProvider)` überpüft zunächst, ob die Kombination der Provider in der Blacklist enthalten ist. Wenn dies der Fall ist, gibt die Funktion sofort `false` zurück. Wenn nicht, wird zunächst geprüft, ob es für den Quell-Provider einen Eintrag als Quelle in der Whitelist gibt. Falls es keinen gibt, hat der Provider keine Whitelist und erlaubt daher die Verbindung. Gibt es mindestens einen Whitelist-Eintrag wird überprüft, ob der Ziel-Provider durch einen dieser Einträge freigegeben ist.

Der Request des SP wird dann, wie in Listing 6.15 dargestellt, in der Tomcat-Session gespeichert. Zusätzlich werden dort auch der Relaystate, sowie die EntityID des SP und IDP hinterlegt. Diese Informationen werden benötigt, wenn die TTP den ursprünglichen Request des SP an den IDP weiterleiten soll. Danach wird ein neuer Authentication Request an den IDP erstellt und abgeschickt. Hierbei wird immer das HTTP-Redirect-Binding verwendet.

Listing 6.14: Teil 1 der `handleAuthentication` Funktion: Verifizierung des Authentication Requests

```
1  private void handleAuthentication(HttpServletRequest req, HttpServletResponse res)
       throws WayfRequestHandled,
2    WayfException {
3
4    String idpEntityID, spSamlRequestBase64, spSamlRequestRelayStateStr,
       spSamlRequestSigAlgStr, spSamlRequestSigBase64, spSamlRequestStr;
5    byte[] spSamlRequestDecoded;
6
7    idpEntityID = req.getParameter(SP_AUTH_REQ_IDP_ENTITYID_ATTRIBUTE_NAME);
8    spSamlRequestBase64 = req.getParameter(SP_AUTH_REQ_ATTRIBUTE_NAME);
9    spSamlRequestDecoded = Base64.decode(spSamlRequestBase64);
10   spSamlRequestRelayStateStr =
       req.getParameter(SP_AUTH_REQ_RELAY_STATE_ATTRIBUTE_NAME);
11   spSamlRequestSigAlgStr = req.getParameter(SP_AUTH_REQ_SIG_ALG_ATTRIBUTE_NAME);
12   spSamlRequestSigBase64 = req.getParameter(SP_AUTH_REQ_SIG_ATTRIBUTE_NAME);
13   //[...]
14   verifyRequestSig(spSamlRequestSigAlgStr,spSamlRequestSigBase64);
15   //[...]
16   spSamlRequestStr = TTPUtils.gzdeflate(spSamlRequestDecoded);
17   //[...]
18
19   AuthnRequest spAuthnRequest = getAuthnRequestFromString(spSamlRequestStr);
20
21   //[...]
22
23   if(!db.isProviderConnectionAllowed(sp.getId(), idp.getId())){
24     throw new WayfException("The␣selected␣service␣provider␣does␣not␣allow␣a␣
       connection␣with␣this␣identity␣provider");
25   }
26   if(!db.isProviderConnectionAllowed(idp.getId(), sp.getId())){
27     throw new WayfException("The␣selected␣identity␣provider␣does␣not␣allow␣a␣
       connection␣with␣this␣service␣provider␣");
28   }
```

Listing 6.15: Teil 2 der `handleAuthentication` Funktion: Erstellung eines neuen Authentication Requests

```
1
2   //[...]
3
4   SAMLObjectBuilder<AuthnRequest> authRequestBuilder =
      (SAMLObjectBuilder<AuthnRequest>)
      builderFactory.getBuilder(AuthnRequest.DEFAULT_ELEMENT_NAME);
5   AuthnRequest authnRequest = authRequestBuilder.buildObject();
6   String authnRequestID = generator.generateIdentifier();
7   authnRequest.setID(authnRequestID);
8   authnRequest.setVersion(SAMLVersion.VERSION_20);
9   authnRequest.setIssueInstant(new DateTime());
10
11  SAMLObjectBuilder<Issuer> issuerBuilder = (SAMLObjectBuilder<Issuer>)
      builderFactory.getBuilder(Issuer.DEFAULT_ELEMENT_NAME);
12  Issuer issuer = issuerBuilder.buildObject();
13  issuer.setValue(entityID);
14  authnRequest.setIssuer(issuer);
15
16  SAMLObjectBuilder<NameIDPolicy> nameIDPolicyBuilder =
      (SAMLObjectBuilder<NameIDPolicy>)
      builderFactory.getBuilder(NameIDPolicy.DEFAULT_ELEMENT_NAME);
17  NameIDPolicy nameIDPolicy = nameIDPolicyBuilder.buildObject();
18  nameIDPolicy.setSPNameQualifier(entityID);
19  nameIDPolicy.setAllowCreate(true);
20  nameIDPolicy.setFormat("urn:oasis:names:tc:SAML:2.0:nameid-format:transient");
21  authnRequest.setNameIDPolicy(nameIDPolicy);
22
23  SingleSignOnService ssoService = new SingleSignOnServiceBuilder().buildObject();
24  Endpoint endpoint = (Endpoint) ssoService;
25  endpoint.setLocation(redirectEndpoint.getLocation());
26  endpoint.setResponseLocation(entityID+"?action=SSO-Post");
27  endpoint.setBinding(SAMLConstants.SAML2_REDIRECT_BINDING_URI);
28  authnRequest.setAssertionConsumerServiceURL(endpoint.getResponseLocation());
29  authnRequest.setDestination(endpoint.getLocation());
30
31  session.setAttribute("SPRequestLocation", authnRequest.getDestination());
32
33  HttpServletResponseAdapter responseAdapter = new HttpServletResponseAdapter(res,
      true);
34  BasicSAMLMessageContext<SAMLObject, AuthnRequest, SAMLObject> context = new
      BasicSAMLMessageContext<SAMLObject, AuthnRequest, SAMLObject>();
35  context.setPeerEntityEndpoint(endpoint);
36  context.setOutboundSAMLMessage(authnRequest);
37  context.setOutboundMessageTransport(responseAdapter);
38
39  HTTPRedirectDeflateEncoder encoder = new HTTPRedirectDeflateEncoder();
40  encoder.encode(context);
41  //[...]
42 }
```

6.4. Embedded Discovery Service

Der Embedded Discovery Service (EDS), der in Abschnitt 4.1.3.1 vorgestellt wurde, ermöglicht es einem Service Provider in eine Webseite einen Dialog einzubauen, über den der Benutzer seinen IDP auswählen kann. In der bisherigen Implementierung, zeigt dieser EDS nur die IDPs an, die bei dem SP registriert sind. Für die Verwendung der TTP benötigt der EDS daher eine Option, die es dem Benutzer erlaubt, anzugeben, dass sein IDP nicht verfügbar ist und die IDP-Auswahl an den CDS der TTP zu eskalieren. Abbildung 6.8 zeigt diese Modifizierung.

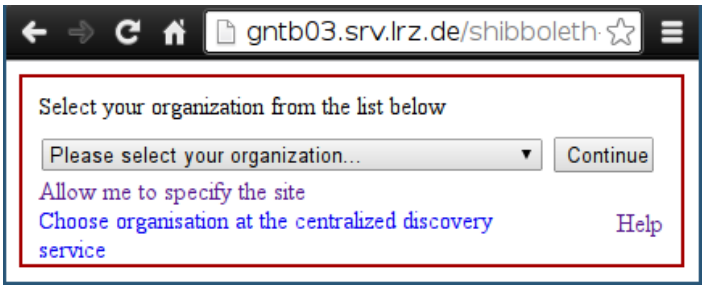

Abbildung 6.8.: Modifizierter Dialog des Embedded Discovery Service

Der Embedded Discovery Service ist vollständig in JavaScript implementiert. Die Datei `idpselect.js` enthält dabei den Programmcode und wird durch die Datei `idpselect_config.js` konfiguriert. Der für die Erweiterung relevante Teil ist in Listing 6.16 dargestellt. Die Funktion `buildIdPSelector` ist dafür verantwortlich, den DIV für die Darstellung der Auswahl aufzubauen. Diese Funktion wurde daher um einen weiteren Funktionsaufruf `buildCentralizedDSForwardTile` erweitert, wenn die Option `enableCDSForward` aktiviert ist. Der Funktionsname enthält einen Hinweis auf den Centralized Discovery Service (CDS) anstatt der TTP, da das zur Weiterleitung verwendete Protokoll nicht TTP spezifisch, sondern Teil des CDS ist und auch mit einem CDS ohne TTP verwendet werden kann. Die Funktion `buildCentralizedDSForwardTile` erhält aus der Konfiguration über `forwardCDS Host` die URL, an die der Benutzer weitergeleitet werden soll. Die Parameter, die für die CDS relevant sind, sind die gleichen Parameter, wie sie bei dem Aufruf des EDS übergeben wurden. Allerdings muss der `return` Parameter ausgetauscht werden. Anstelle des normalen SAML2SessionInitiator soll der TTP-DS die Auswahl des Benutzers an den extra dafür vorgesehenen TTPSessionInitiator des SP schicken. Daher werden alle Parameter ausgelesen, der `return` Parameter ersetzt und danach der Parameterstring wieder zusammengesetzt.

Die Konfigurationsdatei `idpselect_config.js` wurde entsprechend der Implementierung um drei Optionen erweitert. Diese Optionen sind in Listing 6.17 dargestellt.

6.5. Verwaltungs-Webfrontend

Das Webfrontend für die Trusted Third Party baut auf der Implementierung des Centralized Discovery Services, welcher in Abschnitt 6.3 beschrieben wurde, auf. Daher ist die in Abbildung 6.9 dargestellte Übersicht des Webfrontends eine Erweiterung der Übersicht des

Listing 6.16: Erweiterung des EDS um einen Weiterleitungs-Link in der Datei `idpselect.js`

```
var buildIdPSelector = function(){
  var containerDiv = buildDiv('IdPSelector');
  var preferredTileExists;
  preferredTileExists = buildPreferredIdPTile(containerDiv);
  buildIdPEntryTile(containerDiv, preferredTileExists);
  buildIdPDropDownListTile(containerDiv, preferredTileExists);
  if(enableCDSForward)
    buildCentralizedDSForwardTile(containerDiv);
  return containerDiv;
};

var buildCentralizedDSForwardTile = function(parentDiv) {
  cdsForwardDiv = buildDiv('CDSForwardTile');
  cdsLink = document.createElement('a');
  setID(cdsLink, 'CDSLink');

  var params = getParameterMap(window.location.search);
  var returnURL = decodeURIComponent(params['return']);
  var returnParams = returnURL.substr(returnURL.indexOf("?"));
  returnURL = ttpReturnPath + returnParams;
  params['return'] = encodeURIComponent(returnURL);

  var newSearch = "?";
  for(param in params){
    newSearch += (newSearch.length > 1 ? "&" : "") + param + "=" + params[param];
  }

  cdsLink.href = forwardCDSHost + newSearch;

  var nameDiv = buildDiv(undefined, 'TextDiv');
  var nameStr = getLocalizedMessage('cdsForwardLink');
  nameDiv.appendChild(document.createTextNode(nameStr));
  cdsLink.appendChild(nameDiv);
  setClass(cdsLink, 'CDSForwardButton');

  cdsForwardDiv.appendChild(cdsLink);
  parentDiv.appendChild(cdsForwardDiv);
  buildHelpText(cdsForwardDiv);
};
```

Listing 6.17: Erweiterte Konfiguration des EDS

```
this.enableTTPForward = false;
this.forwardTTPHost = 'https://ds.example.org';
this.ttpRequrnPath = 'https://sp.example.org/Shibboleth.sso/TTP'
```

CDS aus Abbildung 6.6. Hinzugefügt wurden Servlets für Login, User-, Provider-, Provider-User-Beziehungs-, Metadaten- und Konvertierungsregel-Management. Die Funktionen dieser Servlets werden in den folgenden Abschnitten näher erleutert.

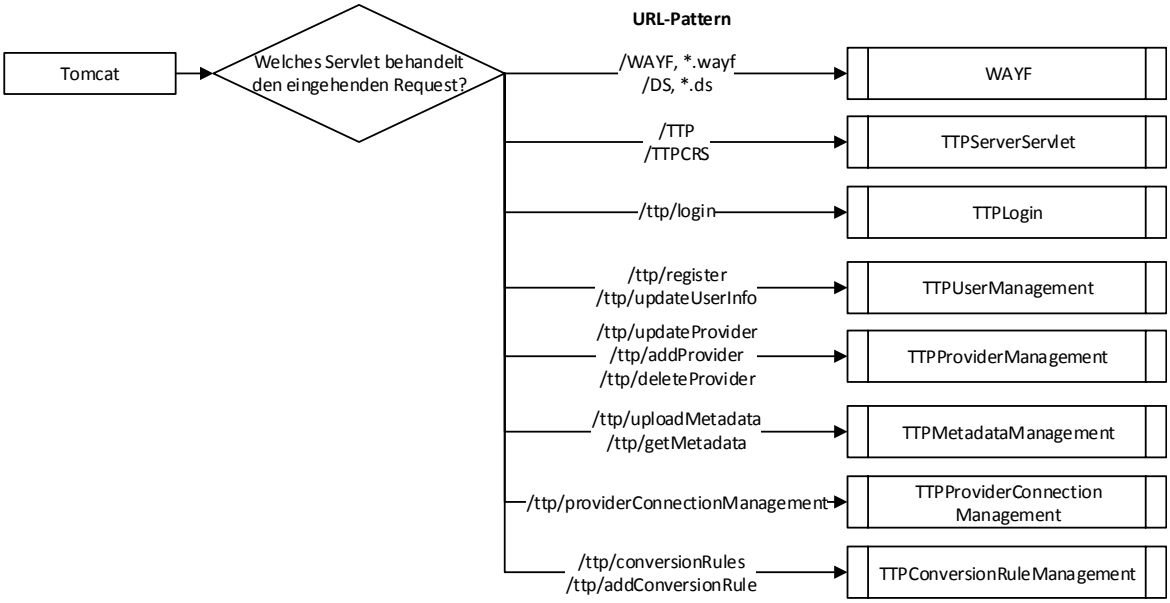

Abbildung 6.9.: Übersicht über die Implementierung des Webfrontends

Neben den Tomcat-Servlets bzw. JSP-Seiten, die die Anwendungslogik sowie die Darstellung der Seiten übernehmen, wird das Bootstrap-Framework (`http://getbootstrap.com/`) für die Darstellung der Webseiten verwendet.

Das Webfrontend verwendet dabei ebenfalls die Datenbank, die für den CDS implementiert wurde und in Abbildung 6.7 dargestellt ist.

6.5.1. TTPLogin

Das Login-Servlet hat die Aufgabe, einen eingehenden HTTP-POST-Request an die URL `/ttp/login` zu empfangen und dort die Parameter `loginname` und `password` auszulesen. Wenn die Parameter vorhanden sind, wird versucht aus der Datenbank das entsprechende Salt auszulesen. Dieses Salt wird vor das von dem Benutzer übermittelte Passwort geschrieben und aus dieser Kombination ein Hash berechnet. Hierbei wird der SHA-2 Hashing-Algorithmus in der SHA-512 Version verwendet, der unter anderem durch das Bundesamt für Sicherheit in der Informationstechnik (BSI) als kryptographisch sicherer Hash beschrieben wird [BSIKV14, Kapitel 4]. Daraufhin überprüft das Servlet, ob die Kombination von `loginname` und Passwort-Hash in der Datenbank vorhanden sind und in dem Fall, dass ein entsprechender Eintrag gefunden wurde, wird eine neue Sitzung für den Benutzer initialisiert. In dieser Sitzung wird ein Benutzer-Objekt gespeichert, das die Benutzer ID sowie die Namen des Benutzers enthält.

Nach der Anmeldung wird der Benutzer zurück zur Startseite geleitet, wo anhand des Sitzungsobjektes festgestellt wird, dass der Benutzer angemeldet ist und daher Optionen, die

anderen Verwaltungsfunktionen nutzen zu können, bereitstehen.

6.5.2. TTPUserManagement

Mit Hilfe des User-Management-Servlets können neue Benutzer registriert werden. Dazu werden die durch die Registrierungs-Maske als HTTP-POST-Request an die URL `/ttp/register` übermittelten Daten ausgelesen und zunächst validiert. Für das Registrieren eines neuen Benutzers werden folgende Parameter übermittelt: `loginname`, `username`, `givenName`, `sn`, `mail`, `password` und `password2`

Zunächst wird überprüft, dass alle Felder, die durch die Datenbank vorgegebenen Zeichenanzahl, einhalten und benötigte Felder, wie der `loginname`, `username`, `mail` und `password`, nicht leer sind sowie dass `password` und `password2` übereinstimmen.

Daraufhin wird ein neues Salt für den Benutzer erstellt. Hierzu werden durch die Java-Bibliothek `java.security.SecureRandom.SecureRandom()` 512 Byte Zufallsdaten generiert, die mit Hilfe des SHA-512 Hashes in eine Zeichenkette, die in der Datenbank abgelegt werden kann, gewandelt werden. Wie bei dem Login beschrieben, wird der Password-Hash, der ebenfalls in der Datenbank gespeichert werden muss dadurch generiert, dass das vom Benutzer vergebene Passwort an das Salt gehängt wird und beides mit SHA-512 gehasht wird.

Beim Einfügen des neuen Benutzers in die Datenbank wird durch die `UNIQUE` Einschränkung für `loginname` und `username` überprüft ob einer der beiden Namen schon vergeben ist. Durch dieses Vorgehen ist diese Operation atomar und es können keine Race-Conditions auftreten.

6.5.3. TTPProviderManagement

Über das Provider-Management kann ein Benutzer einen neuen Provider anlegen, bzw. vorhandene Provider modifizieren. Hierzu werden die URLs `/ttp/updateProvider`, `/ttp/addProvider` und `/ttp/deleteProvider` verwendet, an die für eine Aktion jeweils ein HTTP-POST-Request geschickt wird.

Das Hinzufügen eines neuen Providers funktioniert sehr ähnlich zu dem Hinzufügen eines neuen Benutzers. Die benötigten Parameter `entityID`, `type`, `organization` und `description` werden aus dem Request ausgelesen, auf Existenz sowie Einhaltung von Längenbegrenzungen validiert und in die Datenbank eingefügt.

Bevor eine Änderung oder Löschung eines Providers durchgeführt wird, überprüft das Servlet anhand der `providerUserRelationship`-Tabelle der Datenbank, ob der Benutzer dazu berechtigt ist, Änderungen an diesem Provider vorzunehmen.

Beim Aktualisieren der Einstellungen über `/ttp/updateProvider` wird in dem POST-Request ein `action`-Parameter verwendet um anzuzeigen, was für eine Änderung der Benutzer durchführen möchte. Mögliche Aktionen sind `updateDescription`, für das Aktualisieren der Beschreibung des Providers, sowie `setActiveMetadata`, um einen bestimmten Metadatensatz zu aktivieren.

6.5.4. TTPProviderConnectionManagement

Dafür, dass ein Benutzer nachweisen kann, dass er dazu berechtigt ist, Einstellungen an einem Provider vorzunehmen, wurden in Abschnitt 5.4.2.1 verschiedene Methoden vorgestellt. Hier wird die Methode implementiert, die den Benutzer dazu auffordert eine bestimmte Datei auf der Domain seines Providers zu hosten. Diese Methode ist allgemein gut geeignet, da keine von einer CA unterschriebenen Zertifikate benötigt werden, sondern ein einfacher Webserver, der zum Betreiben eines IDP oder SP sowieso benötigt wird, ausreicht.

Dazu wird aus der EntityID des Providers, für den der Benutzer sich registrieren möchte, die Domain extrahiert. Für die Erstellung des zufälligen URL-Pfades werden über die Bibliothek, die auch für das Salt des Benutzers verwendet wurde, `java.security.SecureRandom.SecureRandom()`, 128 Byte Zufallsdaten generiert und als Hexadezimalzahl dargestellt. Hier werden nur 128 Byte Zufallsdaten verwendet, da das Erzeugen guter Zufallszahlen viel Zeit kostet und mit dem kurzen Gültigkeitszeitraum dieses Parameters weitestgehend ausgeschlossen ist, dass eine identische Datei bei dem Provider schon existiert. Durch diesen Prozess wird zum Beispiel aus der EntityID `https://idp.example.com/idp/shibboleth` die URL `http://idp.example.com/1o1gh7lfd4fe4kac5gaa4ksahd`. Diese URL wird in der Sitzung des Benutzers gespeichert. Sobald der Benutzer bestätigt, dass er die entsprechende Datei angelegt hat, versucht das Servlet die Datei abzurufen und bestätigt den Providerzugriff des Benutzers, falls der Server mit dem Status `200 Ok` antwortet. In jedem anderen Fall wird der Vorgang abgebrochen. Der Benutzer kann dann die Konfiguration seines Servers überprüfen und den Vorgang wiederholen.

6.5.5. TTPMetadataManagement

Die Metadata-Management Funktion erlaubt es einem Benutzer, neue Metadaten hochzuladen. Eine Modifikation von Metadaten ist wegen der Versionierung fast identisch mit dem Hinzufügen neuer Metadaten. Der einzige Unterschied ist, dass beim Hochladen modifizierter Metadaten in der Datenbank das `parent`-Attribut gesetzt werden muss.

Das Metadata-Management-Servlet erwartet für das Hinzufügen von Metadaten unter der URL `/ttp/uploadMetadata`, dass mit einem HTTP-POST-Request die Parameter `metadata`, mit dem Text-Inhalt der Metadaten, falls vorhanden `metadataParent`, mit der ID der Parent-Metadaten und `comment`, mit einer Beschreibung der Metadaten-Version bzw. der Änderungen übermittelt werden. Der Text der Metadaten wird daraufhin über die dazugehörigen Shibboleth-Funktionen in die entsprechende Java-Objekt-Repräsentation der Metadaten zu parsen. Sollte dies fehlschlagen wurden keine gültigen Metadaten übermittelt und der Vorgang wird abgebrochen. Nach dem erfolgreichen Parsen wird außerdem überprüft, ob die in den Metadaten angegebene EntityID zu der EntityID eines Providers, den der Benutzer bearbeiten darf, passt. Ist auch diese Validierung erfolgreich, werden die Eigenschaften der Metadaten in der Datenbank gespeichert und die Provider-Metadaten in einer Datei, deren Name sich, wie in Abschnitt 5.4.2.3 beschrieben, aus dem SHA-1 Hash der EntityID und dem aktuellen Zeitpunkt zusammensetzt, gespeichert.

Für das Editieren der Metadaten in dem Webfrontend des Prototypen wurde die JavaScript-Plugin SAMLmetaJS (`http://samlmetajs.simplesamlphp.org/`) verwendet, die eine über-

sichtliche formularbasierte Bearbeitung der Metadaten ermöglicht. Abbildung 6.10 zeigt einen Screenshot der Verwaltungswebseite, auf der das Plugin zu sehen ist. Außerdem weist das Plugin den Benutzer auf fehlende Einträge in den Metadaten hin. Allerdings hat sich in ersten Tests herausgestellt, dass diese Bibliothek nicht alle von Shibboleth benötigten Elemente, wie das `AssertionConsumerService`-Element, in den Metadaten behandeln kann und diese daher aus den Metadaten löscht. Dadurch werden die Metadaten leider für Shibboleth unbrauchbar, so dass dieses Plugin nur zeigt, wie eine benutzerfreundliche Bearbeitung der Metadaten aussehen könnte, praktisch allerdings nicht einsetzbar ist.

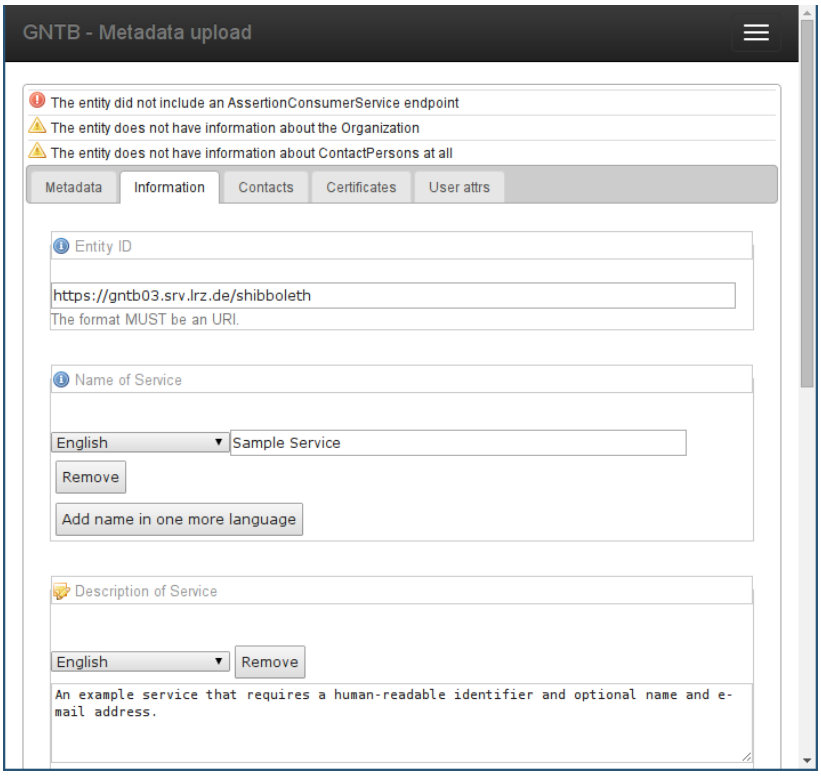

Abbildung 6.10.: SAMLmetaJS in dem Webfrontend der TTP

6.5.6. TTPConversionRuleManagement

Die Verwaltung der Konvertierungsregeln ist ähnlich der Verwaltung der Metadaten. Allerdings muss bei den Konvertierungsregeln nicht darauf geachtet werden, dass sie einem bestimmten Provider, bei dem der Benutzer zusätzlich Schreibrechte haben muss, zugeordnet werden.

Das Servlet erwartet zum Hinzufügen einen HTTP-POST-Request unter der URL `/ttp/addConversionRule` die Parameter `name`, mit einem Namen für die Regel, `target`, mit der ID des Attributs, dass durch die Regel erzeugt wird, `sources`, ein Array von Attributs IDs, von denen die Regel abhängt, `description`, mit einer kurzen Beschreibung der Regel und `code`, mit dem eigentlichen Regel-Code.

Wenn auch bei diesen Parametern die Validierung erfolgreich war, d.h. die Parameter innerhalb ihrer Zeichengrenzen waren und die referenzierten Attribute wirklich existieren, wird die Regel in die Datenbank eingefügt und der Code der Regel in eine Datei gespeichert. Wie in Abschnitt 5.4.4 vorgeschlagen wird hier der SHA-1 Hash des Regelnamens sowie der aktuelle Zeitpunkt für den Dateinamen verwendet.

7. Evaluation & Test

Dieses Kapitel beschreibt im ersten Abschnitt zunächst die Umgebung, in der die Tests durchgeführt wurden. Daraufhin werden in Abschnitt 7.2 die Durchführung des Tests und dessen Ergebnisse beschrieben. Der letzte Abschnitt 7.3 beschreibt, wie jeder das System testen kann, in dem er entweder die vorhandenen SPs und IDPs verwendet oder einen neuen eigenen Dienst in das TTP-Netzwerk integriert. Abschließend wird in Abschnitt 7.4 betrachtet, welche Anforderungen durch die Implementierung aus dem vorhergehenden Kapitel umgesetzt wurden.

7.1. Beschreibung der Testumgebung

Die Testumgebung besteht aus acht virtuellen Maschinen, die am LRZ gehostet sind und auf denen die stabile Version `7.6` „Wheezy“ der Linux-Distribution Debian läuft. Diese Maschinen wurden für die Tests aufgeteilt, sodass mehrere SPs und IDPs zur Verfügung stehen, die über die TTP kommunizieren können. Als Software werden bei einigen Hosts leicht verschiedene Versionen eingesetzt, was durchaus reale Bedingungen wiederspiegelt, da nicht immer alle Dienste und Provider die neuste Version installiert haben. Verwendet wurden die Versionen `2.4.0` und `2.4.2` des Shibboleth Identity Providers. Die neueste Version `3`, für die es momentan nur eine Alphaversion gibt, ließ sich auf einem Testsystem nicht installieren. Zum einen, passte die sehr kurze Anleitung für diese Alphaversion nicht zu den in der Alphaversion vorhandenen Dateien. Zum anderen gab es Konflikte durch nicht aktuelle Java-Libraries in der Debian Installation. Für den Shibboleth Service Provider wurde hauptsächlich die Version `2.4.3`, die, im Gegensatz zum IDP direkt aus den Debian Paketquellen, über das Paket `libapache2-mod-shib2` installiert werden kann, verwendet. Ein System wurde mit der aktuellsten Version `2.5.3` aufgesetzt, wofür der SP und einige seiner Abhängigkeiten manuell kompiliert werden mussten.

Mit diesem Aufbau ist ein Szenario, wie es in Abbildung 7.1 dargestellt ist, umgesetzt worden. Dieses Szenario versucht, durch zwei Föderationen und zwei externe Partner, möglichst

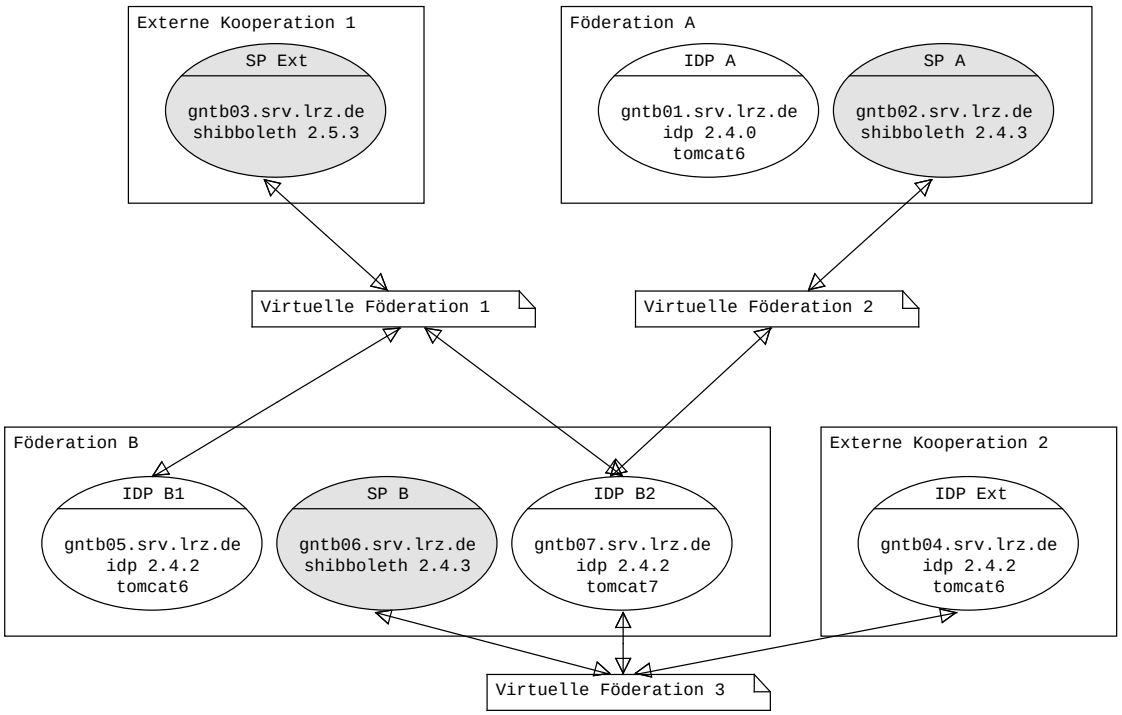

Abbildung 7.1.: Aufbau der Testumgebung ohne TTP

viele in der Praxis vorkommende Konstellationen nachzustellen. In den meisten realen Fällen würde eine Situation mit nur einem IDP, wie in der externen Kooperation 2, nicht vorkommen. Organisationsinterne SPs, die nicht relevant sind, wurden hier nicht dargestellt. Durch diesen Aufbau ist direkt erkennbar, dass die durch „nationale“ Grenzen bestimmten Föderationen nicht geeignet sind, um den virtuellen Föderationen von Nutzen zu sein. Die drei virtuellen Föderationen, die gemeinsam Dienste verwenden wollen, sind:

- **Virtuelle Föderation 1 (VF1)**: Die IDPs der Föderation B möchten ihren Benutzern Zugriff zu dem externen Dienstleister `SP Ext` gewähren. Ein reales Beispiel für diesen Fall wäre, dass Microsoft, in der Rolle des SPs, es Studenten ermöglicht sich über die IDPs ihrer Universitäten für Studentenrabatte zu registrieren.

- **Virtuelle Föderation 2 (VF2)**: Ein IDP aus der Föderation B, IDP B2, möchte seinen Benutzern Zugriff zu dem Service Provider aus Föderation A geben. Ein Beispiel hierfür wäre es, dass Wissenschaftler über die Grenzen von nationalen Föderationen hinweg kooperieren wollen.

- **Virtuelle Föderation 3 (VF3)**: Der IDP B2 und der SP aus Föderation B möchten zusammen mit einem externen IDP, IDP Ext, arbeiten. Diese Situation könnte auftreten, wenn ein Unternehmen, das selbst einen IDP für seine Mitarbeiter betreibt, an einem Projekt einer Universität, die in dem Fall IDP B2 und den SP B betreibt, teilnimmt und die Mitarbeiter des Unternehmens Zugang zu einem Dienst der Universität benötigen.

Die verschiedenen Dienste benötigen für ihre Nutzung unterschiedliche Attribute. In Tabelle 7.1 ist zusammengefasst, welcher SP welche Attribute benötigt. Der externe SP benötigt nur eine für jeden Benutzer dauerhafte und eindeutige ID (`eduPersonTargetedID`) sowie die Gruppenzugehörigkeit des Benutzers (`eduPersonAffiliation`). Damit kann der Dienst, der im Beispiel den Studentenstatus eines Benutzers überprüfen möchte, den Benutzer eindeutig bestimmen und entscheiden, ob er der Gruppe der Studenten angehört.

SP A und B sind Dienste, die von zwei unterschiedlichen Forschungsgruppen verwendet werden. Dazu benötigen sie diverse Informationen über den Benutzer. SP A, fordert für eine genaue Identifizierung des Benutzers dessen ID (`eduPersonTargetedID`), den Namen des Benutzers (`cn`), die Gruppenzugehörigkeit innerhalb der Organisation mit Scope (`eduPerson ScopedAffiliation`), den Benutzernamen (`eduPersonPrincipalName`) sowie ein Geburtsdatum (`schacDateOfBirth`).

SP B kommt mit weniger Attributen aus und fordert daher nur eine eindeutige ID sowie Vor- und Nachname des Benutzers und eine E-Mail-Adresse.

Provider	**Host**	**Geforderte Attribute**
SP Ext	gntb03	eduPersonTargetedID
		eduPersonAffiliation
SP A	gntb02	eduPersonTargetedID
		cn
		eduPersonScopedAffiliation
		eduPersonPrincipalName
		schacDateOfBirth
SP B	gntb06	eduPersonTargetedID
		givenName
		sn
		mail

Tabelle 7.1.: Durch die SPs der Testumgebung geforderten Attribute

Entsprechend den Service Providern liefern die Identity Provider unterschiedliche Attribute, welche in Tabelle 7.2 dargestellt sind. Die Attribute von SPs und IDPs sind dabei so gewählt, dass sie den realen Anforderungen möglichst nahe kommen. D.h. innerhalb von Föderationen ist der Betrieb der Dienste möglich, föderationsübergreifend fehlt allerdings ein Teil der benötigten Attribute.

7.2. Durchführung des Tests

Der Test besteht aus zwei Schritten. Im ersten Schritt, der im nächsten Abschnitt näher beschrieben ist, wird die im vorherigen Abschnitt beschriebene Testumgebung mit herkömlichen Mitteln, wie Föderationen und explizitem Metadatenaustausch, umgesetzt. Im darauffolgenden Abschnitt 7.2.2 wird der gleiche Aufbau verwendet, allerdings treten die Systeme der Trusted Third Party bei und stellen über diese, die nötigen Verbindungen her.

Gemessen wird dabei zum einen, wie viele Metadatensätze lokal und global explizit konfiguriert werden müssen. Zum anderen soll gezeigt werden, dass über geeignete Konvertie-

Provider	Host	Verfügbare Attribute
IDP A	gntb01	eduPersonTargetedID
		cn
		mail
		eduPersonScopedAffiliation
		eduPersonPrincipalName
		schacDateOfBirth
IDP B1	gntb05	eduPersonTargetedID
		givenName
		sn
		mail
		eduPersonAffiliation
IDP B2	gntb07	eduPersonTargetedID
		givenName
		sn
		mail
		eduPersonAffiliation
		norEduPersonBirthDate
IDP Ext	gntb04	uid
		cn
		mail

Tabelle 7.2.: Durch die IDPs der Testumgebung direkt verfügbaren Attribute

rungsregeln die unterschiedlichen geforderten und bereitgestellten Attribute, automatisch angepasst werden können.

7.2.1. Ohne dynamische virtuelle Föderationen

Ohne das Hinzufügen einer TTP und die dadurch möglichen dynamischen virtuellen Föderationen, müssen grundsätzlich innerhalb der Föderationen Metadaten ausgetauscht werden. Daher müssen die Administratoren der Dienste und Provider aus den Föderationen zum einen ihre Metadaten bei der Föderation hinterlegen und zum anderen die Föderationsmetadaten installieren. Somit müssen IDP A, SP A, IDP B1, IDP B2 und SP B auf jeden Fall einen Metadatensatz konfigurieren. Im Folgenden werden schrittweise die zu installierenden Metadaten aufgelistet und in jedem Schritt die aktuellen Zwischenstände angegeben. Dabei werden die hinzugekommenen Metadateneinträge fett gedruckt. In der bisher beschriebenen Ausgangssituation ergibt sich der in Tabele 7.3 dargestellte Zustand.

Für die Kommunikation in der virtuellen Föderation 1, zwischen IDP B1, IDP B2 und SP Ext gibt es für die beteiligten Dienste zwei Möglichkeiten, die Kommunikation zwischen den Diensten zu ermöglichen. Zum einen können jeweils weitere Metadaten installiert werden, d.h. IDP B1 und IDP B2 installieren manuell die Metadaten von SP Ext und SP Ext die von IDP B1 und IDP B2. Zum anderen könnte der SP Ext der Föderation beitreten und müsste daher nur einen Metadatensatz installieren. Die IDPs würden in diesem Fall automatisch über die aktualisierten Metadaten der Föderation die Metadaten des SP Ext bekommen. Ob

Provider	Installierte Metadaten
IDP A	**FöderationA.xml**
SP A	**FöderationA.xml**
IDP B1	**FöderationB.xml**
IDP B2	**FöderationB.xml**
SP B	**FöderationB.xml**
IDP Ext	–
SP Ext	–

Tabelle 7.3.: Ausgangszustand mit Föderationen: Insgesamt 5 Metadatensätze

ein Dienst in eine Föderation aufgenommen werden kann, hängt hauptsächlich von organisatorischen Voraussetzungen ab, daher werden beim Zählen der zu installierenden Metadatensätze Zahlen beider Methoden geführt. Der Zwischenstand nach dem Konfigurieren der VO1 ist in Tabelle 7.4 abgebildet.

Provider	Installierte Metadaten	
	Mit Föderationsbeitritt	**Ohne Föderationsbeitritt**
IDP A	FöderationA.xml	FöderationA.xml
SP A	FöderationA.xml	FöderationA.xml,
IDP B1	FöderationB.xml	FöderationB.xml, **SPExt.xml**
IDP B2	FöderationB.xml	FöderationB.xml, **SPExt.xml**
SP B	FöderationB.xml	FöderationB.xml
IDP Ext	–	–
SP Ext	**FöderationB.xml**	**IDPB1.xml**, **IDPB2.xml**

Tabelle 7.4.: Zwischenstand mit Föderationen und VF1: Insgesamt 6 bis 10 Metadatensätze

Bei der Kommunikation zwischen den Parteien, SP A und IDP B2, der virtuellen Föderation 2 bestehen prinzipiell auch zwei Möglichkeiten. Die beiden Föderationen A und B könnten einer Inter-Föderation beitreten, wodurch die beiden beteiligten Dienste jeweils den Inter-Föderations-Metadatensatz installieren müssten. Alternativ können sie auch direkt die Metadaten des jeweils anderen installieren, sodass die Anzahl der zu installierenden Metadatensätze bei beiden Optionen gleich ist. Tabelle 7.5 zeigt den Zwischenstand nach dem Hinzufügen von VF2.

Für die virtuelle Föderation 3 gibt es wie bei der VF1 die Möglichkeiten, dass der externe IDP der Föderation beitritt oder die Metadaten direkt ausgetauscht werden. Der Beitritt des IDPs zur Föderation ist hierbei, aus Sicht der zu installierenden Metadaten, leicht zu bevorzugen. Der IDP Ext müsste dafür nur die Metadaten der Föderation konfigurieren. Es ist aber eventuell nicht möglich, dass der IDP der Föderation beitritt und in diesem Fall müsste der IDP Ext die Metadaten des SP B und umgekehrt installieren. Der End-Zustand der Konfiguration aller virtuellen Föderationen wird in Tabelle 7.6 dargestellt.

Unter der Annahme, dass es den externen Diensten bzw. Providern möglich ist, den Födera-

Provider	Installierte Metadaten	
	Mit Föderationsbeitritt	**Ohne Föderationsbeitritt**
IDP A	FöderationA.xml	FöderationA.xml
SP A	FöderationA.xml, **InterFöderationAB.xml**	FöderationA.xml, **IDPB2.xml**
IDP B1	FöderationB.xml	FöderationB.xml, SPExt.xml
IDP B2	FöderationB.xml, **InterFöderationAB.xml**	FöderationB.xml, SPExt.xml, **SPA.xml**
SP B	FöderationB.xml	FöderationB.xml
IDP Ext	–	–
SP Ext	FöderationB.xml	IDPB1.xml, IDPB2.xml

Tabelle 7.5.: Zwischenstand mit Föderationen, VF1 und VF2: Insgesamt 8 bis 11 Metadatensätze

tionen, in denen die Dienste bzw. Provider vertreten sind, beizutreten, müssen insgesamt 9 Metadatensätze konfiguriert werden. Diese Lösung führt allerdings dazu, dass die Gesamtmetadaten der Föderation immer größer werden und Dienste bzw. Provider enthalten, die für den Rest der Föderation eventuell uninteressant sind. Die Alternative, die Metadaten bei den Diensten, die sie wirklich benötigen zu installieren, führt dazu, dass mehr Metadaten installiert werden müssen, aber die anderen Teilnehmer der Föderation davon nicht betroffen sind.

Eine automatisierte Konvertierung von Attributen ist mit herkömmlichen Methoden nicht möglich, daher müssen die Konvertierungen manuell installiert oder konfiguriert werden. In der Testumgebung fehlen dem IDP B2 für seine Verbindung zu SP A die Attribute `cn`, `eduPersonScopedAffiliation`, `eduPersonPrincipalName` und `schacDateOfBirth`. Dem IDP Ext fehlen für die Verbindung zu SP B die Attribute `eduPersonTargetedID`, `givenName` sowie `sn`.

Der IDP B2 kann allerdings den `cn` durch die Konkatenation von `givenName` und `sn` erstellen. Die `eduPersonScopedAffiliation` kann er aus der `eduPersonAffiliation` durch das Anhängen eines Scopes ableiten und für die `eduPersonPrincipalName` die `mail` des Benutzers verwenden. Das Attribut `schacDateOfBirth` ist identisch zu dem vorhandenen `norEduPersonBirthDate`, sodass dieses nur umbenannt werden muss.

Auch der IDP Ext kann die nicht passenden Attribute aus vorhandenen ableiten. Als `eduPersonTargetedID` kann er zum Beispiel die `uid` verwenden. Um allerdings die Daten des Benutzers zu schützen, ist es besser die `uid` nicht direkt zu verwenden, sondern sie mit der EntityID des Zieles zu kombinieren und daraus einen Hash zu generieren. Das Ergebnis erfüllt weiterhin die Anforderung einer für jeden Dienst eindeutigen ID, ohne den Benutzer anhand der `uid` über mehrere Dienste hinweg verfolgbar zu machen. Das Ableiten von `givenName` und `sn` kann prinzipiell durch das Aufspalten des `cn` geschehen.

Provider	**Installierte Metadaten**	
	Mit Föderationsbeitritt	**Ohne Föderationsbeitritt**
IDP A	FöderationA.xml	FöderationA.xml
SP A	FöderationA.xml, InterFöderationAB.xml	FöderationA.xml, IDPB2.xml
IDP B1	FöderationB.xml	FöderationB.xml, SPExt.xml
IDP B2	FöderationB.xml, InterFöderationAB.xml	FöderationB.xml, SPExt.xml, SPA.xml, **IDPExt.xml**
SP B	FöderationB.xml	FöderationB.xml, **IDPExt.xml**
IDP Ext	**FöderationB.xml**	**SPB.xml**, **IDPB2.xml**
SP Ext	FöderationB.xml	IDPB1.xml, IDPB2.xml

Tabelle 7.6.: Zwischenstand mit Föderationen, VF1, VF2 und VF3: Insgesamt 9 bis 15 Metadatensätze

7.2.2. Verwendung einer Trusted Third Party

Bei der Verwendung der Trusted Third Party müssen alle beteiligten IDPs nur einen Metadatensatz für die TTP hinzufügen, sowie die Erweiterung installieren und konfigurieren. Ein SP muss nur den MetadataProvider für die TTP, aber nicht deren Metadaten konfigurieren. Daher müssen nach dieser Zählweise in diesem System nur vier Metadatensätze installiert werden. Dies ist deutlich weniger als mit der herkömmlichen Methode, führt allerdings zu einem initial größeren Aufwand für die Installation und Konfiguration der Systeme. Im Laufe der Zeit wird dieser Aufwand allerdings dadurch mehr als ausgeglichen, dass für neue virtuelle Föderationen keine zusätzliche Konfiguration notwendig ist. Abbildung 7.2 zeigt den Aufbau der Testumgebung mit der hinzugefügten TTP.

Um den konkreten Aufwand der Einrichtung der Erweiterung abzuschätzen, wird hier nun ein Überblick über die durchzuführenden Schritte gegeben. Eine genaue Installationsanleitung für die jeweilige Erweiterung wird mit dem aktuellen Sourcecode in der README-Datei geliefert und ist für den IDP in Anhang B sowie für den SP in Anhang C angegeben.

Für die Installation der Erweiterung bei einem IDP, muss zunächst die aktuellste Version der Erweiterung von `http://gntb08.srv.lrz.de` heruntergeladen und entpackt werden. Das Archiv enthält dabei einen Ordner `lib`, der die als `.jar` gepackten Teile der Erweiterung enthält. Diese müssen in das `lib`-Verzeichnis des Ordners, aus dem der IDP installiert wurde, kopiert werden. Um die neuen Funktionen zu aktivieren, muss dort die Datei `src/main/webapp/WEB-INF/web.xml` um die neuen Servlets erweitert werden. Daraufhin ist wie bei einem Update des IDPs das `install.sh`-Skript auszuführen, welches die aktualisierten Dateien in eine `.war`-Datei packt und diese auf dem Tomcat installiert. Zum Schluss muss noch der neue MetadataProvider in der `relying-party.xml` konfiguriert und der Tomcat-Server

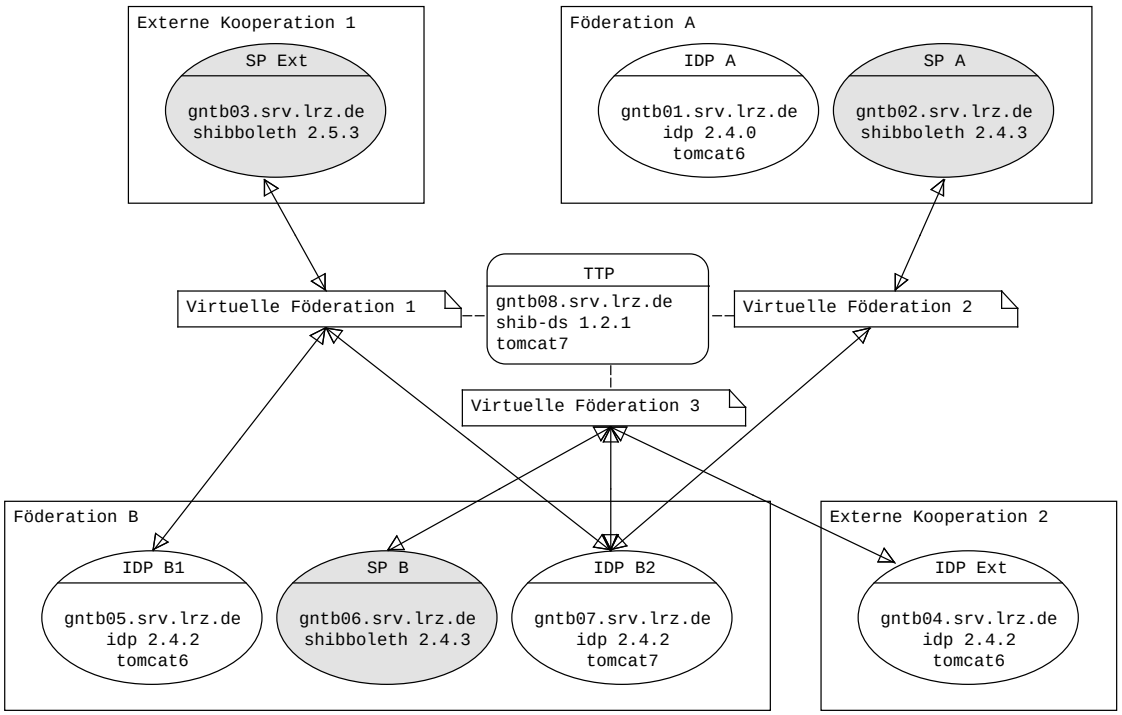

Abbildung 7.2.: Aufbau der Testumgebung mit TTP

neugestartet werden.

Bei der Installation der Erweiterung auf einem SP muss zunächst die Shared-Library, die danach in den Apache2 sowie den Shibd-Prozess geladen wird, kompiliert werden. Dieser Schritt ist der aufwändigste, da die Abhängigkeiten für das Kompilieren nicht alle direkt über die Paketquellen von Debian „Wheezy“ verfügbar sind. Für aktuelle Versionen der XMLTooling- und SAML-Bibliotheken müssen diese aus den Backports installiert werden. Danach kann die Erweiterungs-Bibliothek kompiliert werden. Die Verwendung dieser Bibliothek muss dann noch in der `shibboleth2.xml`-Konfigurationsdatei des SPs aktiviert werden. Zusätzlich sind in der Konfigurationsdatei der neue SessionInitiator, MetadataSyncHandler sowie der neue MetadataProvider zu konfigurieren. Anschließend müssen der Apache2- sowie der Shibd-Dienst neugestartet werden, woraufhin die Erweiterung verwendbar sein sollte.

Die Installation der Erweiterungen erhöht die Installations- und Konfigurations-Aufwände der zugrunde liegenen Komponenten etwas. Bei IDPs für die es viele weitere Erweiterungen (zum Beispiel uApprove) gibt, die nach dem gleichen Schema zu installieren sind, ist dieser Vorgang vermutlich den zuständigen Administratoren bereits bekannt. Die Installation des SPs ist im Moment durch das manuelle Installieren von Abhängigkeiten und dem Kompilieren der Erweiterung etwas aufwändiger. Die TTP hat nach der Installation allerdings den Vorteil, dass nachträglich keine weitere manuelle Konfiguration notwendig ist und man von der automatischen Synchronisation von Konvertierungsregeln profitiert.

Im vorherigen Abschnitt wurde das Vorgehen für die Konvertierung der notwendigen Attribute in der Testumgebung bereits kurz beschrieben. Um zu zeigen, wie diese Konvertie-

Listing 7.1: Konvertierung von `givenName` und `sn` zu `cn`

```
<!-- [...] -->
  <xsl:element name="resolver:AttributeDefinition">
    <xsl:attribute name="id">commonName</xsl:attribute>
    <xsl:attribute name="xsi:type">ad:Script</xsl:attribute>

    <xsl:element name="resolver:Dependency">
      <xsl:attribute name="ref">
        <xsl:value-of select="$givenName/@id"/>
      </xsl:attribute>
    </xsl:element>
    <xsl:element name="resolver:Dependency">
      <xsl:attribute name="ref">
        <xsl:value-of select="$surname/@id"/>
      </xsl:attribute>
    </xsl:element>

    <xsl:element name="resolver:AttributeEncoder">
      <xsl:attribute name="xsi:type">enc:SAML1String</xsl:attribute>
      <xsl:attribute name="name">urn:mace:dir:attribute-def:cn</xsl:attribute>
    </xsl:element>
    <xsl:element name="resolver:AttributeEncoder">
      <xsl:attribute name="xsi:type">enc:SAML2String</xsl:attribute>
      <xsl:attribute name="name">urn:oid:2.5.4.3</xsl:attribute>
      <xsl:attribute name="friendlyName">cn</xsl:attribute>
    </xsl:element>

    <ad:Script><![CDATA[
      importPackage(
        Packages.edu.internet2.middleware.shibboleth.common.attribute.provider);
      commonName = new BasicAttribute("commonName");
      if ((typeof givenName != "undefined" && givenName != null &&
    givenName.getValues().size()>0) &&
          (typeof surname != "undefined" && surname != null &&
    surname.getValues().size()>0))
        {
          commonName.getValues().add(givenName.getValues().get(0)+"
    "+surname.getValues().get(0));
        }
    ]]></ad:Script>
  </xsl:element>
<!-- [...] -->
```

rungen durch automatisierte Konvertierungsregeln umsetzbar sind, wird in Listing 7.1 eine skriptbasierte Konvertierungsregel vorgestellt. Eine vollständige XSLT für die Konvertierung durch Umbenennen, wie für die Konvertierung von `norEduPersonBirthDate` zu `schacDateOfBirth`, wurde in Abschnit 5.3.3 vorgestellt. Der XSLT-Teil außerhalb der AttributeDefinition ist dabei gleich.

Das Attribut `cn` kann aus der Verkettung von `givenName` und `sn` hergestellt werden. Diese Verkettung wird in einer AttributeDefinition durch ein Skript implementiert. Der Attributs-Definitions-Teil dieser Konvertierungsregel ist in Listing 7.1 dargestellt.

Auf ähnliche Weise, kann in einem Skript das Attribut `eduPersonScopedAffiliation` erstellt werden, indem die `eduPersonAffiliation` mit dem Domainnamen des `mail`-Attributs verknüpft wird. Da auf alle Java-Klassen innerhalb eines Skriptes zugegriffen werden kann, können auch Hashfunktionen wie zum Beispiel `DigestUtils.md5Hex()` aus dem Paket `org.apache.commons.codec.digest` verwendet werden, um aus der `uid` einen Hash für eine

`eduPersonTargetedID` zu generieren.

Die TTP ist daher dazu geeignet Metadaten auszutauschen und reduziert den Administrationsaufwand für das Einrichten von virtuellen Föderationen. Außerdem kann durch die Definition von Konvertierungsregeln ein automatischer Abgleich von Attributen stattfinden. Neben weiteren Funktionen, die über das Webfrontend der TTP konfiguriert werden können erfüllt der Prototyp der Implementierung daher die an ihn gestellten Anforderungen. Im nächsten Abschnitt wird beschrieben, wie zu Testzwecken weitere IDPs und SPs dem TTP-Netzwerk beitreten können.

7.3. Zugang zum Testsystem

Das Testsystem ist vorerst öffentlich zugänglich, sodass zum Testen jeder Interessierte eigene Test-SP oder Test-IDP in das System integrieren kann. Dazu muss zunächst ein neuer Account bei der TTP, die unter `http://gntb08.srv.lrz.de:8080/discovery/ttp/index.jsp` zu erreichen ist, registriert werden. Dieser Account muss daraufhin freigeschaltet werden, wozu eine kurze Mail an geant-trustbroker@lists.lrz.de ausreicht. Daraufhin kann ein neuer Provider angelegt werden. Für diesen muss dann, durch das Anlegen einer bestimmten Datei in dessen Domain, bewiesen werden, dass der Benutzer die Kontrolle über den Provider hat. Daraufhin können die Metadaten des Providers zur TTP hinzugefügt werden.

Auf dem hinzuzufügenden SP oder IDP muss die Erweiterung entsprechend der Beschreibung in der Dokumentation, die im vorherigen Abschnitt kurz beschrieben wurde, konfiguriert werden. Daraufhin können die in Abschnitt 7.1 beschriebenen SPs und IDPs verwendet werden, um den eigenen SP bzw. IDP zu testen. Die IDPs haben alle einen Benutzer `testuser` mit dem Password `testuser` und die SPs einen geschützten Bereich unter `https://gntb0*.srv.lrz.de/secure`.

7.4. Vergleich der Anforderungen mit der Erweiterung

In diesem Abschnitt werden die aufgestellten Anforderungen mit der prototypischen Implementierung der Erweiterung verglichen.

7.4.1. Funktionale Anforderungen

F1: Dynamischer Verbindungsaufbau: Es ist durch die Erweiterung möglich, innerhalb von 10 Sekunden, eine Verbindung zwischen SP und IDP aufzubauen. Die fest vorgegebene Wartezeit von 10 Sekunden wird für den Metadaten- und Konvertierungsregelaustausch sowie deren Installation verwendet. In der Praxis könnte diese Wartezeit deutlich reduziert werden, da der Vorgang normalerweise innerhalb 1 Sekunde abgeschlossen ist. Die Anforderung wird daher erfüllt.

F2: Trusted Third Party: Die Erweiterung des Centralized Discovery Services ermöglicht es diesem als zentrle Vermittlungsinstanz in dem TTP-Netzwerk aufzutreten. Daher ist diese Anforderung erfüllt.

F3: TTP-Anmeldung: Die Implementierung erlaubt zwar das Registrieren von neuen Accounts und deren Verwendung, zusätzliche Funktionen wie das Modifizieren oder Löschen von Accounts sind nicht Implementiert. Daher kann diese Anforderung nicht erfüllt werden.

F4: Zuständigkeit: Für die TTP ist ein domainbasiertes System implementiert, das es einem Administrator ermöglicht zu beweisen, dass er Kontrolle über eine bestimmte Domain hat. Diese Anforderung wird daher erfüllt.

F5: Zugriffsberechtigungen: Die TTP-Implementierung erlaubt nur Benutzern, die zuvor beweisen konnten, dass sie Kontrolle über einen Provider haben, diesen zu modifizieren. Daher wird diese Anforderung erfüllt.

F6: Dienstverwaltung: Registrierte Benutzer, die nachgewiesen haben, dass sie für einen Provider zuständig sind, können diesen modifizieren und auch löschen. Eine Weitergabe von Berechtigungen an einen Nachfolger ist nicht implementiert, der Nachfolger kann aber einfach selber beweisen, dass er für einen Provider zuständig ist. Daher wird diese Anforderung erfüllt.

F7: Kompatibilität: Die Erweiterung ist so implementiert, dass sie mit verschiedenen aktuellen Versionen von Shibboleth kompatibel ist. Zudem können sowohl die Erweiterung als auch Shibboleth unabhänig voneinander aktualisiert werden. Damit wird diese Anforderung erfüllt.

F8: Fehlertoleranz: Die Implementierung ist nicht in allen Fällen fehlertolerant und liefert dem Benutzer nicht unbedingt sinnvolle Informationen, um einen Fehler zu beheben. Dadurch wird diese Anforderung nicht erfüllt.

F9: Metadatensynchronisation: Die Aufgabe der Metadatensynchronistaion wird durch die Erweiterung untersützt, weshalb diese Anforderung erfüllt wird.

F10: Benutzerinitiierter Verbindungsaufbau: Durch die Modifizierungen an Embedded und Centralized Discovery Service, können Benutzer den Verbindungsaufbau zwischen zwei Providern anstoßen. Dadurch wird diese Anforderung erfüllt.

F11: IDP Auswahl: Die Erweiterungen des EDS und CDS erlauben es den Benutzern zuerst lokal bei dem SP nach dem gewünschten IDP zu suchen und bei Bedarf die Anfrage an den CDS weiterzuleiten. Damit wird diese Anforderung erfüllt.

F12: Attributsidentifikation: Da das System auf Shibboleth aufbaut, sind alle Möglichkeiten der Attributsidentifiation von Shibboleth weiterhin vorhanden. Dies erfüllt die Anforderung.

F13: Konvertierungsregelverwaltung: Es gibt Schnittstellen zum Suchen, Anzeigen und Erstellen von Konvertierungsregeln. Das Modifizieren von Konvertierungsregeln wird dadurch erreicht, dass eine neue Version der Regel erstellt wird. Die Anforderung kann daher erfüllt werden.

F14: Konvertierungsregelbeschreibung: Die Konvertierungsregeln werden in der Implementierung mit Metadaten versehen, die in einer Datenbank gespeichert werden. Damit wird diese Anforderung erfüllt.

F15: Konvertierungsregelbewertung: Die Implementierung von Bewertungsschnittstellen für die Konvertierungsregeln wurde auf eine spätere Version der Implementierung verschoben, da diese für den Prototypen nicht essentiell war. Die Anforderung wird daher nicht erfüllt.

F16: Konvertierungsregeloperationen: Für die Konvertierungsregeln werden die von Shibboleth verwendeten Attributs-Definitionen verwendet, die bereits alle nötigen Konvertierungsoperationen implementieren. Daher wird diese Anforderung erfüllt.

F17: Konvertierungsregelabgleich: Die Erweiterung implementiert Funktionen, die es dem IDP ermöglichen festzustellen, ob er alle erforderlichen Attribute erstellen kann und gegebenenfalls für die fehlenden Attribute Konvertierungsregeln bei der TTP abzufragen. Dies erfüllt die Anforderung.

F18: Fehlerbenachrichtigung: Der Prototyp implementiert keine Fehlerbenachrichtigung, weshalb diese Anforderung nicht erfüllt wird.

F19: Attribute Release Policies: Aufbauend auf den in Shibboleth für den IDP vorhandenen ARP-Funktionen, werden durch Konvertierungsregeln freigegebene Attribute nur auf den anfragenden SP eingeschränkt. Zusätzlich kann über die Datenbank festgelegt werden, welche Attribute in welcher SP-IDP-Paarung durch Konvertierungsregeln erstellt werden dürfen. Dadurch wird diese Anforderung erfüllt.

F20: Access Control Lists: Ähnlich zu den ARPs können durch eine Blacklist und eine Whitelist ACLs erstellt werden, die bestimmte Verbindungen zwischen SP und IDP blockieren bzw. ermöglichen. Dies erfüllt diese Anforderung.

F21: Attributsfreigabe: Die Attributsfreigabe durch den Benutzer wird durch die Erweiterung nicht eingeschränkt bzw. ermöglicht. Hierfür muss wie bei dem normalen Shibboleth eine Erweiterung wie uApprove verwendet werden. Die Anforderung wird daher nicht erfüllt.

F22: Auditierbarkeit: Die Erweiterung spezifiziert keine explizite Schnittstelle zur Überprüfung von Attributsfreigaben oder Authentifizierungen. Für den Prototypen sind hier die zusammen mit anderen Programmteilen generierten Logausgaben ausreichend. Die Anforderung wird dadurch jedoch nicht erfüllt.

F23: Testsysteme: Für den Protoypen gibt es eine frei zugängliche Testumgebung, in der Interessierte ihre eigenen Systeme testen können. Diese Anforderung wird daher erfüllt.

F24: Kompatibilitätstests: Es gibt in der Erweiterung keinen Kompatiblitätstest, der vor einem echten Verbindungsversuch feststellen kann, ob diese Verbindung erfolgreich sein wird. Daher ist diese Anforderung nicht erfüllt.

F25: Schemaerweiterungen: Die Eigenschaft Shibboleths, beliebige Schemaerweiterungen definieren zu können, wird durch die Erweiterung nicht eingeschränkt. Daher bleibt diese Anforderung erfüllt.

7.4.2. Nichtfunktionale Anforderungen

NF1: Wartbarkeit: Da die Erweiterung ein von der Shibboleth Code-Basis getrennt ist, können beide unabhänig von einander weiterentwickelt und aktualisiert werden. Damit ist diese Anforderung erfüllt.

NF2: Kosten: Die Erweiterung ist als Open-Source ohne weitere Lizenzkosten verfügbar, wodurch diese Anforderung erfüllt wird.

NF3: Zukunftssicherheit: Durch die Unabhänigkeit von Shibboleth selbst, wird die Erweiterung auch mit neuen Versionen von Shibboleth kompatibel sein. Eingeschränkt wird dies durch die Entwicklung von Version 3 des IDP, bei dem nicht sichergestellt ist, dass für die Version 2 entwickelte Erweiterungen ohne Portierung funktionieren. Bei solchen Neuentwicklungen kann in den meisten Fällen nicht mit einer Rückwärtskompatiblität gerechnet werden, weshalb die Anforderung trotzdem erfüllt wird.

Anforderung	Erfüllt		Relevanz
	ja	nein	
F1: Dynamischer Verbindungsaufbau	✓		1
F2: Trusted Third Party	✓		1
F3: TTP-Anmeldung		✓	1
F4: Zuständigkeit	✓		1
F5: Zugriffsberechtigungen	✓		1
F6: Dienstverwaltung	✓		1
F7: Kompatibilität	✓		2
F8: Fehlertoleranz		✓	2
F9: Metadatensynchronisation	✓		1
F10: Benutzerinitiierter Verbindungsaufbau	✓		1
F11: IDP Auswahl	✓		1
F12: Attributsidentifikation	✓		1
F13: Konv.-Regelverwaltung	✓		1
F14: Konv.-Regelbeschreibung	✓		1
F15: Konv.-Regelbewertung		✓	2
F16: Konv.-Regeloperationen	✓		1
F17: Konv.-Regelabgleich	✓		1
F18: Fehlerbenachrichtigungen		✓	3
F19: Attribute Release Policies:	✓		2
F20: Access Control Lists	✓		2
F21: Attributsfreigabe		✓	2
F22: Auditierbarkeit		✓	3
F23: Testsysteme	✓		3
F24: Kompatibilitätstests		✓	3
F25: Schemaerweiterungen	✓		2

Tabelle 7.7.: Vergleich der funktionalen Anforderungen mit der TTP-Implementierung

NF4: Open Source: Die Erweiterung wird unter eine Open-Source-Lizenz veröffentlich, wodurch diese Anforderung erfüllt wird.

Die weiteren nichtfunktionalen Anforderungen sind, wie in Abschnitt 5.5.2 beschrieben schon durch das Konzept erfüllt. Die Implementierung der Erweiterung ändert hieran nichts. Die aus dem Konzept übernommenen Anforderungen sind mit einem „*“ gekennzeichnet.

Anforderung	**Erfüllt**		
	ja	nein	**Relevanz**
NF1: Wartbarkeit	✓		2
NF2: Kosten	✓		1
NF3: Zukunftssicherheit	✓		2
NF4: Open Source	✓		1
NF5: Anwendbarkeit	*		2
NF6: Protokoll	*		1
NF7: Verlässlichkeit der SPs	–	–	1
NF8: Auth.-Verantwortung	*		1
NF9: Datenaktualität	–	–	2
NF10: User-Experience-Konsistenz	*		3
NF11: Zugang zu einer TTP	*		1

Tabelle 7.8.: Vergleich der nichtfunktionalen Anforderungen mit dem TTP-Implementierung

8. Zusammenfassung und Ausblick

In dieser Arbeit wurden die unterschiedlichen Föderationskonzepte von Projekten, Communities, geographisch strukturierten Föderationen und Inter-Föderationen betrachtet. Ziel dabei war es, herauszufinden, welche Anforderungen die einzelnen Konzepte haben und diese einem in dieser Arbeit entwickelten, neuen Konzept für dynamische virtuelle Föderationen gegenüber zu stellen. Dynamische virtuelle Föderationen stellen im Gegensatz zu geographisch strukturierten Föderationen bedarfsorientierte virtuelle Föderationen. Diese Föderationen können, durch den Benutzer angestoßen, innerhalb kürzester Zeit erstellt werden.

Für diese Betrachtung wurden in Kapitel 2 zunächst die Grundlagen des Föderierten Identity Managements sowohl aus organisatorischer als auch technischer Sicht analysiert. In Kapitel 3 wurde eine Analyse des neuen dynamischen virtuellen sowie herkömmlicher Föderationskonzepte durchgeführt, deren Ergebnis insgesamt 25 funktionale und 12 nichtfunktionale Anforderungen waren. Diese Anforderungen wurden in Kapitel 4 mit den zwei bekanntesten SAML Implementierungen Shibboleth und simpleSAMLphp sowie einigen speziellen Erweiterungen für bestimmte Teilaspekte verglichen.

Bei diesem Vergleich konnte keine Lösung oder Kombination von Lösungen gefunden werden, die alle Anforderungen zufriedenstellend erfüllen würde. Bei dem Vergleich der Lösungen mit den Anforderungen wurde allerdings festgestellt, dass Shibboleth am besten als Grundlage für eine Erweiterung geeignet wäre. Hauptsächlich ausschlaggebend für diese Entscheidung war das Vorhandensein eines bestehenden Systems, Attribute durch Attributs-Definitionen einfach konvertieren zu können. Dies wurde, im Vergleich zu der durch JANUS oder PEER umgesetzten Metadaten-Verwaltung, als wichtiger eingestuft.

Daher wurde in Kapitel 5 ein eigenes neues Konzept entwickelt, welches aufbauend auf Shibboleth die nötigen Erweiterungen für die Erfüllung der fehlenden Anforderungen umsetzt. Für das dynamische Management der virtuellen Föderationen ist in dem entwickelten Konzept eine Trusted Third Party zuständig. Diese koordiniert den Metadatenaustausch und die Konvertierung von eventuell unterschiedlichen bzw. die Ableitung unbekannter Attribute.

Zur Kommunikation der Provider mit der TTP wurde ein auf HTTP basierendes Protokoll zum Metadaten- und Konvertierungsregel-Austausch entworfen. Mit Hilfe eines neuen Metadata-Providers können die Provider, die so übertragenen Metadaten, mit den bereits in Shibboleth vorhandenen SAML-Funktionen, verwenden. Zum Anwenden der Konvertierungsregeln wurde eine providerunabhängige Regelbeschreibung auf XSLT-Basis entworfen,

die es ermöglicht die notwendigen Änderungen an den XML-Konfigurationsdateien von Shibboleth vorzunehmen.

Zur Demonstration dieses Konzeptes wurde dieses als Prototyp implementiert. Die in C/C++ und Java geschriebene Implementierung wurde in Kapitel 6 näher beschrieben. Die Implementierung wurde dabei als Erweiterung für Shibboleth implementiert, sodass sie die hierfür vorgesehenen Schnittstellen von Shibboelth verwenden kann und nicht direkt den Quellcode von Shibboleth modifiziert. Dadurch ist eine unabhängige Entwicklung von Shibboleth und den Erweiterungen möglich.

Der Prototyp wurde daraufhin in einer aus acht virtuellen Maschinen bestehenden Testumgebung analysiert. Dabei wurden ein in Kapitel 7 beschriebenes Szenario mit zwei Föderationen und zwei externen Providern entwickelt, in dem gezeigt werden konnte, dass der Einsatz einer TTP einen Vorteil gegenüber herkömmlichen Lösungen bietet.

8.1. Diskussion

Das Konzept dynamischer virtueller Föderationen konnte erfolgreich für Shibboleth implementiert und in einer Testumgebung analysiert werden. Dabei konnte in Kapitel 7 gezeigt werden, dass das Konzept eine Reduzierung der zu installierenden Metadaten bewirkt. Dies ist ein Vorteil, da der Administrator eines Dienstes für das Hinzufügen neuer Verbindungen zu anderen Diensten weder den potentiell sehr großen Metadatensatz einer Föderation noch für jeden verbundenen Dienst dessen Metadaten einzeln installieren muss.

In der mit effektiv sieben Providern aufgesetzten Testumgebung sind die Unterschiede durch die kleine Anzahl an Testsystemen relativ gering. Wie in Tabelle 3.1 in Abschnitt 3.2.2.2 allerdings gezeigt wurde, können in realen nationalen Föderationen durchaus tausende Provider registriert sein. Daher ist die Anwendbarkeit des Konzeptes in größeren realen Umgebungen von Interesse, bei denen es einen nennenswerten Unterschied macht, ob ein Dienst nur mit wenigen oder potentiell hunderten Providern verwendbar ist. Von den vielen Providern kann der für den Dienst zuständige Administrator vermutlich nicht alle genau kennen, wodurch er nicht genau weiß, wer Zugang zu seinem Dienst hat. In der Erweiterung mit dynamischen virtuellen Föderationen ist die Situation auf den ersten Blick noch komplizierter. Es soll jedem Provider möglich sein, seinen SP oder IDP zu registrieren, wodurch die Menge an potentiellen Verbindungen noch größer und flexibler wird. Davon können jedoch Dienste profitieren, die keine hohen Anforderungen an die Qualität der vom IDP übermittelten Informationen haben. SPs oder IDPs, die mehr Kontrolle haben möchten, können über ACLs und ARPs mit der TTP die für ihren SP bzw. IDP möglichen Verbindungen individuell anpassen.

In beiden Fällen ist der automatisierte Austausch von Metadaten und Konvertierungsregeln ein Vorteil, der einem Administrator die Konfiguration erleichtert und den Benutzern flexibleren Zugang zu Diensten ermöglicht.

Damit das Konzept funktioniert, ist es notwendig, dass eine möglichst große Menge an SPs und IDPs an dem Konzept interessiert sind. Dies wäre am einfachsten dadurch zu erreichen, dass ein im Umfeld von nationalen und Inter-Föderationen bekannter Betreiber dieses Konzept verwendet. Ähnlich wie bei sozialen Netzwerken wie Facebook oder Google+, bei denen die Nutzer in dem Netzwerk aktiv sind, in dem auch ihre Freunde aktiv sind, macht es für

eine Organisation nur dann Sinn, dem TTP-Netzwerk beizutreten, wenn dort auch für sie relevante Kooperationspartner vertreten sind.

8.2. Ausblick

Das in dieser Arbeit entwickelte Konzept und dessen Implementierung könnten in zukünfigen Arbeiten weiterentwickelt werden. Dazu gehören vor allem die in dem aktuellen Konzept bzw. dessen Prototypen nicht berücksichtigten Anforderungen. Dazu gehört das von Anforderung F3 TTP-Anmeldung geforderte vollständige Benutzerverwaltungssystem. Diese als „grundlegend“ eingestufte Anforderung ist weitestgehend umgesetzt und unterstützt das Registrieren von neuen Benutzern sowie deren Authentifizierung, es fehlen insbesondere aber noch Funktionen für die Modifikation, sowie das Löschen von bestehenden Accounts. Diese Aspekte der Anforderung sind für den Prototypen als nicht essentiell wichtig eingestuft und können in einer späteren Version hinzugefügt werden. Genauso wurde die Möglichkeit Konvertierungsregeln bewerten zu können, wie es in der „erweiterten“ Anforderung F15 Konvertierungsregel-Bewertung gefordert wurde, auf eine spätere Version verschoben.

Des Weiteren sollten die Funktionen zu den Anforderungen F8: Fehlertoleranz und F18: Fehlerbenachrichtigungen erweitert werden, damit sowohl Benutzer als auch Administratoren bei auftretenden Fehlern entsprechend informiert werden.

Auch die Anforderung F22 Auditierbarkeit spielt für einen Produktiveinsatz eine wichtige Rolle und sollte daher behandelt werden. Weniger wichtig ist die Umsetzung von Anforderung F24 Kompatibilitätstests, da diese, aufgrund der kurzen Dauer eines echten Verbindungsversuches, kaum zusätzlichen Nutzen bringen würde.

Über die Anforderungen hinaus wäre ein besonders interessantes Thema die automatische Umwandlung von Shibboleth XML-Attributs-Definitionen in XSLT-Dokumente, die als Konvertierungsregel verwendet werden können. Ein solches System würde die Konvertierungsregeln noch nützlicher machen, da vorhandene Konvertierungen ohne Kenntnis von XSLT einfach übernommen werden könnten.

Ebenfalls eine Verbesserung des Benutzer-Interfaces wäre eine Überarbeitung oder ähnliche Neuentwicklung von SAMLMetaJS. Über Dialoge in einem Webfrontend bestimmte Eigenschaften, wie zum Beispiel lokalisierte Namen und Beschreibungen für einen Dienst hinzufügen zu können, ist dem direkten Editieren des XML-Textes der Metadaten überlegen. Zusätzlich können Validierungen und Bedingungen eingebaut werden, sodass zum Beispiel jeder Dienst mindestens einen Namen, eine Beschreibung und eine Kontaktperson angeben muss.

Für Administratoren, die keine Weboberfläche benutzen möchten und stattdessen einige Funktionen aus der Kommandozeile oder durch Skripte steuern wollen, wäre eine API denkbar, die das Verwalten von Providern, Metadaten und Konvertierungsregeln erlaubt.

Neben diesen hauptsächlich die Interaktion mit der TTP betreffenden Weiterentwicklungen wäre auch eine Abwandlung des Metadatensynchronisations-Ablaufs denkbar. Hierbei würde zunächst nur eine „deaktivierte“ Verbindung hergestellt und die Administratoren der beteiligten Dienste darüber informiert werden. Die Administratoren können die Verbindung

dann bestätigen oder ablehnen, wodurch sie ohne großen zusätzlichen Aufwand noch mehr Kontrolle über die mit ihrem Provider verwendeten anderen Provider bekommen. Idealerweise würden die Administratoren über eine E-Mail mit detaillierten Informationen über den anderen Provider benachrichtigt und könnten die Verbindung dann mit einem Klick im Webfrontend erlauben. Der Benutzer könnte ebenfalls eine E-Mail-Adresse angeben und über die Entscheidung der Administratoren informiert werden. Dadurch gäbe es dann einen standardisierten Weg, die Administratoren der Provider über einen Verbindungswunsch zu informieren, der potentiell sehr schnell beantwortet werden könnte.

Abbildungsverzeichnis

Literaturverzeichnis

[Bee14] Kristina Beer. *BSI: Mehrere Millionen Internet-Konten durch Botnetze geknackt*. Jan. 2014. URL: http://heise.de/-2090167.

[Bre+11] Michael Brenner et al. *Praxisbuch ISO/IEC 27001: Management der Informationssicherheit und Vorbereitung auf die Zertifizierung*. Hanser, Carl, 2011.

[Bro+13] Daan Broeder et al. *Federated identity management for research collaborations*. Tech. rep. Aug. 2013. URL: http://cds.cern.ch/record/1442597/files/CERN-OPEN-2012-006.pdf.

[BSIKV14] Bundesamt für Sicherheit in der Informationstechnik (BSI). *Kryptographische Verfahren: Empfehlungen und Schlüssellängen*. Technische Richtlinie BSI TR-02102-1. Feb. 2014. URL: https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR02102/BSI-TR-02102_pdf.pdf?__blob=publicationFile.

[Chr13] Jacob Christiansen. "janus-ssp – Metadata Registry for SimpleSAMLphp – Google Project Hosting". In: (2013). URL: https://code.google.com/p/janus-ssp.

[CLARINAbout] CLARIN ERIC. *About | CLARIN ERIC*. Aug. 2014. URL: http://clarin.eu/content/about.

[CLARINSPF] CLARIN ERIC. *Service Provider Federation*. URL: https://www.clarin.eu/content/service-provider-federation.

[CS12] Scott Cantor and Thomas Scavo. *SAMLIdPProxy – GridShib – Internet2 Wiki*. Jan. 2012. URL: https://spaces.internet2.edu/display/GS/SAMLIdPProxy.

[Deu+08] Jörg Deutschmann et al. *DFN-AAI – Technische und organisatorische Voraussetzungen – Attribute für den Bereich E-Learning*. Nov. 2008.

[Eik14] Ronald Eikenberg. *Flash-Update verhindert Cookie-Klau*. July 2014. URL: http://heise.de/-2252252.

[Gie+08] Peter Gietz et al. *Technische und organisatorische Voraussetzungen - Attribute für alle Anwendungen*. Nov. 2008. URL: https://www.aai.dfn.de/fileadmin/documents/attributes/200811/DFN-AAI-Attribute-V.1.0.pdf.

[Hae14] Lukas Haemmerle. *eduGAIN:FAQ - eduGAIN Wiki*. Jan. 2014. URL: https://wiki.edugain.org/eduGAIN:FAQ.

[Hä14] Lukas Hämmerle. *eduGAIN – Updates since October 2013*. Mar. 2014. URL: https://www.clarin.eu/sites/default/files/eduGAIN-at-DASISH-20140310.pdf.

[Hö14] Rainer Hörbe. *SAML V2.0 Metadata Guide*. Jan. 2014. URL: https://www.oasis-open.org/committees/download.php/51890/SAMLMDsimplifiedoverview.pdf.

[IDCInter] *OC5:OpenID Connect Interop 5 - OSIS Open Source Identity Systems*. June 2014. URL: http://osis.idcommons.net/wiki/OC5:OpenID_Connect_Interop_5.

[ISO/IEC29115] ISO/IEC. "Entity authentication assurance framework (ISO/IEC 29115)". In: (Oct. 2011).

[JA+13] Leif Johansson, Pål Axelsson, et al. *Attribute Profile - SWAMID - NORDUnet Portal*. Aug. 2013. URL: https://portal.nordu.net/display/SWAMID/Attribute+Profile.

[JWE] M. Jones, E. Rescorla, and J. JoeHildebrand. *JSON Web Encryption (JWE)*. Internet-Draft draft-ietf-jose-json-web-encryption-31. Work in progress. Internet Engineering Task Force, July 2014. URL: http://tools.ietf.org/html/draft-ietf-jose-json-web-encryption-31.

[JWS] M. Jones, J. Bradley, and N. Sakimura. *JSON Web Signature (JWS)*. Internet-Draft draft-ietf-jose-json-web-signature-31. Work in progress. Internet Engineering Task Force, July 2014. URL: http://tools.ietf.org/html/draft-ietf-jose-json-web-signature-31.

[KW13] Nate Klingenstein and Rod Widdowson. *Define and Release a New Attribute in an IdP*. Shibboleth, Mar. 2013. URL: https://wiki.shibboleth.net/confluence/display/SHIB2/IdPAddAttribute.

[Laa13] Kari Laalo. *CSC Wiki - Joining and registrations*. Oct. 2013. URL: https://confluence.csc.fi/display/HAKA/Joining+and+registrations.

[Lie14] Lucas van Lierop. "GitHub janus-ssp/janus Dokumentation". In: (July 2014). URL: https://github.com/janus-ssp/janus/blob/develop/docs/index.md.

[Lin+13] Mikael Linden et al. "eduGAIN Policy Framework Policy Declaration". In: (Jan. 2013). URL: http://www.geant.net/service/eduGAIN/resources/Documents/GN3-10-327%20eduGAIN_policy_declaration%20v2.0.pdf.

[Lin11] Mikael Linden. *eduGAIN Policy Framework Attribute Profile (RECOMMENDED)*. Apr. 2011. URL: http://www.geant.net/service/eduGAIN/resources/Documents/eduGAIN%20Attribute%20profile.pdf.

[Lod14] Torsten Lodderstedt. *OpenID Connect: Login mit OAuth*. June 2014. URL: http://heise.de/-2218446.

[LS+12] Thomas Lenggenhager, Patrik Schnellmann, et al. *AAI – Authentication and Authorization Infrastructure – Attribute Specification*. SWITCH, Nov. 2012. URL: https://www.switch.ch/aai/docs/AAI_Attr_Specs.pdf.

[Mah07] Michael Mahemoff. *OAuth-OpenID: You're barking up the wrong tree if you think they're the same thing*. Nov. 2007. URL: http://softwareas.com/oauth-openid-youre-barking-up-the-wrong-tree-if-you-think-theyre-the-same-thing.

[Mas11] Javi Masa, ed. *SCHAC Attribute Definitions for Individual Data*. TERENA, July 2011. URL: http://www.terena.org/activities/tf-emc2/docs/schac/schac-schema-IAD-1.4.1.pdf.

[OAuthPostResp] M. Jones and B. Campbell. *OAuth 2.0 Form Post Response Mode - draft 03*. Feb. 2014. URL: http://openid.net/specs/oauth-v2-form-post-response-mode-1_0.html.

[OAuthRespType] B. de Medeiros et al. *OAuth 2.0 Multiple Response Type Encoding Practices*. Feb. 2014. URL: http://openid.net/specs/oauth-v2-multiple-response-types-1_0.html.

[OpenIDCore] N. Sakimura et al. *OpenID Connect Core 1.0*. Feb. 2014. URL: http://openid.net/specs/openid-connect-core-1_0.html.

[OpenIDDisc] N. Sakimura et al. *OpenID Connect Discovery 1.0*. Feb. 2014. URL: http://openid.net/specs/openid-connect-discovery-1_0.html.

[OpenIDDyn] N. Sakimura, J. Bradley, and M. Jones. *OpenID Connect Dynamic Client Registration 1.0*. Feb. 2014. URL: http://openid.net/specs/openid-connect-registration-1_0.html.

[OpenIDSession] B. de Medeiros et al. *OpenID Connect Session Management 1.0 - draft 19*. Feb. 2014. URL: http://openid.net/specs/openid-connect-session-1_0.html.

[OpenIDSpec] OpenID Foundation. *OpenID – Specifications*. July 2014. URL: http://openid.net/developers/specs/.

[Plu10] Jeroen Pluimers. *Maximum URL lengths*. Apr. 2010. URL: http://wiert.me/2010/04/20/maximum-url-lengths/.

[Rei08] Helmut Reiser. "Ein Framework für föderiertes Sicherheitsmanagement". In: *Habilitation. LMU München, München* (2008).

[RFC2616] R Fielding et al. *Hypertext Transfer Protocol – HTTP/1.1*. RFC. June 1999. URL: http://www.ietf.org/rfc/rfc2616.txt.

[RFC3339] G Klyne and C Newman. *RFC 3339: Date and Time on the Internet: Timestamps*. RFC. July 2002. URL: http://www.ietf.org/rfc/rfc3339.txt.

[RFC4648] Simon Josefsson. *The base16, base32, and base64 data encodings*. RFC. 2006. URL: http://www.ietf.org/rfc/rfc4648.txt.

[RFC6749] Dick Hardt. *The OAuth 2.0 Authorization Framework (RFC6749)*. RFC. Oct. 2012. URL: http://www.ietf.org/rfc/rfc6749.txt.

[RFC6750] M Jones and D Hardt. *The OAuth 2.0 Authorization Framework: Bearer Token Usage*. RFC. Oct. 2012. URL: http://www.ietf.org/rfc/rfc6750.txt.

[RFC7033] Donald Eastlake and Paul Jones. *SHA1*. RFC. Sept. 2001. URL: http://www.ietf.org/rfc/rfc3174.txt.

[RFC7033] Paul Jones et al. *WebFinger*. RFC. Sept. 2013. URL: http://www.ietf.org/rfc/rfc7033.txt.

[RFC7033] Ronald Rivest. *MD5*. RFC. Apr. 1992. URL: http://www.ietf.org/rfc/rfc1321.txt.

[SAMLBind] Eve Maler et al. *Bindings for the OASIS Security Assertion Markup Language (SAML) V2. 0*. OASIS, Mar. 2005. URL: http://docs.oasis-open.org/security/saml/v2.0/saml-bindings-2.0-os.pdf.

[SAMLConf] Prateek Mishra, Rob Philpott, Eve Maler, et al. *Conformance Requirements for the OASIS Security Assertion Markup Language (SAML) V2.0*. Mar. 2005. URL: http://docs.oasis-open.org/security/saml/v2.0/saml-conformance-2.0-os.pdf.

[SAMLCore] Scott Cantor, Internet2 John Kemp, and Nokia Rob Philpott. *Assertions and protocols for the oasis security assertion markup language*. OASIS, Mar. 2005. URL: http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf.

[SAMLIDPDS] Oasis. *Identity Provider Discovery Service Protocol and Profile*. Mar. 2008. URL: http://docs.oasis-open.org/security/saml/Post2.0/sstc-saml-idp-discovery.pdf.

[SAMLMeta] Scott Cantor and Maler Philpott. *Metadata for the OASIS Security Assertion Markup Language (SAML) V2. 0*. Mar. 2005. URL: http://docs.oasis-open.org/security/saml/v2.0/saml-metadata-2.0-os.pdf.

[SAMLProf] John Hughes et al. *Profiles for the OASIS Security Assertion Markup Language (SAML) V2. 0*. OASIS, Mar. 2005. URL: http://docs.oasis-open.org/security/saml/v2.0/saml-profiles-2.0-os.pdf.

[Sch14] Fabian Scherschel. *Noch mehr Herzbluten bei OpenSSL*. June 2014. URL: http://heise.de/-2217286.

[Sol13] Andreas Åkre Solberg. "DiscoJuice – Getting started". In: (2013). URL: http://discojuice.org/getting-started.

[Uyt13] Dieter Van Uytvanck. "Analysis of eduGAIN use for CLARIN purposes". In: (Dec. 2013). URL: http://www.clarin.eu/sites/default/files/CE-2013-0247-analysis-edugain.pdf.

[Wis+05] Thomas Wisniewski et al. *SAML V2.0 Executive Overview*. 2005.

[WL12] Rod Widdowson and Thomas Lenggenhager. *DiscoveryService*. Aug. 2012. URL: https://wiki.shibboleth.net/confluence/display/SHIB2/DiscoveryService.

[XML] Tim Bray et al., eds. *Extensible Markup Language (XML) 1.0 (Fifth Edition)*. W3C, Nov. 2008. URL: http://www.w3.org/TR/xml/.

[XMLSig] Mark Bartel et al. *XML Signature Syntax and Processing (Second Edition)*. W3C, June 2008. URL: http://www.w3.org/TR/xmldsig-core.

[Ope13] OpenID Foundation. *Developer's Guide | Account Chooser*. 2013. URL: http://www.accountchooser.net/developers.

[sim14a] simpleSAMLphp.com. *SimpleSAMLphp Identity Provider QuickStart*. Aug. 2014. URL: https://simplesamlphp.org/docs/stable/simplesamlphp-idp.

[sim14b] simpleSAMLphp.com. *simpleSAMLphp Installation and Configuration*. Aug. 2014. URL: https://simplesamlphp.org/docs/stable/simplesamlphp-install.

[sim14c] simpleSAMLphp.com. *SimpleSAMLphp Service Provider QuickStart*. Aug. 2014. URL: https://simplesamlphp.org/docs/stable/simplesamlphp-sp.

[GEA14] GEANT. *GEANT: About GEANT*. Aug. 2014. URL: http://www.geant.net/About/Pages/home.aspx.

[Goo14a] Google. *OpenID 2.0*. Feb. 2014. URL: https://developers.google.com/accounts/docs/OpenID.

[Goo14b] Google. *Using OAuth 2.0 for Login (OpenID Connect) - Google Accounts Authentication and Authorization — Google Developers*. July 2014. URL: https://developers.google.com/accounts/docs/OAuth2Login.

[Int12] Internet2. *eduPerson Object Class Specification*. Mar. 2012. URL: https://www.internet2.edu/media/medialibrary/2013/09/04/internet2-mace-dir-eduperson-201203.html.

[Mic14] Microsoft. *OpenID Connect*. Apr. 2014. URL: http://msdn.microsoft.com/en-us/library/azure/dn645541.aspx.

[Ope14] OpenID Foundation. *Welcome to OpenID Connect*. Feb. 2014. URL: http://openid.net/connect.

[Shi14] Shibboleth Consortium. *Shibboleth Consortium - Service Provider*. Aug. 2014. URL: https://shibboleth.net/products/service-provider.html.

[Sta13] Statistisches Bundesamt. *Bildung und Kultur – Studierende an Hochschulen – Fächersystematik*. Nov. 2013. URL: https://www.destatis.de/DE/Methoden/Klassifikationen/BildungKultur/StudentenPruefungsstatistik.pdf?__blob=publicationFile.

[UNI14] UNINETT. *norEdu* Object Class Specification*. UNINETT AS, Jan. 2014.

[W3C07] W3C. *SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)*. W3C, Apr. 2007. URL: http://www.w3.org/TR/soap12-part1/.

[Yah14] Yahoo. *Yahoo-OpenID*. 2014. URL: http://openid.yahoo.com.

[edu14a] eduGAIN. *eduGAIN membership status*. Apr. 2014. URL: `http://www.edugain.org/technical/status.php`.

[edu14b] eduGAIN. *eduGAIN technical site – Joining checklist*. Webpage. Aug. 2014. URL: `https://www.edugain.org/technical/joining_checklist.php`.

Anhang

A. SAML Requests und Responses

Listing A: Assertion des LMU-IDP

```
<?xml version="1.0" encoding="UTF-8"?>
<saml2p:Response xmlns:saml2p="urn:oasis:names:tc:SAML:2.0:protocol"
   Destination=
     "https://fsso.springer.com/federation/Consumer/metaAlias/SpringerServiceProvider"
   ID="_6c19f7fc7add698fbe1177280983c1fd"
   InResponseTo="s279d48ff26b5939b0fae9ac22d84fcf8f727c3822"
   IssueInstant="2014-05-30T13:25:29.274Z" Version="2.0">
 <saml2:Issuer xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion"
    Format="urn:oasis:names:tc:SAML:2.0:nameid-format:entity"
    >https://lmuidp.lrz.de/idp/shibboleth</saml2:Issuer>
 <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
  <ds:SignedInfo>
   <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
   <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
   <ds:Reference URI="#_6c19f7fc7add698fbe1177280983c1fd">
    <ds:Transforms>
     <ds:Transform
     Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
     <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
    </ds:Transforms>
    <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
    <ds:DigestValue>kCxp5W61jwpvJR8wBexZwH9Z0qo=</ds:DigestValue>
   </ds:Reference>
  </ds:SignedInfo>
  <ds:SignatureValue>rJyyFdZ/uwrV+JKwH[...]by4cXFFvsj/eQQ==</ds:SignatureValue>
  <ds:KeyInfo>
   <ds:X509Data>
    <ds:X509Certificate>MIIFjDCCBHSgAwIB[...]KNL0+E7LXkVRkH4=</ds:X509Certificate>
   </ds:X509Data>
  </ds:KeyInfo>
 </ds:Signature>
 <saml2p:Status>
  <saml2p:StatusCode Value="urn:oasis:names:tc:SAML:2.0:status:Success"/>
 </saml2p:Status>
 <saml2:EncryptedAssertion xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion">
  <xenc:EncryptedData xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
     Id="_ecc34ddd3f6d36fb7d35e852d397f0cd"
     Type="http://www.w3.org/2001/04/xmlenc#Element">
   <xenc:EncryptionMethod xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
    Algorithm="http://www.w3.org/2001/04/xmlenc#aes128-cbc"/>
   <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
    <xenc:EncryptedKey xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
    Id="_0d920776147cd20a7ad67dda1a7e3b50">
     <xenc:EncryptionMethod xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
    Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p">
      <ds:DigestMethod xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
    Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
     </xenc:EncryptionMethod>
     <ds:KeyInfo>
      <ds:X509Data>
```

```
44          <ds:X509Certificate>MIIF8jCCBNqgAwIB[...]wSXYR5kNE7w5V6M=</ds:X509Certificate>
45        </ds:X509Data>
46       </ds:KeyInfo>
47       <xenc:CipherData xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
48        <xenc:CipherValue>e+oFsk8JDu/F3/XC[...]fJrRP25kEN1YzA==</xenc:CipherValue>
49       </xenc:CipherData>
50      </xenc:EncryptedKey>
51     </ds:KeyInfo>
52     <xenc:CipherData xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
53      <xenc:CipherValue>FOCCUjD/qgGqj5Q4[...]ZRjLX3tu1aLpfV8=</xenc:CipherValue>
54     </xenc:CipherData>
55    </xenc:EncryptedData>
56   </saml2:EncryptedAssertion>
57 </saml2p:Response>
```

B. Installationsanleitung für die Identity Provider Erweiterung

```
This extension adds metadata synchronization for the TTP.

Download and extract the recent version:
-------------------------------------------------------------------------------
~# unzip ttp-sync-0.0.4-src.zip
-------------------------------------------------------------------------------


Change to the directory that contains the source files of the shibboleth IDP
and copy the file 'ttp-sync*.jar' to the 'lib' directory. You might have have
to remove older versions of this library from the 'lib' folder.
-------------------------------------------------------------------------------
~# cd /usr/local/src/shibboleth-identityprovider-2.4.0
/usr/local/src/shibboleth-identityprovider-2.4.0#
    cp ~/ttp-sync-0.0.4/lib/ttp-sync-0.0.4.jar lib/
-------------------------------------------------------------------------------


Edit the 'web.xml' file and include the metadata sync <servlet> and
<servlet-mapping> elements inside the <web-app> element. Edit the URL parameter
and local storage location according to your configuration. Also ensure that
the directory you specify to store the metadata files is readable and writable
by the tomcat servers user.
-------------------------------------------------------------------------------
/usr/local/src/shibboleth-identityprovider-2.4.0#
    vim src/main/webapp/WEB-INF/web.xml
-------------------------------------------------------------------------------
---[web.xml]-----------------------------------------------------------------
<web-app>
    ...
    <!-- TTP Metadata sync handler -->
    <servlet>
        <servlet-name>ttp_metasync</servlet-name>
        <servlet-class>de.lrz.gntb.idpextension.TTPMetadataSyncServlet
</servlet-class>
        <init-param>
            <param-name>initiatorACL</param-name>
            <param-value>gntb08.srv.lrz.de 10.13.37.42</param-value>
        </init-param>
        <init-param>
            <param-name>metadataStorageDirectory</param-name>
            <param-value>/opt/shibboleth-idp/metadata/ttp/</param-value>
        </init-param>
    </servlet>

    <servlet-mapping>
        <servlet-name>ttp_metasync</servlet-name>
        <url-pattern>/DAME</url-pattern>
    </servlet-mapping>
```

```
    ...
</web-app>
--------------------------------------------------------------------------------


Rerun the installation script *without* changing the installation directory
or overwriting the configuration.
--------------------------------------------------------------------------------
/usr/local/src/shibboleth-identityprovider-2.4.0#
    env JAVA_HOME=/usr/lib/jvm/default-java ./install.sh
--------------------------------------------------------------------------------


Change the relying-party configuration to add a new metadata provider within
the probably already existing chaining metadata provider.
--------------------------------------------------------------------------------
/usr/local/src/shibboleth-identityprovider-2.4.0# cd /opt/shibboleth-idp/
/opt/shibboleth-idp# vim conf/relying-party.xml
--------------------------------------------------------------------------------
---[relying-party.xml]-------------------------------------------------------
<rp:RelyingPartyGroup ...
    xmlns:ttp="urn:mace:gntb.lrz.de:ttpextension:ttpmetadataprovider"
    xsi:schemaLocation="...
        urn:mace:gntb.lrz.de:ttpextension:ttpmetadataprovider
        classpath:/schema/ttp-metadata-provider.xsd">
    ...
    <metadata:MetadataProvider id="ShibbolethMetadata"
        xsi:type="metadata:ChainingMetadataProvider">

        ...

        <metadata:MetadataProvider id="TTP" xsi:type="ttp:TTPMetadataProvider"
            location="/opt/shibboleth-idp/metadata/ttp">
        </metadata:MetadataProvider>
    </metadata:MetadataProvider>
    ...
</rp:RelyingPartyGroup>
--------------------------------------------------------------------------------

After finishing the configuration restart the tomcat server to activate the
changes.
--------------------------------------------------------------------------------
/opt/shibboleth-idp# service tomcat6 restart
--------------------------------------------------------------------------------
```

C. Installationsanleitung für die Service Provider Erweiterung

```
This extension adds metadata synchronization for the TTP extension to the
Shibboleth Service Provider to the shibd process and the mod_shib Apache module.

Download and extract the recent version:
-------------------------------------------------------------------------------
~# tar -xzvf ttp_sync-0.0.4.tar.gz
-------------------------------------------------------------------------------

Change to the expanded directory:
-------------------------------------------------------------------------------
~# cd ~/ttp_sync-0.0.4
-------------------------------------------------------------------------------

The build process needs autoconf, g++ and some libraries to build. The
configure script will check if everything is available, the following shows a
list of things that need to be installed on a clean debian system.
-------------------------------------------------------------------------------
# apt-get install dh-autoreconf g++
# apt-get install libssl-dev libboost-dev liblog4shib-dev libxerces-c-dev
# apt-get -t wheezy-backports install libxmltooling-dev libshibsp-dev
# apt-get install xmltooling-schemas
# apt-get -t wheezy-backports install libsaml2-dev opensaml2-schemas \
    opensaml2-tools
# apt-get install apache2-threaded-dev
-------------------------------------------------------------------------------

To start the build process run:
-------------------------------------------------------------------------------
~/ttp_sync-0.0.4# autoreconf -if
~/ttp_sync-0.0.4# ./configure
~/ttp_sync-0.0.4# make
-------------------------------------------------------------------------------

If the build was successful you can install the shared libraries. The default
location for them will be /usr/local/lib/shibboleth
-------------------------------------------------------------------------------
~/ttp_sync-0.0.4# make install
-------------------------------------------------------------------------------

Depending on your distribution the shibboleth service provider configuration
files will be in differen locations. If you compiled it yourself on debian the
default location for them will be /usr/local/etc/shibboleth.

Open the configuration file shibboleth2.xml and add the following <OutOfProcess>
and <InProcess> tags right at the beginning of the file. If you chose to install
the libraries to another folder you will have to change the path.

Within the Sessions element, add an additional SessionInitiator of the type
'TTPSessionInitiator' below your regular SessionInitiator.
```

```
Additionally you need to create a new Handler within the Sessions Element.
Adjust the storage directory and acl to your configuration. WARNING: The ACL can
only contain ip addresses and every host specified with the ACL will be able to
trigger your system to download files from anywhere to the storage directory.

The last configuration change is to add a new MetadataProvider. The storage
location should be the same as the one specified at the corresponding handler.
------------------------------------------------------------------------------
/usr/local/etc/shibboleth# vim shibboleth2.xml
------------------------------------------------------------------------------
---[shibboleth2.xml]-----------------------------------------------------------
<SPConfig ...>

    <OutOfProcess>
        <Extensions>
            <Library path="/usr/local/lib/shibboleth/ttp_sync.so"
                fatal="true" />
        </Extensions>
    </OutOfProcess>
    <InProcess>
        <Extensions>
            <Library path="/usr/local/lib/shibboleth/ttp_sync-lite.so"
                fatal="true" />
        </Extensions>
    </InProcess>

    ...

    <Sessions ...>

        ...

        <SessionInitiator type="Chaining" Location="/Login"
            isDefault="true" id="Login">
            <SessionInitiator type="SAML2"
                template="bindingTemplate.html" />
            <SessionInitiator type="SAMLDS"
                URL="http://gntb03.srv.lrz.de/shibboleth-ds/index.html"/>
        </SessionInitiator>

        <SessionInitiator type="TTPSessionInitiator" Location="/TTP"
            id="TTP" />

        ...

        <!-- TTP Metadata syncronisation. -->
        <Handler type="TTPMetadataSyncHandler" Location="/MetaSync"
            metadataStorageDirectory="/usr/local/etc/shibboleth/metadata/ttp"
            initiatorACL="129.187.163.168"/>
        ..
    </Sessions>
```

```
    ...

    <MetadataProvider type="TTPMetadataProvider"
        metadataStorageDirectory="/usr/local/etc/shibboleth/metadata/ttp"/>

    ...

</SPConfig>
-------------------------------------------------------------------------------

After configuring the extension, make sure the storage location exists and is
writable by the shibd process.
-------------------------------------------------------------------------------
/usr/local/etc/shibboleth# mkdir -p metadata/ttp
-------------------------------------------------------------------------------

The last step is to restart the apache webserver and the shibd process.
-------------------------------------------------------------------------------
/usr/local/etc/shibboleth# service apache2 restart && service shibd restart
-------------------------------------------------------------------------------
```