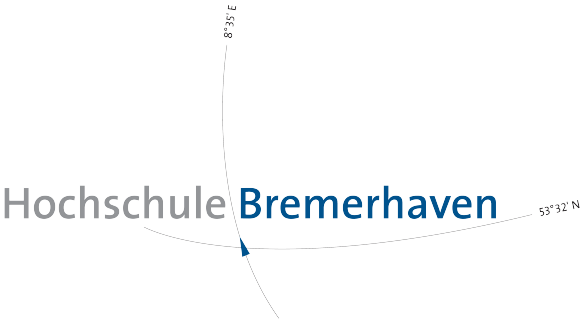

ALFRED-WEGENER-INSTITUT
HELMHOLTZ-ZENTRUM FÜR POLAR- UND MEERESFORSCHUNG

Hochschule Bremerhaven

Fachbereich 2 – Management und Informationssysteme

Einbettung einer lokalen Software eines Föderationsmitgliedes zur Bereitstellung in einem Föderationsumfeld (DFN-AAI)

Bachelorarbeit im Studiengang Informatik in Kooperation mit dem Alfred-Wegener-Institut, Helmholtz-Zentrum für Polar- und Meeresforschung

Autor: Fabian Mangels <fabian@mangels.it>
Matrikel-Nr.: 34295

Ort / Datum der Abgabe: Bremerhaven, den 03.12.2018

Erstgutachter: Prof. Dr.-Ing. Henrik Lipskoch
Zweitgutachter: Prof. Dr.-Ing. Oliver Radfelder Dipl. Inform.
Betrieblicher Betreuer: Siegfried Makedanz

Inhaltsverzeichnis

Danksagung

An dieser Stelle möchte ich mich bei all denen bedanken, die mich während der Anfertigung der Bachelorarbeit unterstützt und motiviert haben.

Danken möchte ich zuallererst meinen Gutachtern der Hochschule Bremerhaven, Herrn Prof. Dr.-Ing. Henrik Lipskoch und Herrn Prof. Dr.-Ing. Oliver Radfelder Dipl. Inform., die meine Abschlussarbeit und somit auch mich betreut haben. Durch die regelmäßig stattgefundenen Kolloquien in einem erweiterten Kreis von Studierenden konnte ein Austausch im wissenschaftlichen Kontext erfolgen. Auch Probleme während der Ausarbeitung konnten hier – als auch persönlich – konstruktiv erörtert werden.

Danken möchte ich auch meinem betrieblichen Betreuer und Arbeitskollegen vom *Alfred-Wegener-Institut*, Siegfried Makedanz. Durch ihn wurde es mir ermöglicht, während meines beinahe gesamten Bachelorstudiums als studentische Hilfskraft im Rechenzentrum des AWI zu arbeiten und letztendlich auch das Thema und den Kontext der Ausarbeitung zu wählen. Ich konnte stets auf Mithilfe bei Problemen als auch auf langjährige Erfahrungen im Themenkomplex Identitätsmanagement zurückgreifen.

Ein ganz besonderer Dank gilt außerdem meinen Eltern, die mich im Studium finanziell unterstützt haben und in der gesamten Zeit immer Vertrauen, Verständnis und Hilfe entgegen gebracht haben. Sehr dankbar bin ich ebenfalls meiner Zwillingsschwester, Sabrina Mangels, die mir bei Zweifeln oder Problemen stets beratend und geduldig zur Seite stand.

Abkürzungsverzeichnis

AAI Authentication and Authorization Infrastructure

API Application Programming Interface

AWI Alfred-Wegener-Institut

CPU Central Processing Unit

DESY Deutsches Elektronen-Synchrotron

DFN Deutsches Forschungsnetz

DIT Directory Information Tree

DKFZ Deutsches Krebsforschungszentrum

DN Distinguished Name

DS Discovery Service, dt. Lokalisierungsdienst

FIM Federated Identity Management, dt. Föderatives Identitätsmanagement

FZJ Forschungszentrum Jülich

GSI GSI Helmholtzzentrum für Schwerionenforschung GmbH

HDF Helmholtz Data Federation, dt. Helmholtz-Datenföderation

HTTP(S) Hypertext Transfer Protocol (Secure)

IdM Identity Management, dt. Identitätsmanagement

IdP Identity Provider, dt. Identitätsanbieter

JSON JavaScript Object Notation

JWT JSON Web Token

KDF Key Derivation Function

KIT Karlsruher Institut für Technologie

LDAP(S) Lightweight Directory Access Protocol (over SSL), dt. Leichtgewichtiges Verzeichniszugriffsprotokoll

LDIF LDAP Data Interchange Format

OASIS Organization for the Advancement of Structured Information Standards

OAuth Open Authorization

OIDC OpenID Connect

OS Operating System

OTP One-time Password

PKI Public Key Infrastructure

RAM Random-Access Memory

REST Representational State Transfer

SAML Security Assertion Markup Language

SASL Simple Authentication and Security Layer

SCP Secure (File) Copy

SHA Secure Hash Algorithm

SLO Single Log-out, dt. Einmalabmeldung

SP Service Provider, dt. Diensteanbieter

SSH Secure Shell

SSL Secure Sockets Layer

SSO Single Sign-on, dt. Einmalanmeldung

TLS Transport Layer Security

VM Virtuelle Maschine

vRA (VMware) vRealize Automation

XML Extensible Markup Language

Abbildungsverzeichnis

Tabellenverzeichnis

Listings-Verzeichnis

Abstract

Thema

Einbettung einer lokalen Software eines Föderationsmitgliedes zur Bereitstellung in einem Föderationsumfeld (DFN-AAI).

Stichworte

Identitätsmanagement, SSO, SAML, Shibboleth, DFN, AAI, Föderation, Föderatives Identitätsmanagement, SP-IdP-Proxy, Unity IdM, LDAP, OpenLDAP

Kurzzusammenfassung

Diese Ausarbeitung beschäftigt sich mit dem vielschichtigen Themenkomplex Identitätsmanagement (IdM) und einem Ansatz, wie Ressourcen bzw. Anwendungen für NutzerInnen in einem Föderationsumfeld bereitgestellt werden können, für die lokale Benutzerkonten notwendig sind. Innerhalb der eigenen Domäne ist diese Bereitstellung ohne weitere Maßnahmen möglich. Grundsätzlich erfolgt der Zugriff auf geteilte Anwendungen von Diensteanbietern (SPs) im Kontext einer Föderation anhand übermittelter Attribute einer zugehörigen Entität. Für domänenfremde NutzerInnen, deren digitale Identität in einem unbekannten IdM verwaltet wird, reicht eine Übermittlung der Attribute für die hier betrachtete Bereitstellung nicht aus, daher wird die Erstellung eines lokalen Benutzerkontos erforderlich. Die Mitgliedschaft der beteiligten Einrichtungen in einer AAI und die Konzepte der Authentifizierung und Autorisierung stellen hier die wichtige Grundlage. Um die benötigten Attribute der Entitäten und die Metadaten der verschiedenen Einrichtungen (Identitätsanbieter, IdPs) auszutauschen, kommt das XML-*Framework* SAML zum Einsatz. Ein sogenannter *SP-IdP-Proxy* agiert in diesem Szenario als Zwischenakteur, der fremde Entitäten authentifiziert und autorisiert sowie ein lokales Benutzerkonto im eigenen dafür vorgesehenen Identitätsspeicher in Ausprägung eines *OpenLDAP*-Verzeichnisdienstes anlegt. Bei Vorhandensein eines zuvor erzeugten Benutzerkontos durch den *SP-IdP-Proxy* (*Unity IdM*) kann anschließend auf die am *Alfred-Wegener-Institut* zur Verfügung gestellte Anwendung (*VMware vRealize Automation*) innerhalb der Föderation zugegriffen werden. Damit dieser Zugriff allerdings funktionieren kann, muss zuvor eine automatisierte Benutzer-Entitätensynchronisation mit einem selbst entwickelten *Bash*-Skript durchgeführt werden. Bei diesem Vorhaben kann der Ansatz des *Single Sign-on* (SSO) nicht verwirklicht werden, für den die Software *Shibboleth* mit SAML in der DFN-AAI ursprünglich eingesetzt wird; die in diesem Szenario notwendige Erzeugung einer lokalen Referenz (Benutzerkonto) zu einer föderativen digitalen Identität bleibt vorhanden.

1 Einleitung

Das 1980 gegründete *Alfred-Wegener-Institut* (AWI) ist eine bedeutende deutsche Forschungseinrichtung in der Polar- und Meeresforschung und hat ihren Hauptsitz in Bremerhaven. Zu den Außenstellen zählen Potsdam, Helgoland und Sylt. In den Anfängen beschäftigte die Stiftung öffentlichen Rechts nur wenige Mitarbeiter; heute sind es mehr als 1000 Beschäftigte in den verschiedensten Disziplinen. Geforscht wird vorwiegend in den Fachbereichen Geo-, Bio- und Klimawissenschaften. Unterstützt werden die Forscher und Forscherinnen durch eine leistungsfähige Infrastruktur, die auch die weit entfernten Stationen in der Arktis und Antarktis sowie Schiffe und Flugzeuge zugänglich macht. Das AWI ist ein international anerkanntes Kompetenzzentrum, welches die deutsche Polarforschung koordiniert, aber auch die Nordsee und ihre deutschen Küstenregionen erforscht [AT17].

Auch das Hauptrechenzentrum befindet sich in der bremischen Seestadt, in dem der praktische Anteil dieser Bachelorarbeit durchgeführt wurde. Mit zunehmender Menge an digitalen Forschungsdaten – viele von diesen haben sogar einen einmaligen Charakter – steigt auch der Bedarf an Rechenleistung und die Absicht Daten in einem internationalen Forschungsumfeld zur Verfügung zu stellen [BJK$^+$13, vgl. S. 2]. Durch die Anbindung am Deutschen Forschungsnetz (DFN) und die hiermit bereitgestellten Dienste bzw. der Infrastruktur (AAI) ist es möglich, sich mit Hochschulen und Forschungseinrichtungen auszutauschen und letztendlich zu kooperieren. Außerdem ist das AWI Mitglied der Helmholtz-Gemeinschaft und arbeitet an der Entwicklung einer international vernetzten Forschungsdateninfrastruktur mit; konkrete Arbeiten finden hier u. a. im Projekt *Helmholtz Data Federation* (HDF) statt.

Die Helmholtz-Datenföderation wird von der Helmholtz-Gemeinschaft finanziert und geht auf Empfehlungen eines Abschlussberichts (FP7, 2007-2013) der Europäischen Kommission ein [Eur15]. Grundsätzlich wird sich in diesem Umfeld mit „einer der großen Herausforderungen des nächsten Jahrzehnts [beschäftigt]: Die Bewältigung der Datenflut in der Wissenschaft, insbesondere aus den großen Forschungsinfrastrukturen der Helmholtz-Zentren“ [HT18]. Insgesamt werden bundesweit die sechs Rechenzentren der Forschungseinrichtungen AWI, DESY, DKFZ, FZJ, GSI und KIT mit Investitionen gefördert, damit der föderale Gedanke von verteilten Daten und Rechenressourcen effizient und sicher gestaltet werden kann [HT18]. Im wissenschaftlichen Kontext wird dieser Ansatz auch als *Federated Identity Management* (FIM) aufgefasst. Es ist eine unter Vertrauen zustande kommende Vereinbarung, die zwischen mehreren Institutionen getroffen werden kann, damit Nutzer dieselben Identifikationsdaten verwenden können, um Zugriff auf gesicherte Ressourcen in diesem Verbund zu erhalten. Vorteile ergeben sich sowohl bei den Organisationen sie benötigen keine zusätzliche Benutzerverwaltung, als auch bei deren Nutzern, die einen höheren Komfort verspüren, da sie sich nur ein zentrales Benutzerkonto mit Passwort merken müssen. Beispielsweise könnten auch unterschiedliche Einrichtungen eine Anwendung gemeinsam nutzen, um Kosten zu sparen, Ressourcen zu konsolidieren und die Zusammenarbeit zu fördern [BJK$^+$13, ABB$^+$18, vgl. S. 1ff, S. 4ff].

Das zu realisierende Szenario im Kontext dieser Bachelorarbeit ist in der Abbildung 1.1 vereinfacht dargestellt. Über die dabei benötigte Komponente soll es NutzerInnen aus Fremdeinrichtungen im Föderationsumfeld ermöglicht werden, einen Zugriff auf sonst nur im AWI intern bereitgestellte Dienste (Daten, Rechenressourcen) zu erlangen (2.). Dies ist gleichzeitig eine Konzeptprüfung für gleichwertige Szenarien bei anderen Föderationsmitgliedern, für die MitarbeiterInnen des

AWIs dementsprechend Fremdnutzer wären. Das Teilen von Diensten mit domänenfremden Nutzern in einem größeren Umfeld erfordert zuvor die Authentifizierung und Autorisierung der Identität (1.). Dies wirft die Frage auf, wo die Nutzer legitimiert werden sollen. Das *AARC-Projekt* (*Authentication and Authorisation for Research and Collaboration*) hat dabei gezeigt, dass Forschungsgemeinschaften vermehrt einen *SP-IdP-Proxy* verwenden, um in einem FIM die Autorisierung des einzelnen Nutzers für einen föderativen Dienst zu erreichen; ein entsprechender Proxy wird Bestandteil der zu erstellenden Komponente sein. Als Referenz für den Aufbau einer solchen Zugriffsverwaltung in einer Föderation gilt die aus der AARC-Initiative entstandene *Blueprint-Architektur* (BPA, `https://aarc-project.eu/architecture/`) [ABB+18, SFH+18, KLDS15, vgl. S. 5f.].

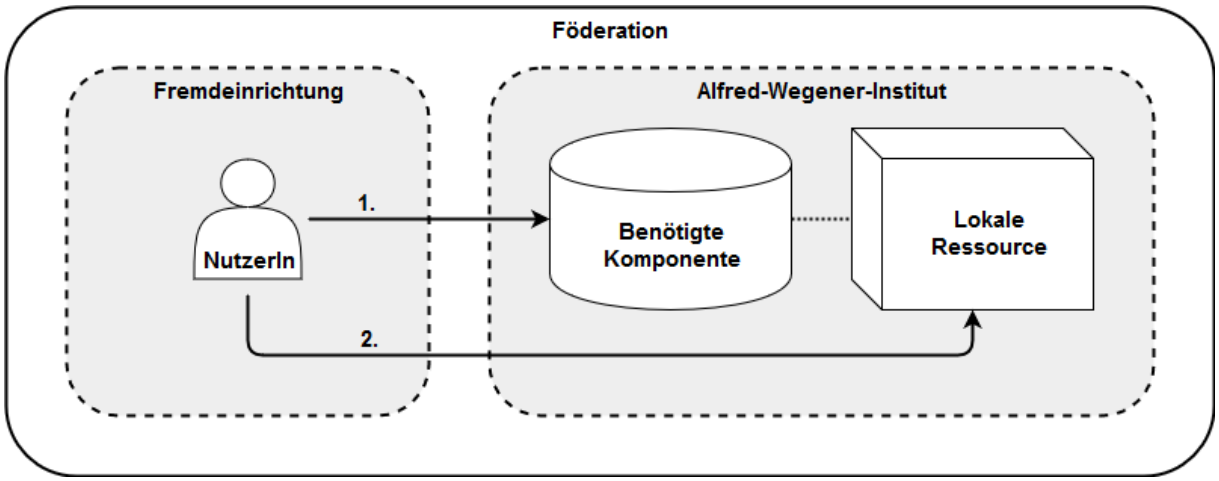

Abbildung 1.1: Vereinfachter Überblick des Szenarios

Als Einstieg in diese Arbeit wird Kenntnis über den aktuellen Stand der Möglichkeiten des föderativen Austausches von Identitätsdaten vermittelt. In diesem Kontext werden im Kapitel 2 und 3 die Grundlagen des Identitätsmanagements, der Authentifizierung / Autorisierung, als auch die unter *Single Sign-on* verwendeten Protokolle und Verfahren beleuchtet (s. Abschnitt 3.1). Anschließend wird die zum Einsatz kommende Software (*Shibboleth*, *Unity IdM*, *OpenLDAP*, *VMware vRealize Automation*) betrachtet und deren Verwendung aufgezeigt. Zuvor wird die als Grundlage fungierende Infrastruktur (DFN-AAI) analysiert und beschrieben. Das nächste Kapitel 4 stellt den praktischen Anteil dieser Ausarbeitung bereit. Es werden also hier die Vorarbeiten und die Implementierung des Vorhabens beschrieben. Anhand von Aktivitätsdiagrammen wird das mögliche Nutzer- und Anwendungsverhalten der benötigten Komponente anschaulich dargestellt. Im letzten Kapitel 5 wird die Ausarbeitung noch einmal kritisch begutachtet und abschließend zusammengefasst. Weitere Maßnahmen, die projektübergreifend und zukünftig Anwendung finden können, werden hier auch thematisiert.

Durch die studienbegleitende Tätigkeit im Rechenzentrum des *Alfred-Wegener-Instituts* ergab sich die Möglichkeit für den Autor im HDF-Projekt an der Entwicklung der international vernetzten Forschungsdateninfrastruktur mitzuarbeiten. Die Erkenntnisse aus diesem Projekt sollen dem Austausch mit anderen Einrichtungen und dem Identitätsmanagement des AWIs zugutekommen.

2 Grundlagen

In diesem Kapitel werden die grundlegenden Begriffe im Umfeld des Identitätsmanagements erläutert, die für die nachfolgenden Kapitel zum Verständnis benötigt werden. Betrachtet wird hier das IdM mit seinen Hauptkomponenten bezogen auf ein Unternehmens- bzw. Organisationsumfeld (s. Abschnitt 2.1). Abschließend wird auf den föderativen Ansatz (Identitätsverteilung, s. Abschnitt 2.2) eingegangen und welche Möglichkeiten dieser bietet.

2.1 Identitätsmanagement

In dieser Bachelorarbeit geht es hauptsächlich um digitale Identitäten von realen Personen und wie damit umgegangen wird. Eine passende Definition des IdM lautet: „Identitätsmanagement liefert die notwendigen Grundlagen zu jedweder Form von personalisiertem und berechtigtem Zugriff auf schützenswerte Ressourcen, Dienste und Systeme und bildet somit einen elementaren Baustein des IT-Sicherheitsmanagements“ [JD08, S. 225]. Demnach erfüllt das IdM mehr als nur die Verwaltung von digitalen Identitäten, sondern ist auch für den sicherheitsrelevanten Kontext in einer Organisation wichtig. Alle möglichen Systeme und Geschäftsprozesse benötigen in der heutigen Unternehmenslandschaft die Unterstützung des Identitätsmanagements, was oftmals zu sehr komplexen Identitätsmanagement-Prozessen führen kann. Beim Umsetzen des IdM – also wie mit digitalen Identitäten verfahren werden soll – ist es von großem Vorteil, sich von Anfang an mit den organisatorischen Abläufen der zu betrachtenden Institutionen zu beschäftigen und sie auch zu verstehen. Dabei reicht es nicht aus, die Prozesse initial zu erfassen, sondern es müssen auch kontinuierliche Erweiterungen oder Veränderungen verfolgt werden [JD08, vgl. S. 225f.].

Um einen konkreteren Einblick zu übermitteln, welche Möglichkeiten und Handlungsvorkehrungen das Identitätsmanagement umfasst, werden in den nächsten zwei Absätzen die Herausforderungen und Ziele der gegenüberliegenden Blickwinkel, nämlich einmal aus der Betreibersicht und anschließend aus der Nutzersicht, aufgezeigt.

Aus der Betreibersicht ist es unbedingt notwendig, dass ein ausgefeiltes IdM zur Verfügung steht. Letztendlich brauchen nahezu alle Geschäftsprozesse diese Unterstützung – die Schlagworte Struktur und Abrechnung sind hier zu nennen – und auch im rechtlichen Sinne kann hierauf nicht verzichtet werden. Insgesamt wird nachstehend auf sieben entscheidende Aspekte eingegangen [JD08, vgl. S. 227ff]:

- Verwaltung von Nutzerkonten –
  Die Nutzerkonten müssen konsistent und fehlerfrei auf allen beteiligten Systemen des IdM verwaltet werden können. Die Verwaltung sieht das automatische Hinzufügen, die Modifizierung und das Entfernen eines Kontos vor [JD08, vgl. S. 227].

- Aktualität von Zugriffsberechtigungen –
  Der Zugriff auf alle Ressourcen darf bekanntlich nicht von jeder Person erfolgen. Es dürfen folglich nur berechtigte Personen auf schützenswerte Daten einen Zugang haben (Autorisierung, s. Unterabschnitt 2.1.2). Das IdM muss daher eine passende und einheitliche Rechtevergabe und die immer auf einem aktuellen Stand befindlichen Zugriffsberechtigungen

sicherstellen. Zugriffsrechte können vergeben, modifiziert oder andernfalls entzogen werden [JD08, vgl. S. 227].

- Bündnis –
  Durch das Anlegen einer digitalen Identität der Nutzer und einem dafür vorgesehenen Identitätsmanagement können Bündnisse mit verschiedenen Einrichtungen eingegangen werden, um beispielsweise einen einfachen Zugang zu spezifischen Anwendungen bei Bündnispartner durch *Single Sign-on* (SSO) zu ermöglichen. Ein konkretes Beispiel ist hier die *DFN-AAI-Föderation*. In einer solchen Föderation werden das notwendige Vertrauensverhältnis und auch der organisatorische und technische Umfang zwischen den einzelnen Akteuren geschaffen, um letztendlich den Austausch von Nutzerinformationen zwischen den IdP und SP zu ermöglichen [JD08, vgl. S. 228].

- Kostenreduktion –
  Ein effizientes IdM kann den Arbeitsaufwand für Mitarbeiter und Administratoren erheblich erleichtern. Es können u. a. auch komplexe Systeme und deren Fachpersonal verringert werden, wenn zuvor unterschiedliche Systeme im Einsatz waren. Des Weiteren können viele personelle Abläufe, wie eine Neueinstellung oder der Wechsel eines Aufgabenbereichs innerhalb einer Organisation, bedingt durch die Anpassung von Zugriffsberechtigungen, vorteilhafter gestaltet werden. Es lassen sich durchaus auch Supportdienstleistungen wie die eines Helpdesk reduzieren, wenn die Nutzungsfreundlichkeit beispielsweise durch SSO erhöht wird oder ein selbstständiges Passwortzurücksetzen durch den Nutzer möglich ist. Dies alles schlägt sich natürlich positiv auf das zur Verfügung stehende Budget nieder [JD08, vgl. S. 228].

- Vermeidung von Redundanz und Erhöhung der Datenqualität –
  In Unternehmen kommt es oftmals vor, dass an mehreren Stellen gleiche oder ähnliche Personendaten in Verzeichnissen gepflegt werden. Außerdem müssen diese Benutzerdaten konsistent und aktuell auf den verschiedensten produktiven Systemen vorhanden sein. Durch diese Redundanz entstehen Kosten und auch die Qualität der zu verwaltenden Daten leidet darunter. Dieser Mehraufwand kann durch ein IdM vermieden werden, indem ein zentraler Datenbestand allen autorisierten Systemen automatisiert zur Verfügung gestellt wird. Es werden weniger Fehler bei der Dateneingabe gemacht, da die digitale Identität nur initial erhoben wird und alle Änderungen basierend auf diesem Datensatz erfolgen [JD08, vgl. S. 228].

- Schutz vor Missbrauch –
  Aufgrund einer automatisierten Vergabe und Kontrolle von Zugriffsberechtigungen wird die Zuteilung von zusammenhangslosen, zu hohen oder auch veralteten Berechtigungen vermindert. Ferner dürfen Personen nur auf Informationen und Ressourcen Zugriff erlangen, für die es eine rechtmäßige Begründung gibt. Dieses kann durch eine automatisierte Richtlinienkontrolle eingehalten werden. Außerdem soll ein Missbrauch bei Ausscheiden eines Mitarbeiters durch das IdM umgangen werden, indem schnellstmöglich alle zugehörigen Nutzerkonten und die damit einhergehenden Zugriffsberechtigungen entfernt werden [JD08, vgl. S. 228f.].

- IT-Compliance und Audit –
  Allein aus rechtlicher Sicht muss ein Identitätsmanagement in Unternehmen vorhanden sein. *IT-Compliance* steht in diesem Zusammenhang für die Einhaltung aller gesetzlichen Regelungen im IT-Bereich. Erst durch Vorhandensein eines IdM besteht die Möglichkeit, z. B. Zugriffe auf geschützte Ressourcen zu auditieren. Auch wird hiermit die Grundlage geschaffen die Schutzziele der Informationssicherheit – Authentizität, Integrität, Vertraulichkeit, Verfügbarkeit und Verbindlichkeit bzw. Nachprüfbarkeit [JD08, vgl. S. 188] – aufrechtzuerhalten [JD08, vgl. S. 229].

Auch aus der Nutzersicht bietet ein vorhandenes und den gesetzlichen Vorgaben entsprechendes Identitätsmanagement Vorteile, auf die sogar in vielerlei Hinsicht ein vertraglicher / rechtmäßiger Anspruch besteht. Das Erheben, Speichern und Verarbeiten von sensiblen Nutzerdaten – personenbezogene Datensätze gehören hier ausnahmslos dazu – wird durch etliche Richtlinien bzw. Gesetzestexte in Deutschland aber auch auf der europäischen Ebene vorgeschrieben und bei Verstößen mit enormen Strafen geahndet. Insgesamt wird hier auf acht hervorgehobene Aspekte aus der Nutzersicht eingegangen [JD08, vgl. S. 229f.]:

- Benachrichtigung –
  Im Falle der Erhebung von personenbezogenen Daten muss der Nutzer darüber in Kenntnis gesetzt werden, wer die Datensätze erhält und speichert. Darüber hinaus muss kenntlich gemacht werden, was für Daten übermittelt werden, wie die Speicherung erfolgt und ob eine Übermittlung an Dritte stattfindet [JD08, vgl. S. 229].

- Wahlmöglichkeit –
  Der Nutzer muss einen Einfluss darauf haben können, zu welchem Zweck und vor allem an wen seine sensiblen Informationen geschickt werden. Des Weiteren muss eine Einsicht vorhanden sein, in der Zustimmungen und Verweigerungen des Nutzers abgebildet werden, die im Datenverarbeitungsprozess von ihm getroffen wurden. Eine Möglichkeit diese Aussagen zu berichtigen, sollte auch existieren [JD08, vgl. S. 229].

- Benutzerzugang zu Identitätsinformationen –
  Alle im Kontext des Identitätsmanagements gespeicherten Daten einer Person müssen dieser zugänglich gemacht werden. Es muss ihr demnach über einen bereitgestellten Zugang des Betreibers ermöglicht werden, Einsicht über die Identitätsinformationen zu erlangen [JD08, vgl. S. 229].

- Beschwerdemöglichkeit –
  Macht der Nutzer die Beobachtung, dass ein vermeintlicher Missbrauch seiner personenbezogenen Daten zustande gekommen ist, muss ihm die Aussicht gegeben werden, eine Beschwerde einreichen zu können [JD08, vgl. S. 229].

- Zweckbindung –
  Einmal erhobene Identitätsinformationen einer Person dürfen nur für den damals zugebilligten Zweck Verwendung finden; Abweichungen sind grundsätzlich nicht zulässig [JD08, vgl. S. 230].

- Qualität –
  Über einen angemessenen Zugang muss dem Nutzer sichergestellt werden, jederzeit eine Korrektur seiner gespeicherten Daten vornehmen zu können [JD08, vgl. S. 230].

- Zeitbeschränkungen –
  Personenbezogene Daten dürfen nur so lange eine Verwendung im Identitätsmanagement des Betreibers finden, wie sie tatsächlich benötigt werden oder inwiefern der Nutzer zum Erhebungszeitpunkt der Verwendung eingewilligt hat [JD08, vgl. S. 230].

- Sicherheit –
  Identitätsdaten sind sensibel zu behandeln, daher müssen umfangreiche Maßnahmen aus Sicht der Informationssicherheit ergriffen werden, um unberechtigten Zugriff oder gar den Verlust der Informationen zu verhindern bzw. vorzubeugen [JD08, vgl. S. 230].

Wie in den vorherigen Absätzen dargestellt, ist das IdM ein komplexes Konstrukt und wird zunehmend durch viele rechtliche Bestimmungen beeinflusst. Folglich legen die genannten Faktoren dessen Einsatz in Unternehmen / Organisationen nahe.

Ferner kann das Identitätsmanagement in seinem umfangreichen Handlungsbereich grob betrachtet in vier Hauptkomponenten eingeteilt werden: Digitale Identitäten, Identitätsspeicher, Integration von Identitätsspeichern und Identitätsmanagement-Prozesse [JD08, vgl. S. 230]. Diese Bestandteile werden in den nachfolgenden Abschnitten erläutert. Zuvor werden allerdings die beiden zentralen Konzepte der Authentifizierung und Autorisierung behandelt. Egal wo Personen oder Maschinen mit Systemen interagieren, sind diese Sicherheitskonzepte von essentieller Relevanz.

2.1.1 Authentifizierung

Die Authentifizierung überprüft mit geeigneten Maßnahmen die Authentizität einer Entität. „Unter der Authentizität eines Objekts bzw. Subjekts [...] [wird] die Echtheit und Glaubwürdigkeit des Objekts bzw. Subjekts, die anhand einer eindeutigen Identität und charakteristischen Eigenschaften überprüfbar ist“ [Eck18, S. 8], verstanden. Folglich weist die Authentifizierung / Authentifikation die Übereinstimmung von behaupteter Identität einer Entität mit dessen charakteristischen Eigenschaften nach [Eck18, vgl. S. 8, 443ff].

Im Allgemeinen kann zwischen zwei verschiedenen Authentizitäten unterschieden werden: Subjekt-Authentizität und Objekt-Authentizität [Eck18, vgl. S. 8f.]. Aus der Sicht der Entität wird die hier beschriebene Sicherheitsmaßnahme Authentisierung genannt. Einfach ausgedrückt wird die Identität und Echtheit einer Entität geklärt, um letztendlich eine Zugangskontrolle durchzuführen [JD08, vgl. S. 209].

Generell wird eine Authentifizierung in Systemen nur für Subjekte – also Nutzer – durchgeführt. Die Identifikation erfolgt anhand der Zuordnung von eindeutigen Benutzerkennungen bzw. Benutzernamen. Die für den Nachweis der Identität charakteristischen Eigenschaften können z. B. die Abfrage eines Passworts – dieses Wissen wird vom Nutzer beim Systemzugang abgefragt – oder sogar biometrische Merkmale wie Fingerabdrücke sein. Außerdem gibt es Maßnahmen, die auf einen persönlichen Besitz abzielen. Die Identitätsnachweise zur Klärung der Authentizität werden häufig auch als *Credentials* bezeichnet. In der Praxis kommen oftmals Kombinationen von ein oder mehr Authentifikationstechniken zum Einsatz, um die Sicherheit und den jeweiligen Vorteil der Technik gezielt auszunutzen; es wird dann von einer Zwei- bzw. Mehr-Faktor-Authentifikation gesprochen [Eck18, vgl. S. 8, 443ff].

Die Relevanz die Authentizität auch von Objekten, wie beispielsweise Webserver, *Access Points* oder Software-Code, zu klären, nimmt mit dem Übergang zu offenen IT-Systemen zu. Kryptografische Verfahren kommen hier als einfache Mechanismen zum Tragen, die die Echtheit von Daten überprüfen, wenn eine Übertragung über einen unsicheren Transportkanal wie der des Internets durchgeführt wird. Anzumerken ist, dass sich diese Authentifizierung von Objekten lediglich auf einen Ursprungs- bzw. Urhebernachweis beschränkt und keinen Nachweis der Funktionalität führt. Die vollständige Authentizität von Objekten zu überprüfen, erweist sich in der Praxis als schwierig und wird auch noch nicht eingesetzt. Für den sicheren Einsatz von offenen Systemen ist es aber dringend erforderlich, neben dem Ursprungsnachweis auch einen Beleg über die korrekte Funktionsweise zu bekommen [Eck18, vgl. S. 8f., 443ff].

2.1.2 Autorisierung

Eine Autorisierung erteilt einem Subjekt die Berechtigung zum Zugriff auf eine Information oder ein Datenobjekt. Ein Zugriff auf eine Information ist dadurch charakterisiert, dass eine Interaktion im System zwischen einem Subjekt und einem Objekt vorhanden ist. Bei dieser Interaktion findet ein Informationsfluss zwischen den Entitäten statt. Damit das Subjekt auf die zu schützenden Informationen bzw. auf die repräsentierenden Daten eines IT-Systems Einsicht

erlangt, sind Zugriffsberechtigungen festzulegen und den jeweiligen Subjekten zuzuordnen [Eck18, vgl. S. 5, 615f.]. Die hier beschriebene Zugriffskontrolle ist zeitlich gesehen nach einer erfolgreichen Zugangskontrolle – also Authentifizierung – platziert. Aufgrund erteilter Zugriffsrechte wird der Entität der Zugriff auf eine Ressource gewährt oder eben verweigert. Dieser Kontrollmechanismus wird je nach System durch ein Sicherheitsmodell definiert [JD08, Eck18, vgl. S. 209, 615f.].

2.1.3 Digitale Identitäten

Die Basis eines Identitätsmanagements bildet die Vielzahl aller in ihr verwalteten digitalen Identitäten. Dabei wird eine in der Wirklichkeit existierende Entität einer auf dem System platzierten digitalen Identität als Repräsentant zugeordnet. Unter Entitäten werden „reale und juristische Personen, aber auch jedwede Objekte, die durch Attribute beschrieben werden können“ [JD08, S. 207f.], verstanden. Damit letztendlich Zugangs- und Zugriffskontrollen auf einem konkreten IT-System stattfinden können, muss einer Entität eine spezifische digitale Identität zugeordnet werden. „Eine digitale Identität besteht aus einem Identifikator, welcher sie im Kontext eines Systems eindeutig identifiziert, und einer Menge zugehöriger Attribute“ [JD08, S. 208]. Beispiele für Identifikatoren sind die Matrikelnummer für StudentInnen oder die Personalausweisnummer für BürgerInnen in Deutschland. Auch nicht abgeleitete Identifikatoren wie ein weltweit eindeutiger *Universally Unique Identifier* (UUID) können ausreichen [LSM05, RFC 4122]. Es wurde bereits erwähnt, dass jedes IT-System die Notwendigkeit besitzt, einer Entität eine digitale Identität zuzuordnen. In einem IdM können Systeme auch unabhängig voneinander verwaltet werden, daher kann eine Entität auch mehrere digitale Identitäten in einem Umfeld besitzen [JD08, vgl. S. 207f.].

Eine digitale Identität besitzt immer einen fest definierten Lebenszyklus. Dabei ist das zugrunde liegende IdM – auch von der Komplexität her – nicht relevant. Drei Phasen lassen sich demnach unterscheiden [JD08, vgl. S. 231]:

1. Initiale Provisionierungsphase –
   In der ersten Phase werden die benötigten Informationen eines Nutzers initial aufgenommen. Darüber hinaus wird automatisiert ein Prozess angestoßen, der auf verschiedenen und für seine Arbeit benötigten Systemen Konfigurationen durchführt. „Die Provisionierung bezieht sich auf die Automation aller hierfür durchzuführenden Prozesse zur Erfassung und Verteilung von digitalen Identitäten, deren Attribute und Berechtigungen“ [JD08, S. 231]. In den automatisch ausgelösten Prozessen können die unterschiedlichsten Personen involviert sein, die beispielsweise ihre Zustimmung bei der Zuteilung bestimmter Zugriffsrechte geben müssen. Das Ziel dieser Phase ist es vorwiegend, weitere Erfassungen von Identitätsdaten und manuellen Zuordnungen von Zugriffsrechten auf den beteiligten Systemen zu vermeiden [JD08, vgl. S. 231].

2. Pflege- und Reprovisionierungsphase –
   Die zweite Phase erstreckt sich über die gesamte Zeit, in der eine Person in einer Organisation verbleibt. Es können Änderungen an der digitalen Identität im Laufe der Zugehörigkeit auftreten, die wiederum an die entsprechenden Systeme verteilt werden müssen. Es können sowohl die eigentlichen Identitätsdaten als auch die zugeordneten Rechte betroffen sein. Wechselt der Aufgabenbereich einer Person, so müssen u. a. die Zugriffsrechte auf bestehenden Systemen angepasst als auch die Zuweisung von Zugriffsrechten auf neuen Systemen durchgeführt werden [JD08, vgl. S. 231].

3. Deprovisionierungsphase –
   Diese Phase tritt in Kraft, wenn ein Nutzer eine Organisation verlässt bzw. aus ihr ausscheidet. Automatisiert müssen zugeordnete Rechte auf den Systemen entzogen, die zur Entität bezogenen Nutzerkonten gesperrt und ggf. personenbezogene Daten entfernt

werden. Sicherheitslücken, wie beispielsweise in Ausprägung von „Accountleichen“, sollen durch die Automation des Verfahrens vermieden werden [JD08, vgl. S. 231].

2.1.4 Identitätsspeicher

Digitale Identitäten werden in sogenannten Identitätsspeichern abgelegt und können damit dem Identitätsmanagement bereitgestellt werden. Die Identitätsspeicher können in unterschiedlichen Ausprägungen auftreten. Einfache Dateien, Datenbanken als auch für den ursprünglich vorgesehenen Zweck entworfene Verzeichnisdienste sind hier zu nennen. Der Unterschied zwischen Datenbanken und Verzeichnisdiensten liegt in deren Beschaffenheit, z. B. stellen sich relationale Datenbanken vor allem durch Transaktionsfähigkeit und Wahrung der referenziellen Integrität heraus. Bei Verzeichnisdiensten stehen wesentlich der hierarchische Aufbau der Daten in einer baumartigen Struktur und die Effektivität lesender Zugriffe im Vordergrund. „Hierdurch sind Verzeichnisdienste für den Einsatz als Identitätsspeicher ideal, denn Identitätsdaten werden häufig gelesen und unterliegen seltener Änderungen“ [JD08, S. 232]. Grob betrachtet verwalten diese spezialisierten Dienste eine Liste mit vielen Einträgen, die in einer Baumstruktur verankert sind und wiederum Eigenschaften / Attribute vorweisen. Repräsentativ stehen die Einträge für beliebige Objekte im IdM, wie beispielsweise Benutzer, Zertifikate, Computer oder auch Domänennamen. Verzeichnisdienste bieten daher nicht nur die Möglichkeit einer Benutzerverwaltung, sondern auch die Basis eines Netzwerk- und Ressourcenmanagements oder gar der Bereitstellung eines *Domain Name Systems* in Netzwerken. Ein weiterer Anwendungsfall wäre die Zurverfügungstellung einer *Public Key Infrastructure* (PKI), die öffentliche Schlüssel in Zertifikatsform ausliefert. Der Verzeichnisdienst würde in diesem Fall Zertifikate vorhalten, damit Anwendungen auf diese zugreifen können. Über dieses zentrale Verzeichnis kann schließlich mit geringen Aufwand eine digitale Signatur überprüft oder gar der öffentliche Schlüssel des Kommunikationspartners für eine Verschlüsselung angefordert werden. Es zeigt sich, dass Verzeichnisdienste einige weitere Anwendungsszenarien bieten als nur die Speicherung von personenbezogenen Daten [JD08, vgl. S. 231f.].

2.1.5 Integration von Identitätsspeichern

Werden digitale Identitäten nur in einem einzelnen Identitätsspeicher platziert, würden redundante Datenbestände und eine Synchronisation von mehreren Verzeichnissen keine Rolle spielen. In diesem Fall stellt folglich ein einziges Verzeichnis alle relevanten Daten im Unternehmenskontext bereit und die gesamten Anwendungen würden dieses idealisiert zur Authentifizierung oder Datenhaltung verwenden. Oftmals gibt es in einer Organisation aber keinen alleinigen zentralen Verzeichnisdienst, wodurch auch Nachteile vermieden werden sollen. Vorhandene Systeme oder Programme, die in den meisten Fällen eine lokale Datenhaltung besitzen, müssten an den zentralen Verzeichnisdienst angepasst werden; soweit eine Realisierung überhaupt möglich erscheint. Außerdem würden mit Einsetzen dieser zentralen Lösung die organisatorischen Einheiten ihre Eigenständigkeit verlieren. Um eine Zusammenführung der Datenbestände zu ermöglichen, muss die Zugriffskontrolle mit ihren den Entitäten zugewiesenen Berechtigungen geklärt sein. Dies ist wiederum nur mit einem umfassenden, zentralen Berechtigungsmanagement durchzusetzen, welches alle lokal vergebenen Rechte aufnimmt. Natürlich entsteht durch zunehmender Bündelung auch immer das Risiko eines *Single-Point-of-Failure*, dies bedeutet, dass ein Ausfall der zentralen Komponente, den Zusammenbruch des gesamten Systems nach sich ziehen würde. Die Konsequenz zielt auf die Integration der Identitätsspeicher ab. Lokal etablierte Verzeichnisse sollen möglichst dezentral aufgestellt bleiben. Der redundante Datenbestand muss hingegen durch eine Datensynchronisation vereinheitlicht werden, um die Datenqualität zu sichern. Eine

Synchronisation kann sich hier allerdings als schwierig erweisen, wenn verschiedene Kodierungen und Informationsmodelle vorherrschen [JD08, vgl. S. 237].

In der Praxis haben sich zwei klassische Ansätze bei der Integration von Identitätsspeichern verbreitet: Das *Meta Directory* und das *Virtual Directory*; zusammengesetzte Formen existieren natürlich auch [JD08, vgl. S. 237].

Das *Meta Directory* fasst alle Datensätze, die einer Entität zugeordnet werden, zu einem einzigen Eintrag im Verzeichnis zusammen. Dieser gesonderte Eintrag enthält infolgedessen alle Attribute einer Entität. Anwendungen können wiederum über definierte Schnittstellen auf diesen zugreifen. Suchoperationen sind in diesem Konstrukt performant, da die Datensätze bereits zentral vorliegen. Um im *Meta Directory* und den angeschlossenen Systemen einen einheitlichen Datenbestand zu garantieren, wird bei einer Eintragänderung zu einer Person eine Synchronisation angestoßen. Nachteile entstehen vor allem, wenn es zu Synchronisierungsverzögerungen kommt oder falls diese zentrale Komponente im IdM ausfallen sollte. Der initiale Aufwand bei einem solchen Aufbau – sprich die Integration aller Datenquellen – dürfte auch problematisch sein [JD08, vgl. S. 237ff].

Im Gegensatz zu dem oberen Ansatz steht das *Virtual Directory*, welches nur die Referenzen auf die lokalen Datensätze der Datenquellen einer Entität speichert. Die Quelldaten werden also nicht direkt im Verzeichnis gehalten. Findet ein Zugriff auf das *Virtual Directory* statt, so werden die benötigten Daten geholt, aggregiert und an den Anforderer ausgeliefert. Synchronisierungsabsichten zwischen den beteiligten Systemen sind im klassischen *Virtual Directory* nicht vorgesehen, da ein Attribut einer digitalen Identität immer nur aus einer Quelle stammen kann. Komplexere Datenflüsse, wie die in einem *Meta Directory*, werden somit vermieden. Dennoch kann die nicht vorhandene Synchronisation bei mehreren Datenquellen zu einem Nachteil führen, wenn gleichartige Informationen konsistent gehalten werden sollen. Außerdem kann die Verfügbarkeit angeforderter Datensätze zur Laufzeit nicht garantiert werden, da sie vom *Virtual Directory* selbst dynamisch angefordert werden müssen [JD08, vgl. S. 239].

2.1.6 Identitätsmanagement-Prozesse

Digitale Identitäten werden in Identitätsspeichern gehalten und weisen – wie weiter oben zu sehen – einen dreigliedrigen Lebenszyklus auf. Verwaltet werden diese Einträge durch Prozesse, die einen wichtigen Bestandteil im IdM übernehmen. Nachfolgend sind sechs Identitätsmanagement-Prozesse erläutert [JD08, vgl. S. 239f.]:

- Anlegen von Nutzerkonten –
  In diesem Prozess wird ein neues Nutzerkonto erzeugt und allen autorisierten Diensten, die mit digitalen Identitäten interagieren, werden diese Datensätze übermittelt. Nach Bekanntmachung des Kontos wird es dem neuen Nutzer erst möglich sein, alle Dienste mit seiner Identität auszuführen. Auch schon beim Erstellen der neuen Datensätze in den Identitätsspeichern kann es notwendig sein, weitere das IdM betreffende Maßnahmen einzuleiten. Konkret könnte dies bedeuten, dass bestimmte Personen benachrichtigt werden müssen oder sogar eine Zustimmung von (IT-)Verantwortlichen eingeholt werden muss. Durchaus können noch viel anspruchsvollere Prozesse an dieser Stelle gestartet werden; entscheidend ist hierbei die Komplexität des IdM [JD08, vgl. S. 239f.].

- Generierung von Daten –
  Bei der Erfassung und Verteilung von digitalen Identitäten ist es notwendig, ggf. weitere Prozesse, die basierend auf den erhobenen personenbezogenen Daten Attribute erzeugen, zu berücksichtigen. Beispiele sind hier das Erstellen von E-Mail-Adressen oder internen Personalnummern nach umfangreichen Richtlinien [JD08, vgl. S. 240].

- Anstoßen real weltlicher Vorgänge –
  Bei der Provisionierung kann es auch vorkommen, dass real weltliche Vorgänge gestartet werden müssen. Das Zustellen von Benutzername und Passwort über den Briefweg ist z. B. ein möglicher Anwendungsfall. Dieser könnte außerdem durch eine Frist unterstützt werden, in der der Nutzer gebeten wird, sich anzumelden, um die Sperrung oder Löschung seines Accounts nicht zu gefährden [JD08, vgl. S. 240].
- Konsistenzüberprüfung –
  Obgleich ein Identitätsmanagement besteht, welches automatisiert Nutzerkonten verwaltet, können durchaus Unstimmigkeiten in den beteiligten Systemen auftreten. Hervorgerufen werden können diese Inkonsistenzen durch manuelle Änderungen oder fehlgeschlagene Aktualisierungsprozesse. Notwendig ist es daher, Konsistenzüberprüfungen in regelmäßigen Abständen zu verwirklichen. Eventuell kann dies fortführend mit Bereinigungs- oder Benachrichtigungsprozessen gekoppelt werden [JD08, vgl. S. 240].
- Zurücksetzen von Kennwörtern –
  Ein Prozess, der das Zurücksetzen von Kennwörtern umsetzt, ist im Identitätsmanagement ein zentraler Bestandteil. Wichtig sind vor allem die Automation der Passwortrichtlinienüberprüfung als auch eine Bestätigung des neuen Passworts bzw. die Benachrichtigung hierüber. In Firmen stellt dieser Vorgang einen großen Kostenfaktor dar, denn je effizienter er gestaltet ist, desto größer ist die Entlastung des Personals / Helpdesks und weniger Ausfallzeiten fallen an [JD08, vgl. S. 240].
- Sperren von Nutzerkonten –
  Liegt ein Missbrauchsvorfall durch ein Nutzerkonto vor, muss dieses unverzüglich gesperrt werden. Des Weiteren muss der Eigentümer und auch dessen Verantwortlichen benachrichtigt werden [JD08, vgl. S. 240].

2.2 Identitätsverteilung

In der heutigen IT-Landschaft entwickelt sich immer stärker der Trend, organisationsübergreifende Geschäftsprozesse zu verwirklichen. Die vorherrschenden Sicherheitskonzepte wie die Authentifizierung und die Autorisierung dürfen dabei dennoch nicht vernachlässigt werden. Folglich besteht die Notwendigkeit Daten aus Geschäftsprozessen und digitale Identitäten in einer Föderation sicher und vertrauensvoll auszutauschen. Diesen Zusammenhang stellt das föderative Identitätsmanagement dar (s. Unterabschnitt 2.2.1). Ein weitverbreitetes Konzept sind zudem *Single Sign-on* (SSO) und das Pendant *Single Log-out* (SLO) (s. Unterabschnitt 2.2.2). Eine weitere Entwicklung stellt den Nutzer in den Mittelpunkt des föderativen Identitätsmanagements, dies wird im letzten Abschnitt aufgegriffen.

2.2.1 Föderatives Identitätsmanagement

Das föderative Identitätsmanagement (FIM) beschäftigt sich mit organisationsübergreifenden Anwendungsszenarien in verteilten Systemen. Es „kann als eine logische Evolution des Identitätsmanagements als Reaktion auf sich ändernde und steigende Anforderungen an die Verwaltung digitaler Identitäten verursacht durch den Einsatz in verteilten Systemen gesehen werden“ [Hö11, S. 25]. Bei dem Begriff der Föderation wird ein Bund selbstständiger Organisationseinheiten, die untereinander über eine Vertrauensbeziehung verfügen, assoziiert. Auch komplexere Geschäftsprozesse sollen in einem solchen Umfeld, welches den gemeinsamen und vereinfachten Zugriff auf Ressourcen von Föderationspartnern vorsieht, ermöglicht werden. Ohne die Entwicklung von Föderationsgemeinschaften könnten diese Prozesse nicht realisiert werden. Hier ist dann die Rede

von einem isolierten Modell, das bedeutet, dass unternehmensabschottende Maßnahmen einen Austausch verhindern. Nicht zu Letzt bietet die Mitgliedschaft in einer Föderation durchaus Kosteneinsparungen in der Administration sowie der Ressourcennutzung [Hö11, JD08, ABB+18, vgl. S. 25ff, S. 241f., S. 4f.].

In der Historie wurden zwei ernst zu nehmende Ansätze des FIM verwirklicht; wohingegen sich nur das verteilte System davon durchsetzen konnte [Hö11, vgl. S. 26].

Der erste Ansatz wird als zentralistisches föderatives Identitätsmanagement Modell bezeichnet. Der Grundansatz sah vor, redundante und inkonsistente Identitätsdaten zu vermeiden und den Nutzer eine einfache Handhabbarkeit seiner Daten zu ermöglichen. Im Mittelpunkt steht hierbei eine zentrale Komponente, nämlich der Identitätsanbieter, welcher sich um die Verwaltung identitätsbezogener Informationen kümmert. Des Weiteren hat er die Aufgabe über definierte Schnittstellen und Protokolle, Diensten und Organisationen alle notwendigen Informationen zukommen zulassen. Entscheidende Nachteile sind die Tatsache eines zentralen Angriffspunktes (*Single-Point-of-Failure*) und das notwendige Vertrauen von Kooperationspartnern in die eine einzelne, zentrale Komponente [Hö11, vgl. S. 26].

Der zweite Ansatz sieht die Umsetzung mit verteilten Identitätsanbietern vor; die Verantwortung wird also auf mehrere Anbieter verlagert. Die dezentralen *Identity Provider* haben auch hier die Aufgabe, digitale Identitäten zu verwalten sowie anderen Diensten und Einrichtungen im Kontext einer Föderation zur Verfügung zu stellen. Vereinfacht ausgedrückt wird hierdurch der folgende Grundsatz aufgestellt: „Lokal authentifizieren, global autorisieren“ [ABB+18, vgl. S. 4f.]. Dieses Modell wird als verteiltes föderatives Identitätsmanagement bezeichnet; im Rahmen dieser Arbeit wird sich bei FIM immer auf das verteilte System bezogen. Folgende vier Kernkomponenten besitzt dieser Ansatz [Hö11, vgl. S. 26f.]:

- Entität –
  In diesem Szenario wird hierunter eine reale Person verstanden (Subjekt), die mit einer behaupteten digitalen Identität auf eine Anwendung zugreifen möchte [Hö11, vgl. S. 26].
- Client –
  Der Client dient der Entität als Hilfsmittel, um auf bereitgestellte Inhalte von Partnern in der Föderation zuzugreifen. Im Falle von *Single Sign-on* werden maßgeblich Webseiten angesteuert. Daher kann der Client beispielsweise die Ausprägung eines Webbrowsers haben [Hö11, vgl. S. 27].
- Identitätsanbieter –
  Der Identitätsanbieter oder auch *Identity Provider* (IdP) ist sowohl für die Verwaltung identitätsbezogener Informationen als auch für den Identitätsaustausch in der Föderation zuständig. Er verwirklicht die Authentifizierung eines Benutzers und stellt auf ihn bezogene Identitätsdaten bereit [Hö11, vgl. S. 27].
- Diensteanbieter –
  Der Diensteanbieter stellt Benutzern im hier dargestellten Umfeld *Services* bereit. Er wird auch mit dem Titel *Service Provider* (SP) versehen [Hö11, vgl. S. 27].

2.2.2 Single Sign-on

Im Umfeld einer hier beschriebenen dezentralen / verteilten AAI stellen die *Service Provider* (SP) und *Identity Provider* (IdP) in ihrer Vielschichtigkeit die Basis dar. Wegen des verteilten Aufbaus dieser Infrastruktur hat sich ein Ansatz entwickelt, der die Anmeldung des Nutzers an jedem einzelnen SP nicht mehr notwendig macht. Dieses Verfahren wird als *Single Sign-on* (SSO) verstanden. Das wesentliche Ziel ist es dabei, die Anzahl benötigter Anmeldungen an

Systemen zu reduzieren bzw. für den Nutzer zu automatisieren. Folglich hat der Endanwender nach erfolgreicher Authentifizierung bei einem Diensteanbieter in einem bestimmten Zeitfenster auch den Zugriff auf alle anderen Dienste in diesem Kontext; eine effizientere und optimierte Arbeitsumgebung kann so geschaffen werden. Die Authentizität einer Entität wird durch einen sogenannten Identitätsanbieter kontrolliert. Nach erfolgreicher Prüfung der Identität wird der Diensteanbieter durch sicher übermittelte Daten darüber in Kenntnis gesetzt. Der SP hat abschließend die Möglichkeit über den Umfang der Autorisierung der digitalen Identität zu entscheiden. Für die Verwaltung der verschiedenen Entitäten bedient sich der IdP an einem Identitätsspeicher. Beide Dienste können zusammengefasst werden, oder jeweils auch als eigenständige Implementierungen vorliegen. Es gibt viele verschiedene Ausprägungen, die sich in Komplexität und Benutzerfreundlichkeit stark unterscheiden [KK11, MD11].

Neben der Einmalanmeldung gibt es auch dazugehörig den Ansatz einer Einmalabmeldung. Diese Umsetzung wird *Single Log-out* (SLO) genannt. Dieser Mechanismus ermöglicht es, aktive Sitzungen eines Nutzers nahezu gleichzeitig zu beenden. Der Abmeldevorgang kann durch den Benutzer selbst, einem Administrator oder von einem IdP oder SP aufgrund einer Zeitüberschreitung erfolgen. Die Anfrage kann also von jedem Sitzungsteilnehmer ausgelöst werden, der Einfluss auf den zugehörigen Authentifizierungskontext hat [KK11, MD11].

Es gibt viele Vorteile aber auch einige Nachteile, die *Single Sign-on* mit sich bringt. Um auf alle Systeme zuzugreifen, wird nur noch eine Anmeldung verlangt und damit sorgt es für eine Zeitersparnis beim Nutzer. Des Weiteren wird ein Sicherheitsgewinn erzielt. Das Passwort muss nur noch einmal pro Sitzung übertragen werden und der Nutzer benötigt in seinem Arbeitsalltag lediglich ein einziges Passwort. Ihm kann zugemutet werden, dieses nun komplexer und sicherer zu wählen, anstatt sich eine Menge von unsicheren Passwörtern zu merken. Auch werden *Phishing*- bzw. *Pharming*-Attacken mit Hilfe von SSO erschwert, da nur noch an einer Stelle die *Credentials* vom Anwender abgefragt werden. Solche Dienste können auch aus Sicht der IT-Sicherheit erweitert werden, indem eine Mehr-Faktor-Authentifikation oder / und sogar biometrische Mittel verwendet werden. Auf der anderen Seite wird auch eine Abhängigkeit zum SSO-System geschaffen. Ist der Dienst, auf dem zugegriffen werden soll, verfügbar, muss auch das System für die Authentifizierung vorhanden sein. Ein zweiter Nachteil befasst sich mit der Sitzungslänge. Ist nämlich kein SLO-Verhalten implementiert, so bleibt der Zugang zu allen nicht geschlossenen Anwendungen bis zum Erreichen einer Zeitüberschreitung offen und der Zugriff von unautorisierten Personen wäre möglich. Der gravierendste Nachteil ist aber mit aller Voraussicht, dass mit dem Erlangen des Nutzerpassworts alle Systeme offen ständen [KK11, MD11].

2.2.3 Nutzerzentriertes Identitätsmanagement

Im Identitätsmanagement ist der Nutzer die zentrale Komponente und sollte auf gar keinen Fall vernachlässigt werden. Wichtig ist es den Nutzer gegen aktuelle Gefahren, wie beispielsweise *Phishing* bzw. *Pharming*, im Netz zu schützen. Probleme treten hierbei nicht durch die eingesetzten Technologien auf, sondern schlichtweg durch den „unkundigen“ Nutzer selbst. Das nutzerzentrierte IdM beschäftigt sich demnach vorwiegend mit dem Anwender und versucht den Umgang mit digitalen Identitäten benutzerfreundlich umzusetzen. Dieser Ansatz sollte natürlich auch in Föderationen für die Identitätsverwaltung berücksichtigt werden. Die grundlegenden Prinzipien, die dieses IdM thematisiert, werden auch als die sogenannten sieben *Laws of Identity* angegeben [JD08, vgl. S. 243ff]; in Deutschland wird dies teilweise durch Datenschutzgesetze bzw. -regelungen abgebildet:

- Kontrolle und Zustimmung des Benutzers –
  Vertrauen in Diensteanbietern kann nur entstehen, wenn es die Zustimmung des Nutzers erfordert, damit personenbezogene Daten zwischen Organisationen ausgetauscht werden

dürfen. Außerdem muss es dem Nutzer überlassen werden, welche digitale Identität er verwendet und welche Daten er damit von sich preisgibt [JD08, vgl. S. 244].

- Minimale Bekanntmachung bei eingeschränkter Nutzung –
  Identitätsanbieter dürfen nur einen so geringen Informationsgehalt wie möglich übermitteln, die die Gegenstelle elementar benötigt. Zum Beispiel kann die Bestätigung über die Volljährigkeit eines Nutzers ausreichen, damit das Geburtsdatum nicht bekannt gegeben werden muss [JD08, vgl. S. 244].
- Berechtigte Parteien –
  Der Nutzer muss über die am Identitätsaustauschprozess beteiligten Parteien informiert werden. Nur dadurch kann er die Entscheidung treffen, welche sensiblen Informationen übergeben werden dürfen [JD08, vgl. S. 244].
- Gerichtete Identitäten –
  Grundlegend ist eine Abgrenzung zwischen öffentlichen Identitäten, wie die eines öffentlich zugänglichen Webseitenzertifikats, und personenbezogenen Identitätsinformationen eines Benutzers zu treffen. Der Schutz der Privatsphäre bedarf einer hohen Priorität [JD08, vgl. S. 244].
- Pluralismus von Betreibern und Technologien –
  Im föderativen Kontext müssen die unterschiedlichsten Technologien der Identitätsanbieter unterstützt werden. Die Vielfalt an Betreibern und Techniken sollte gleichberechtigt nebeneinander bestehen können [JD08, vgl. S. 245].
- Menschliche Integration –
  Es ist wichtig, dass der Mensch als Teil des Systems wahrgenommen wird. Es muss eine unmissverständliche Kommunikation zwischen dem System und dem Nutzer denkbar sein. [JD08, vgl. S. 245].
- Konsistente Erfahrungen unabhängig vom Kontext –
  Unabhängig vom Betreiber des Diensteanbieters als auch der eingesetzten Technologie muss die Verwendbarkeit der eigenen digitalen Identität aus der Sicht des Nutzers einfach und einheitlich sein. Dabei sollte die angesprochene Verwendbarkeit folgende allgemeine Funktionen beinhalten: Anschauen, Hinzufügen und Löschen der digitalen Identität [JD08, vgl. S. 245].

3 Vorhandene Komponenten

Im ersten Abschnitt werden bedeutende Protokolle und Verfahren im Kontext des (föderativen) Identitätsmanagements vorgestellt. Ein besonderes Augenmerk ist auf SAML (s. Unterabschnitt 3.1.1) und LDAP (s. Unterabschnitt 3.1.5) zu richten, da sie im späteren Verlauf dieser Arbeit noch Anwendung finden. Die restlichen Abschnitte befassen sich mit den fünf wesentlichen Komponenten, die für die Realisierung des praktischen Teils der Bachelorarbeit notwendig sind. Insbesondere wird hier die eingesetzte Software bzw. Infrastruktur vorgestellt, als auch deren Funktion erläutert.

3.1 Protokolle und Verfahren

In diesem Abschnitt werden Maßnahmen vorgestellt, die eine Identitätsverteilung in Föderationen ermöglichen. Wie bereits im vorherigen Unterabschnitt 2.2.1 erwähnt, beschäftigt sich das FIM mit organisationsübergreifenden Anwendungsszenarien in verteilten Systemen; „verteilte Systeme“ ist hierbei das bedeutende Stichwort. Die ersten vier Unterabschnitte befassen sich mit einer konkreten Umsetzung der Zugriffskontrolle / Identitätsprüfung. Abschließend behandelt der letzte Unterabschnitt 3.1.5 das Protokoll LDAP, welches in Verzeichnisdiensten (Identitätsspeichern) eine bedeutende Rolle einnimmt.

3.1.1 SAML

Die *Security Assertion and Markup Language* (SAML) ist ein XML-Standard, der sich mit dem Transport von Authentifikations- und Autorisierungsinformationen im browserbasierten Umfeld beschäftigt. SAML wurde von der internationalen Standardisierungsorganisation OASIS im Mai 2002 spezifiziert. Der Austausch der sensiblen Identifikationsdaten erfolgt dabei durch fest definierte Bescheinigungen, den sogenannten *Assertions*. Außerdem werden in diesem Standard Protokolle definiert, die XML-basierte Anfrage- und Antwortnachrichtenformate verwenden, um *Assertions* von SAML-*Authorities* anzufordern und anschließend eine Antwort in Form einer zugehörigen Bescheinigung zurückzubekommen. Durch die generische Aufstellung des SAML-Protokolls können die unterschiedlichsten Kommunikationsprotokolle zum Tragen kommen [Eck18, vgl. S. 656].

Bescheinigungen können in drei Klassen eingeteilt werden: Authentisierungs-, Autorisierungs- und Attribut-*Assertions*. Weiter unten werden die Bescheinigungen noch einmal genauer mit Hilfe von Listings betrachtet. Allgemein beinhaltet eine SAML-Bescheinigung immer Aussagen über ein Subjekt, für welches diese ausgestellt wurde. Konkret können die Aussagen den Namen, Merkmale der Authentifizierung und die Organisationszugehörigkeit des Subjektes, oder aber auch auf welche Ressourcen zugegriffen werden darf, beinhalten. Die Gültigkeit dieser Bescheinigung wird durch die ausstellende SAML-*Authority* (*Asserting Party*) garantiert. Alle bei der Kommunikation auftretenden XML-basierten Dokumente sind durch XML-Schemata strukturiert [Eck18, vgl. S. 656].

Im Kommunikationsgeschehen einer SAML-Interaktion sind mindestens zwei Parteien involviert. Die *Asserting Party* bzw. SAML-*Authority* nimmt hierbei die Rolle ein, Bescheinigungen über

Subjekte auszustellen; dies ist mit einer CA (*Certificate Authority*) in einer PKI gleichzustellen. Die andere interagierende Partei wird *Relying Party* genannt und vertraut den ausgestellten *Assertions*. Damit die Aussagen über die Entitäten akzeptiert werden können, muss ein Vertrauensverhältnis zwischen den Interaktionspartnern, wie in einer AAI vorausgesetzt, arrangiert sein. Dieses Verhältnis wird durch Austausch von Metadaten – meistens in signierter Form – der beteiligten Parteien erschaffen [Eck18, RHP+08, vgl. S. 656].

Im Abschnitt A.1 sind die unverschlüsselten Originaldateien einer SAML-Kommunikation zwischen der *Relying Party* „*unitysrv1.awi.de*“ und der *Asserting Party* „*shib-idp2.awi.de*“ abgebildet. Diese zwei Instanzen werden im praktischen Teil dieser Ausarbeitung noch an Bedeutung gewinnen. Weiter unten stehend wird auf fünf Teilbereiche dieser *Assertion* des SAML-*Response* eingegangen [HCH+05, vgl. S. 14ff].

Das *Issuer*-Element enthält die eindeutige „*EntityId*“ des *Identity Providers*, dies wird in Listing 3.1 ersichtlich. Listing 3.2 gibt Auskunft über das betroffene Subjekt. Es wird hier eine „*NameID*“ hinterlegt, mit der alle beteiligten Parteien genau dieses Subjekt zuordnen können. Das Listing 3.3 zeigt Bedingungen an, unter denen die Bescheinigung als gültig betrachtet wird. Hier sind u. a. die Zielgruppe und ein Zeitraum fixiert.

```
<saml2:Issuer>https://shib-idp2.awi.de/idp/shibboleth</saml2:Issuer>
```

Listing 3.1: SAML-*Assertion* – *Issuer*

```
<saml2:Subject>
  <saml2:NameID Format="urn:oasis:names:tc:SAML:2.0:nameid-format:persistent"
    NameQualifier="https://shib-idp2.awi.de/idp/shibboleth" SPNameQualifier="
    https://unitysrv1.awi.de">SWPEbjnFUQcPOwe0eU5/wNcxgKg=</saml2:NameID>
  <saml2:SubjectConfirmation Method="urn:oasis:names:tc:SAML:2.0:cm:bearer">
    <saml2:SubjectConfirmationData Address="..." InResponseTo="
    SAMLY2lib_msg_3723f93554f1d3c3bf08dfa6432cd911a6221db9745d86a7" NotOnOrAfter="
    2018-10-20T14:14:09.461Z" Recipient="https://unitysrv1.awi.de:443/unitygw/
    spSAMLResponseConsumer"/>
  </saml2:SubjectConfirmation>
</saml2:Subject>
```

Listing 3.2: SAML-*Assertion* – *Subject*

```
<saml2:Conditions NotBefore="2018-10-20T14:09:09.410Z" NotOnOrAfter="2018-10-20
    T14:14:09.410Z">
  <saml2:AudienceRestriction>
    <saml2:Audience>https://unitysrv1.awi.de</saml2:Audience>
  </saml2:AudienceRestriction>
</saml2:Conditions>
```

Listing 3.3: SAML-*Assertion* – *Conditions*

```
<saml2:AuthnStatement AuthnInstant="2018-10-20T14:08:18.569Z" SessionIndex="
    _9bd53c9f0fc8a2255966ae187d4b36a5">
  <saml2:SubjectLocality Address="..."/>
  <saml2:AuthnContext>
    <saml2:AuthnContextClassRef>urn:oasis:names:tc:SAML:2.0
    :ac:classes:PasswordProtectedTransport</saml2:AuthnContextClassRef>
  </saml2:AuthnContext>
</saml2:AuthnStatement>
```

Listing 3.4: SAML-*Assertion* – *AuthnStatement*

Das *AuthnStatement*-Element in Listing 3.4 beschreibt die vollzogene Authentifizierung beim IdP. Es sind hier das Authentifikationsverfahren „*PasswordProtectedTransport*“ und ein Zeitpunkt angegeben, in dem sich das Subjekt authentifiziert hat.

```
<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>Fabian Mangels</saml2:AttributeValue>
  </saml2:Attribute>
  <saml2:Attribute FriendlyName="uid" Name="urn:oid:0.9.2342.19200300.100.1.1"
    NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri">
    <saml2:AttributeValue>fmangels</saml2:AttributeValue>
  </saml2:Attribute>
  <saml2:Attribute FriendlyName="mail" Name="urn:oid:0.9.2342.19200300.100.1.3"
    NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri">
    <saml2:AttributeValue>fabian.mangels@awi.de</saml2:AttributeValue>
  </saml2:Attribute>
  <saml2:Attribute FriendlyName="displayName" Name="urn:oid:2
    .16.840.1.113730.3.1.241" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-
    format:uri">
    <saml2:AttributeValue>Fabian Mangels</saml2:AttributeValue>
  </saml2:Attribute>
  <saml2:Attribute FriendlyName="sn" Name="urn:oid:2.5.4.4" NameFormat="
    urn:oasis:names:tc:SAML:2.0:attrname-format:uri">
    <saml2:AttributeValue>Mangels</saml2:AttributeValue>
  </saml2:Attribute>
  <saml2:Attribute FriendlyName="givenName" Name="urn:oid:2.5.4.42" NameFormat="
    urn:oasis:names:tc:SAML:2.0:attrname-format:uri">
    <saml2:AttributeValue>Fabian</saml2:AttributeValue>
  </saml2:Attribute>
</saml2:AttributeStatement>
```

Listing 3.5: SAML-*Assertion* – *AttributeStatement*

Das letzte hier zu erwähnende Element in Listing 3.5 behandelt die Attributanweisungen. Damit werden weitere Informationen / Attribute über ein spezifisches Subjekt festgelegt, die von der *Relying Party* im vom Nutzer zugestimmten Umfang verwendet werden dürfen. Wie SAML in der DFN-AAI eingesetzt wird, kann im Abschnitt 3.2 nachgelesen werden; ein typischer SSO- (s. Abbildung 3.7) und SLO-Ablauf (s. Abbildung 3.8) werden dort auch aufgezeigt.

Zusammengefasst besteht das XML-*Framework* aus den vier folgenden Hauptkomponenten [JD08, vgl. S. 242f.]:

- Assertions –
  Mit *Assertions* (dt. Bescheinigungen, Aussagen) werden Authentifikationsinformationen, Attribute oder auch Autorisierungsentscheidungen übermittelt. Ein Austausch dieser Bescheinigungen findet zwischen mindestens zwei Parteien (*Asserting Party*, *Relying Party*) statt, die in einem Vertrauensverhältnis stehen [JD08, vgl. S. 242].

- Protocols –
  Hierunter werden Anfrage- und Antwortprotokolle für die Übermittlung der *Assertions* verstanden [JD08, vgl. S. 242].

- Bindings –
  Bindings sorgen für die Abbildung des SAML-Protokolls auf Nachrichten- und Kommunikationsprotokolle, wie SOAP (*Simple Object Access Protocol*) und HTTP [JD08, vgl. S. 242].

- Profiles –
  Sie geben die Randbedingungen zum Einsatz von SAML vor. Dieses wird durch die Kombination von *Assertions*, *Protocols* und *Bindings* realisiert; ein konkreter Anwendungsfall (*Use Case*) wird hierdurch definiert [JD08, vgl. S. 243].

Um auch die Sicherheit in der Kommunikation und im Aufbau von Vertrauensbeziehungen zwischen unterschiedlichen Partnern zu gewährleisten, wird im SAML-*Framework* nahegelegt eine PKI zu verwenden und Sicherheitsstandards wie *XML-Signature* oder *XML-Encryption* anzuwenden. Letzteres dient dazu, digitale Signaturen zu erstellen oder Verschlüsselungen der Bescheinigungen vorzunehmen. Zudem sollte auch der Transport verschlüsselt werden, z. B. mit Standardverfahren wie SSL / TLS oder *IPSec*. Sicherheitsbedenken bestehen in der „flexiblen" Bindung zwischen Subjekt und einer dazugehörigen ausgestellten Bescheinigung. Werden die bereits erwähnten Sicherheitsprotokolle nämlich nicht berücksichtigt, könnten *Man-in-the-Middle*-Angriffe im SAML-Kontext drohen. Es existiert kein wirkliches Bindungs- und Nachweiskonzept, da die Subjekte nicht nachweisen können, dass die jeweilige Bescheinigung aktuell ist und sie auch der berechtigte Besitzer sind. Bescheinigungen könnten folglich von Dritten abgefangen und wieder eingespielt werden (*Spoofing*-Angriff). Eine Gegenmaßnahme wäre, die *Assertions* durch spezielle kryptografische Maßnahmen an das dazugehörige Subjekt zu binden; dies ist in der SAML-Spezifikation aber nicht vorgesehen [Eck18, vgl. S. 659].

3.1.2 OpenID Connect

Das von der *OpenID Foundation* entwickelte *OpenID Connect* (OIDC) ist ein weiteres Authentifizierungsprotokoll, welches auf *OAuth 2.0* basiert und somit Informationen über die Identitätsprüfung mit Hilfe von JSON- / REST-Nachrichten austauscht. Anstatt XML wie bei SAML zu verwenden, kommt hier die *JavaScript Object Notation* (JSON) zum Einsatz. Hauptaufgabe von OIDC ist die Authentifizierung eines Subjektes über einen Autorisierungsserver (*Identity Provider*) auf nahezu allen möglichen Clients. Außerdem wird die Übermittlung überprüfbarer *Assertions* – entsprechen Profilinformationen – jener Identität ermöglicht. Genutzt werden kann das Protokoll sowohl von browserbasierten als auch nativen und mobilen Anwendungen. Die Diensteanbieter (*Relying Parties*) kommen dabei nicht an die Passwortdateien heran und müssen diese deshalb auch nicht verwalten. Trotzdem wird die Identität des Nutzers durch die Verwendung von sogenannten *ID Tokens* ersichtlich, denn hier werden Aussagen über die betroffene Entität getroffen. Repräsentiert werden diese Tokens im Format eines *JSON Web Token* (JWT). Die Nachahmung eines anderen Nutzers soll hierdurch vermieden werden, was das reine *OAuth 2.0* nicht berücksichtigt; das Konzept der verschiedenen Identitäten existiert hier nämlich nicht [SBJ+14, OT18a, MMS16].

Der Vorteil von OIDC gegenüber SAML liegt vor allem in der Sicherheit (PKI), aber natürlich auch in der Möglichkeit ohne einen Webbrowser auf der Client-Seite auszukommen. Konzepte wie Föderationen mit signierten OIDC-Metadaten verfahren können, gibt es auch schon. Des Weiteren wird gerade an der Unterstützung in der IdP-Implementierung der Software *Shibboleth* gearbeitet. Damit ist die Grundlage geschaffen, um zukünftig in der DFN-AAI auf dieses Protokoll umzusteigen [Pem18b, vgl. S. 51f.].

3.1.3 OAuth 2.0

Die in den vorherigen Unterkapiteln beschriebenen Protokolle (SAML, OIDC) befassen sich grundsätzlich mit der Identität eines Subjektes. *OAuth 2.0* stellt dagegen die Zugriffsverwaltung auf Inhalte, die natürlich auch mit einer digitalen Identität verknüpft sind, in den Vordergrund, ohne dabei die Übermittlung eines Passwortes zu veranlassen. Das Protokoll ist offen und im

RFC 6749 standardisiert [Har12]. Der Wegfall des Passwortes (*Password Anti Pattern*), also die Unterbindung der Verarbeitung der *Credentials* durch einen Drittanbieter, ist auch hier einer der größten Merkmale des Protokolls. Damit Daten eines Dienstes auch in anderen dritten Diensten verwendet werden können, ermöglicht *OAuth 2.0* eine spezielle erlaubnisbasierte Prüfung und verzichtet auf die ursprüngliche vollständige Identitätsprüfung. Ende 2009 begannen die Arbeiten an der zweiten Version des Protokolls. Die einfache Bedienbarkeit und der Einsatzbereich sowohl für kleine als auch komplexe Installationen waren als Ziele definiert. Der vollständige Verzicht auf digitale Signaturen innerhalb des Protokolls und die konsequente Verwendung von HTTPS, Zufallszahlen und der Einmalverwendung von Parametern, führten zur Verbesserung der Skalierbarkeit im Vergleich zur ersten Version von *OAuth*. Serverseitige Dienste lassen sich hiermit komplett zustandslos implementieren. Obwohl *OAuth 2.0* als Autorisierungsprotokoll gedacht ist, wird es häufig für die Anmeldung auf Webseiten verwendet [LH13, Har12].

Abbildung 3.1 soll die beteiligten Akteure und den abstrakten *OAuth 2.0*-Protokollfluss verdeutlichen. Zentraler Bestandteil ist die Benutzerkontrolle, die vor unbefugtem Zugriff auf personenbezogene Datensätze schützt. Nutzer werden dementsprechend als *Resource Owner* und die verwaltende Ressourceninstanz als *Resource Server* bezeichnet. Von der letzteren Instanz wird erwartet, dass sie nur autorisierte Zugriffe des jeweiligen Nutzers akzeptiert und vollzieht. Eine Autorisierung wird durch sogenannte *Access Tokens* legitimiert, die von einem vertrauenswürdigen *Authorization Server* ausgestellt werden. Die Voraussetzung für die Erstellung der *Access Tokens* ist dabei die zuvor durchgeführte Authentifizierung des Nutzers. Der letzte Akteur ist der *Client*, welcher im Auftrag des Nutzers auf geschützte Ressourcen zugreift [LH13, Har12]. Ein *Client* bekommt den Zugriff auf die Daten des Nutzers nur, wenn dieser zugestimmt hat.

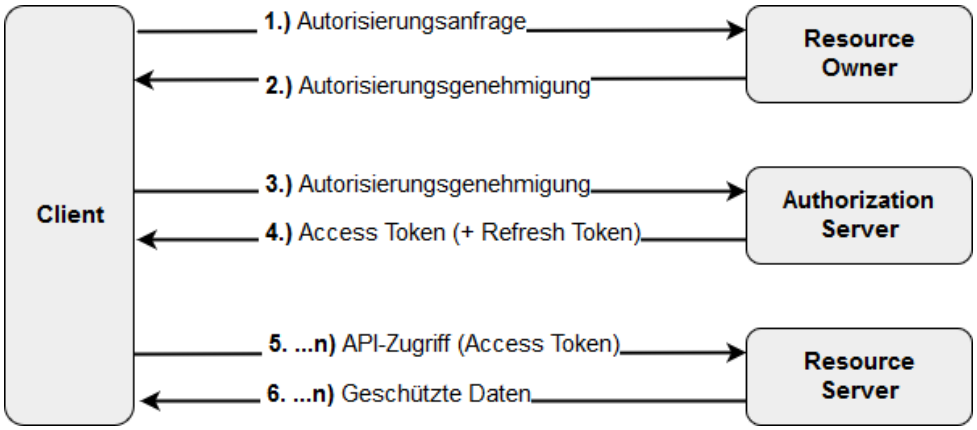

Abbildung 3.1: Abstrakter *OAuth 2.0*-Protokollfluss [Har12, vgl. S. 7, S. 11]

Dafür wird vom *Client* eine Autorisierungsanfrage (*Authorization Request*) an den *Resource Owner* gestellt (1.). Diese Anfrage kann entweder direkt vom *Client* erfolgen, was die Abfrage nach den erforderlichen *Credentials* nach sich zieht, oder es kann sich auch um eine Anfrage von einer Browserweiterleitung über den *Authorization Server* als Mittler handeln. Nach Zustimmung des Nutzers wird eine Autorisierungsgenehmigung (*Authorization Grant*) übermittelt (2.). Die ausgehändigte Genehmigung sendet der *Client* an den *Authorization Server* (3.) und erhält im Gegenzug ein *Access Token* zurück (4.). Dieses *Token* wird anschließend vom *Client* für die Autorisierung von Zugriffen beim *Resource Server* verwendet (5. ...n), der wiederum die geschützten Daten freigibt (6. ...n). *Access Tokens* haben in diesem Kontext eine begrenzte Lebensdauer. Damit der *Client* ein neues erzeugen kann, bedarf es der erneuten Rückfrage mit dem Nutzer oder aber die Verwendung eines *Refresh Tokens*. *Refresh Tokens* haben eine deutlich längere Gültigkeit und eine Nutzerinteraktion ist nicht mehr notwendig; sie können allerdings vom Nutzer jederzeit zurückgezogen werden [LH13, Har12].

3.1.4 JSON Web Token

Das *JSON Web Token* (JWT) ist ein offener Standard [JBS15, RFC 7519] für ein *Access Token*, das den sicheren Austausch von JSON-Objekten – sogar über ungesicherte Verbindungen – zwischen verschiedenen Parteien ermöglicht. Die übertragenen Informationen können verifiziert und vertrauenswürdig sein, wenn sie zuvor digital signiert worden sind. Zur Signierung kann ein öffentliches / privates Schlüsselpaar Anwendung finden, wobei der öffentliche Schlüssel allen beteiligten Akteuren bekannt sein muss. Das JWT zeichnet sich durch seine Kompaktheit und geringe Größe aus, daher ist die Übertragung schnell. Außerdem ist es in sich geschlossen, denn ein Bestandteil des *Tokens* sind Identitätsinformationen einer Entität, die nicht erneut irgendwo abgefragt werden müssen. Es wird zudem als Erweiterung von *OAuth 2.0* gesehen. Das JWT besteht aus drei *Base64*-kodierten Zeichenketten (*Header*, *Payload* und *Signature*), die wiederum mit einem Punkt („.“) voneinander getrennt sind. Neben den für die Verarbeitung erforderlichen / reservierten Informationen wie Herausgeber und Gültigkeitsdauer kann der Nutzer zusätzliche zu übermittelnde Daten bestimmen. Der zentrale Bestandteil ist aber sicherlich die digitale Signatur. *Header* und *Payload* werden verschlüsselt und zusammen mit einem *Secret* signiert, das den agierenden Akteuren bekannt sein muss. Damit ist sichergestellt, dass nur die Empfängerseite die Identität des Senders und die Integrität der Nachrichten überprüfen kann. Aufgrund der genannten Vorteile wird das JWT vorwiegend für die Authentifizierung im *Single Sign-on*-Kontext und zum sicheren Informationsaustausch verwendet. Der herkömmliche Sitzungsaufbau auf einem Server wird durch das *Access Token*, welches lokal beim Nutzer gespeichert wird, neu aufgegriffen. Es ist folglich ein zustandsloser Authentifizierungsmechanismus [AT18, JBS15].

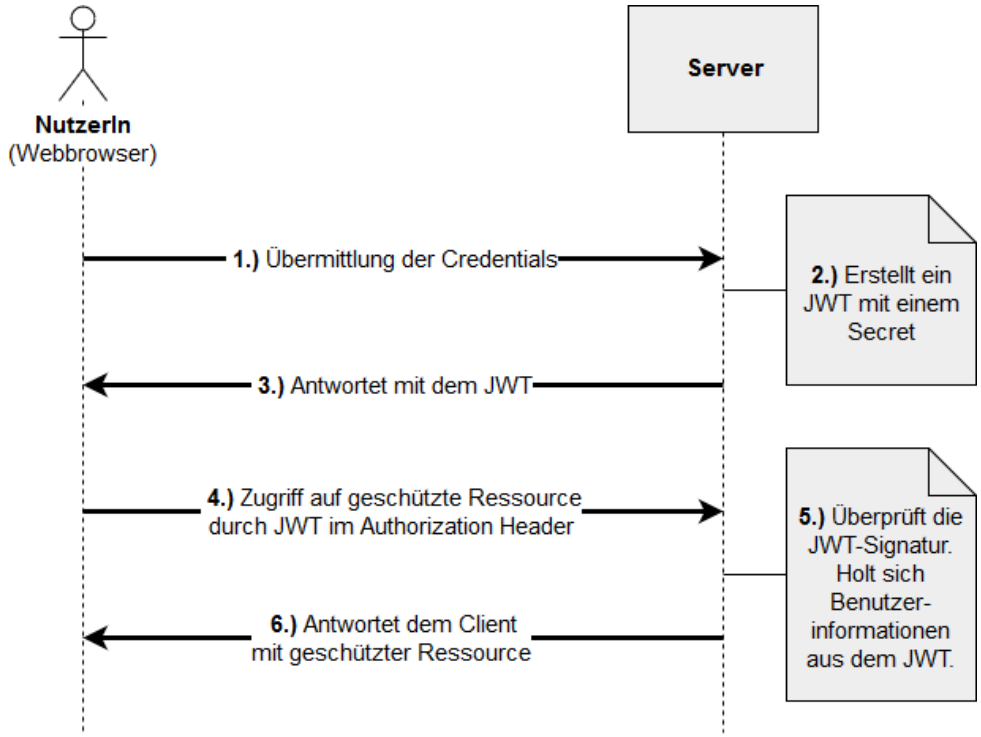

Abbildung 3.2: Authentifizierung mit einem JWT [AT18]

Wie genau die Authentifizierung und Autorisierung mit einem JWT funktioniert ist dem abstrakten Sequenzdiagramm in der Abbildung 3.2 zu entnehmen. Zu Beginn erhält der Nutzer nach erfolgreicher Anmeldung bei einem Dienst (1.) einen *JSON Web Token* vom Server zurück (3.) und speichert dieses lokal auf seinem *Client*. Zuvor muss der Server das *Access Token* mit einem *Secret* generiert haben (2.). Bei jedem weiteren Zugriff auf den Dienst wird durch die zusätzliche Übermittlung des JWT im *Authorization Header* der Anfrage die Authentizität des

Zugreifenden zugesichert (4.). Daraufhin überprüft der Server die Anfrage auf einen gültigen JWT, indem er die Signatur kontrolliert (5.). Liegt eine legitimierte Anfrage vor, antwortet der Server dem Nutzer und gestattet einen autorisierten Zugriff auf die geschützte Ressource (6.) [AT18, JBS15].

3.1.5 LDAP

LDAP steht für *Lightweight Directory Access Protocol* und dient dazu mit Verzeichnisdiensten zu operieren. Aufgrund der leicht zugänglichen Nutzung des Verzeichniszugriffsprotokolls hat sich dieses als Internetstandard durchgesetzt und wird fortlaufend von der IETF (*Internet Engineering Task Force*) weiterentwickelt. Ursprünglich war es als leichtgewichtige Alternative zum Zugriff auf *X.500*-Verzeichnisdienste entwickelt worden. LDAP ist in der dritten Version vorliegend und in diversen RFCs beschrieben [Zei06b, RFC 4510]; die eigentliche Protokollspezifikation ist im RFC 4511 [Ser06] zu finden. Es gibt bekannte Implementierungen des Standards von diversen Herstellern. Z. B. bietet *Microsoft* ab *Windows 2000* das *Active Directory* als zentralen Verzeichnisdienst an. Im *Open Source*-Bereich ist die Software *OpenLDAP* zu nennen, die bereits in vielen Rechenzentren eingesetzt wird. Auch im praktischen Anteil wird ein *OpenLDAP*-Server zur Identitätsspeicherung eingesetzt. In den folgenden Absätzen werden die grundlegenden Ideen von LDAP aufbereitet [JD08, vgl. S. 232f.].

Die Gesamtmenge aller Informationen, die ein Verzeichnis umfasst, wird als *Directory Information Base* (DIB) bezeichnet. Die Einträge in einer DIB sind strukturell gesehen wie ein hierarchischer Baum mit mehreren Wurzeln, Zweigen und Blättern angeordnet. Dabei ist jeder Eintrag als Objekt zu betrachten, welches im objektorientierten Ansatz eine konkrete Instanz einer oder mehrerer Objektklassen sein kann. Eine Objektklasse ist wiederum einem der drei Typen abstrakt, strukturell oder unterstützend zuzuordnen. Abstrakte Objektklassen können selbst nicht instantiiert werden, da von ihr andere Objektklassen abgeleitet werden. Die strukturellen Objektklassen sind Instanzen von anderen abstrakten oder auch strukturellen Objektklassen und erben von den Ursprungsklassen alle definierten Attribute. Der letzte Typ – also unterstützende Objektklassen – sind Hilfsklassen um Attribute zu definieren, die dann mehreren strukturellen Objektklassen zugewiesen werden können. Dabei kann es passieren, dass unterschiedlichen strukturellen Objektklassen die gleichen Attribute hinzugefügt werden, aufgrund der Zuweisung gleicher unterstützender Objektklassen. Eine Instanziierung der abstrakten und unterstützenden Typen ist nicht möglich. Die eigentlichen Einträge / Objekte im Verzeichnis dürfen nur einer einzigen strukturellen Klasse angehören. Sie können dennoch zu beliebig vielen unterstützenden Objektklassen zugehören.

Jede einzelne Objektklasse beinhaltet Attribute, mit denen ein Eintrag charakterisiert wird. Diese Attribute können eine optionale oder aber auch verbindliche Umsetzung in den Einträgen bedeuten. Außerdem können sie einen oder mehrere Werte, nach einem definierten Typ wie etwa Text, enthalten. Die dabei verwendeten Typenbezeichnungen sind überwiegend einfach zu merkende Kürzel, z. B. „*cn*“ für *Common Name*, „*ou*“ für *Organizational Unit*, „*dc*“ für *Domain Component* oder „*mail*“ für *E-Mail Address*. Die für die Struktur benötigten Metadaten eines Verzeichnisdienstes sind in Schemata aufgeführt. In ihr wird folglich auch definiert welche Objektinstanzen in einem Verzeichnis erlaubt sind [JD08, vgl. S. 233f.]. Einige relevante Objektklassen sind in der Tabelle 3.1 abgebildet.

In einem Verzeichnisdienst werden Einträge in einer hierarchischen Baumstruktur angeordnet. Diese Struktur wird auch *Directory Information Tree* (DIT) genannt und bildet ihren eigenen abgeschlossenen Namensraum. Alle vorhandenen Objekte im DIT besitzen einen *Relative Distinguished Name* (RDN), der sie auf ihrer jeweiligen Ebene eindeutig identifiziert. Um einen Knoten im gesamten Baum eindeutig zu referenzieren, wird der *Distinguished Name* (DN) verwendet. Er

Objektklasse	Zuständigkeit
top (abstrakt)	Alle Einträge gehören zur abstrakten Objektklasse „*top*“.
organizationalUnit (strukturell)	Definiert die Basis eines Eintrags, der eine Organisationseinheit darstellt.
organizationalPerson (strukturell)	Grundlage für einen Eintrag, der eine Person in Bezug auf eine Organisation repräsentiert.
inetOrgPerson (unterstützend)	Erweiterung der Objektklasse „*organizationalPerson*“, um den heutigen Anforderungen der Bereitstellung von Internet- und Intranet-Verzeichnisdiensten gerecht zu werden.
eduPerson (unterstützend)	Unterstützt den Austausch zwischen Bildungs- und Forschungseinrichtungen mit vordefinierten Attributen über Personen.

Tabelle 3.1: Grundlegende LDAP-Objektklassen [Smi00, Zei06a, Sci06, IT16, RFC 2798, RFC 4512, RFC 4519]

setzt sich aus allen RDNs auf dem Weg von der Wurzel des DIT bis zum spezifischen Knoten zusammen. Der verwendete Namensraum der praktischen Umsetzung kann aus der Abbildung 3.3 entnommen werden. Für eine Person – hier ist es ein Testkonto – ergibt sich beispielsweise der DN „*cn=Willi Wichtig,ou=People,dc=unity,dc=awi,dc=de*“, wobei u. a. „*ou=People*“ ein RDN ist. Des Weiteren erlaubt ein DIT bedingt durch die hierarchische Baumstruktur eine verteilte Administration und Datenhaltung. Ein Teilbaum, eine sogenannte Partition, kann auf einem eigenen Server platziert und administriert werden [JD08, vgl. S. 234f.].

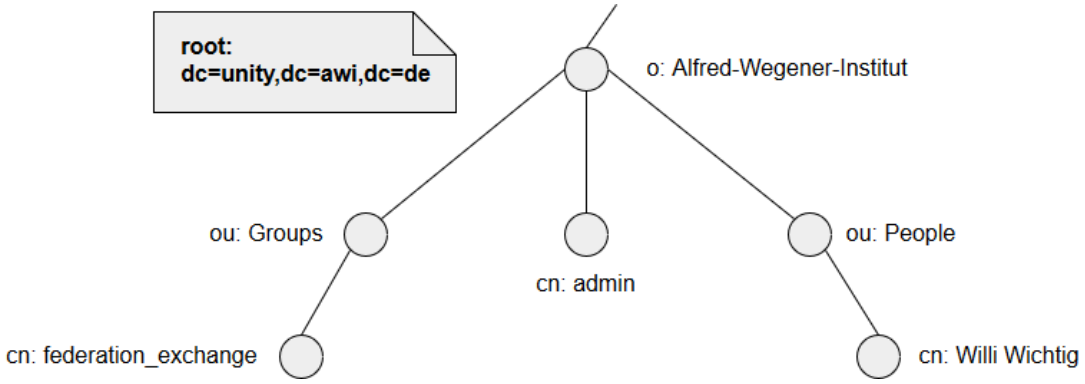

Abbildung 3.3: Verwendeter LDAP-Namensraum (DIT)

Im LDAP sind einige Operationen fest definiert, die auf ein Verzeichnis angewendet werden können [JD08, vgl. S. 235]. Eine Übersicht darüber vermittelt die Tabelle 3.2.

Operation	Erklärung
Bind	Beginn einer Sitzung.
Unbind	Beenden einer Sitzung.
Search	Suche nach Einträgen ab einer übergebenen Stelle, dem *Base DN*, im DIT.
Modify	Änderung eines Eintrags.
Add	Hinzufügen eines Eintrags an einer beliebigen Stelle im DIT.
Delete	Löschen eines Eintrags.
Compare	Vergleich eines Attributwertes mit einem spezifizierten Wert.

Tabelle 3.2: Zugriffsoperationen auf ein LDAP-Verzeichnis [JD08, vgl. S. 235]

Ein Verzeichnisdienst beherbergt nicht nur allgemein zugängliche Informationen, sondern speichert

oft auch sensible personenbezogene Daten, die besonders geschützt und einen ausschließlich autorisierten Zugriff notwendig machen. Die Zugriffskontrolle wird einerseits über verschieden starke Authentifikationsmechanismen verwirklicht und andererseits kommt eine *Access Control List*, die einen nutzer- und operationsspezifischen Zugriff auf Einträge möglich macht, zum Tragen. Die erforderliche Kommunikation kann hierbei über SSL / TLS abgesichert werden [JD08, vgl. S. 235].

Es werden im Wesentlichen drei verschiedene Mechanismen bei der LDAP-Authentifizierung unterschieden: *Anonymous Bind*, *Simple Authentication* und *Simple Authentication and Security Layer* (SASL). Die anonymisierte Sitzung erlaubt einen einfachen Lesezugriff auf ausgewählte Attribute bzw. Objekte, ohne eine Authentifizierung zu verlangen. Als Beispiel können hier öffentliche Verzeichnisdienste als Teil einer PKI genannt werden, die Zertifikate mit öffentlichen Schlüsseln allen Nutzern zur Verfügung stellen. Die *Simple Authentication* hingegen führt eine Authentifikation und dedizierte Zugriffskontrolle durch. Hierfür werden grundsätzlich der DN des Akteurs und das dazugehörige Passwort benötigt. Den dritten Authentifizierungsmechanismus stellt das SASL-*Framework* bereit [ZM06, RFC 4422]. SASL sieht für die Authentifizierung ein *Challenge-Response* Protokoll vor und erlaubt zudem die Nutzung von verschiedenen Mechanismen wie *Kerberos*. Zudem ist das *Framework* nicht ausschließlich auf LDAP ausgelegt, sondern es kann mit anderen Protokollen wie beispielsweise SMTP (*Simple Mail Transfer Protocol*) zusammenarbeiten. Dafür wird einfach eine Sicherheitsschicht zwischen Protokoll und Verbindung initialisiert. Obwohl diese Authentifizierungsmechanismen Subjekte für den Zugriff auf den Verzeichnisdienst identifizieren sollen, verwenden Anwendungen oftmals nur die reine Klärung der Authentizität einer Entität. Der Zugriff auf die Verzeichnisdaten findet demzufolge nicht statt [JD08, vgl. S. 235f.].

Wie bereits weiter oben angesprochen kann der DIT in Partitionen untergliedert werden. Diese Teilbäume werden folglich auf weitere Server ausgelagert, um eine Ausfallsicherheit durch Redundanz zu garantieren oder auch eine Leistungssteigerung des Dienstes zu erlangen; natürlich kann auch der gesamte DIT redundant betrieben werden. Bei einem verteilten Verzeichnisdienst gibt es zwei Möglichkeiten eine LDAP-Anfrage aufzulösen. Die erste Variante sieht die Übermittlung eines *Referrals* auf dem Server vor, der den Teilbaum, auf dem sich die Anfrage bezieht, kontrolliert. Voraussetzung hierfür ist, dass der LDAP-Client *Referrals* verarbeiten sowie weitere Anfragen dem Server stellen kann. Die zweite Variante wird *Chaining* genannt. In ihr wird die Auflösung der Anfrage auf dem LDAP-Server durchgeführt und die Antwort letztendlich dem Client übermittelt [JD08, vgl. S. 236].

Bei Vorhandensein von Replikationen bedarf es auch immer einer wohl überdachten Umsetzung von Schreibrechten auf den einzelnen Verzeichnisdiensten. Im LDAP sind die *Multi-Master*- und die *Single-Master*-Replikation vorgesehen. Bei der ersten Replikation sind Schreibzugriffe auf alle gleichberechtigten Server zulässig. Um einen Datenabgleich / Synchronisation zwischen den einzelnen Instanzen zu realisieren, sind oftmals komplexe Maßnahmen notwendig. In diesem Szenario wäre es fatal, wenn zeitgleiche Änderungen an demselben Eintrag geschehen oder ein automatischer Abgleich nach Ausfall der Verbindung zwischen den *Master*-Servern stattfindet. Die vorwiegend eingesetzte *Single-Master*-Replikation wirkt dem entgegen und erlaubt nur die Änderung von Datensätzen an einem ausgewiesenen *Master*-Server. Alle weiteren Server (*Slaves*) in diesem Verbund lassen nur einen lesenden Zugriff zu. Allerdings kann der *Master* diesen Slaves Änderungen konsistent übermitteln [JD08, vgl. S. 237].

3.2 DFN-AAI

Grundlage für die Umsetzung der Arbeit ist das Vorhandensein einer zusammenhängenden Infrastruktur verschiedener Einrichtungen, in der ein Austausch von Attributen und eine anschlie-

ßende Identitätsfeststellung jener Entität realisiert werden können. Ein solches Konstrukt wird *Authentication and Authorization Infrastructure* (AAI) genannt. Solch eine AAI kann sowohl nur lokal in einer Einrichtung angesiedelt sein, als auch organisationsübergreifend umgesetzt werden. Bei der letztgenannten Ausrichtung wird eine zentrale Instanz benötigt, die die technischen und rechtlichen Rahmenbedingungen sicherstellt und eine sogenannte „Vertrauensstellung“ zwischen den verschiedenen Akteuren verwirklicht. Diese Umsetzung geschieht in der Regel in einer Föderation wie die der DFN-AAI. Der DFN-Verein tritt hierbei als Föderationsbetreiber auf, das bedeutet, er schließt Verträge mit den Teilnehmern, verwaltet die (signierten) Metadaten und führt auch Zertifikatsüberprüfungen durch. Das DFN (Deutsches Forschungsnetz) im Allgemeinen verbindet Hochschulen und Forschungseinrichtungen innerhalb von Deutschland aber auch weltweit über ein Netzwerk (Internet) miteinander und stellt diverse Kommunikationsdienste zur Verfügung. Außerdem wird durch die Mitgliedschaft der Einrichtungen die Anbindung zum allgemeinen Internet geschaffen. Um aktiv in der AAI teilzunehmen, müssen Partner einen Identitätsanbieter bereitstellen, der mit dem Identitätsmanagement der jeweiligen Heimateinrichtung verknüpft ist. Einsatz findet hier u. a. die Software *Shibboleth* (IdP des AWI: `https://shib-idp.awi.de/idp/shibboleth`), die im SAML-Kontext als IdP oder auch als *Service Provider* betrieben wird. Damit die Kommunikation zwischen einem IdP und einem SP hergestellt werden kann, muss wie oben bereits erwähnt Vertrauen zwischen den unterschiedlichen Partnern bestehen. Dafür stellt der DFN-Verein an einer zentralen Stelle Metadaten bereit, die sowohl vertrauenswürdige IdP als auch SP auszeichnen [DT18, Pem18b].

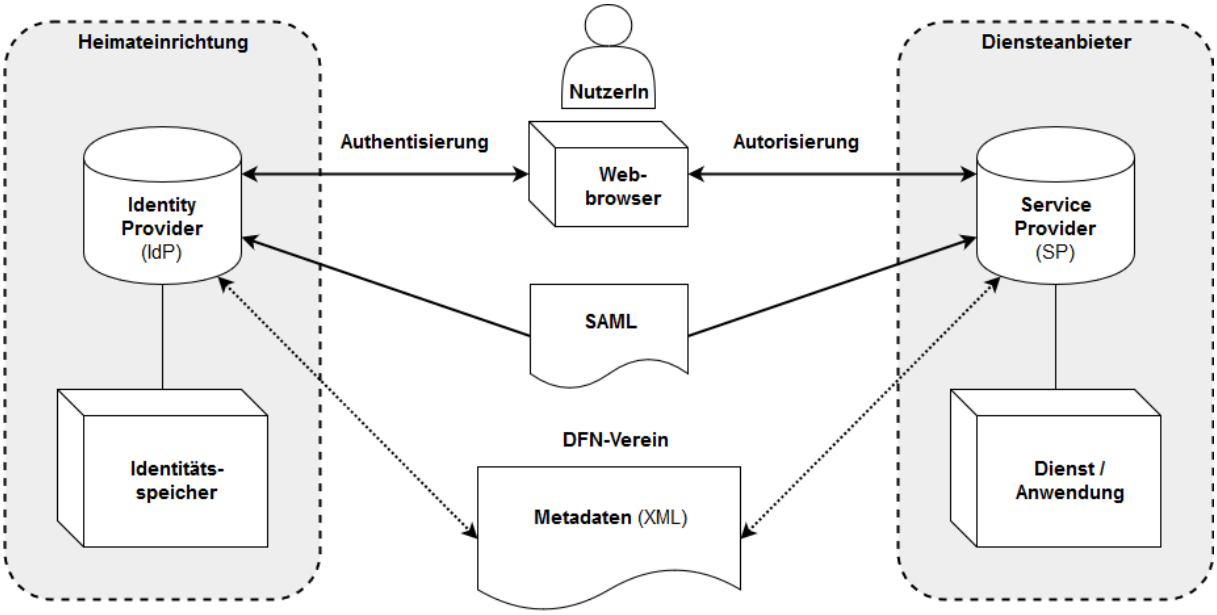

Abbildung 3.4: Grundlegende SSO-Interaktion in der DFN-AAI [Pem18b, vgl. S. 11]

Die Abbildung 3.4 zeigt den Zusammenhang zwischen einer Heimateinrichtung mit IdP, eines Diensteanbieters mit SP und dem Nutzer (Webbrowser). Der *Service Provider* nimmt den Nutzer wahr, der auf eine geschützte Ressource zugreifen möchte. Daraufhin generiert der SP eine Authentifizierungsanforderung (*AuthnRequest*) und leitet diese und den Nutzer zum *Identity Provider* der jeweiligen Heimateinrichtung weiter. Der IdP wiederum authentifiziert den Nutzer und leitet nachfolgend die Authentifizierungsantwort (*Response*) und ihn an den SP zurück. Abschließend überprüft der SP die erhaltene Antwort des IdP (Autorisierung) und schickt bei erfolgreicher Bescheinigung den ursprünglich angeforderten Inhalt an den Nutzer zurück. Wichtig zu erwähnen ist, dass aus Datenschutz- bzw. Datensparsamkeitsaspekten die Übermittlung des Passwortes an Dienste nicht stattfindet; es wird hier nur über den Erfolg der Authentifizierung (*StatusCode*) Auskunft gegeben. Der Austausch der signierten Metadaten erfolgt im betrachteten

Szenario einer AAI asynchron zum eigentlichen Betrieb [Pem18b, Can05, vgl. S. 9ff]. Eine vollständige SAML-Kommunikation – sprich der *AuthnRequest* und der dazugehörige *Response* – ist im Anhang unter Abschnitt A.1 vorzufinden.

Hauptsächlich wird in der DFN-AAI der Zugriff auf Webinhalte via *Single Sign-on* gewährt und somit die Zusammenarbeit zwischen Hochschulen und Forschungseinrichtungen vereinfacht. Konkrete Dienste werden dabei wesentlich von BibliotheksnutzerInnen, Studierenden und Lehrpersonal sowie von Forschenden in Anspruch genommen (s. Abbildung A.1). Für den Identitätsaustausch und die -feststellung wird in diesem Kontext, wie auch schon in der Abbildung 3.4 dargestellt, das immer noch weitverbreitete SAML-*Framework* eingesetzt. Die Bereitstellung von nicht webfähigen Inhalten soll in der näheren Zukunft durch die Verwendung von SAML-Alternativen wie *OpenID Connect* Einzug erlangen [Pem18b, vgl. S. 51].

In den letzten acht Jahren ist die Teilnahme an der DFN-AAI-Föderation, wie in der Abbildung 3.5 zu sehen, stetig gestiegen (Stand: 21. August 2018); zahlenmäßig sind natürlich mehr Diensteanbieter als Identitätsanbieter in der Föderation vorhanden. Insgesamt gibt es 267 verschiedene IdP und 465 SP; der Anteil an lokalen SP von 884 Stück hat sich dabei besonders stark entwickelt. Dies weist auf eine Etablierung des SSO innerhalb einer Organisation hin. Der DFN-Verein mit seiner Föderation ist zudem in einer föderationsübergreifenden AAI integriert. Sie heißt *eduGAIN* und ermöglicht es weltweite Identitätsverteilungen digitaler Identitäten vorzunehmen. Seit 2011 wird diese AAI produktiv von GÉANT auf europäischer Ebene betrieben. Ihre Hauptaufgabe ist es bedingt durch die höhere Organisationsebene die IdP- und SP-Metadaten aus den teilnehmenden Föderationen zu vereinigen und bekannt zu geben (*Upstream Metadata*). Die Föderationen selbst stellen daraufhin die neuen Metadaten ihren Mitgliedern zentral zur Verfügung (*Downstream Metadata*). 53 unterschiedliche Föderationen sind bereits Mitglied in *eduGAIN* und farblich / geografisch in der Abbildung 3.6 dargestellt (Stand: 21. August 2018). Ganz interessant ist auch die Beteiligung je Föderation in dieser weltweiten Zusammenarbeit (s. Abbildung A.2) [Pem18b, DT18, vgl. S. 35ff].

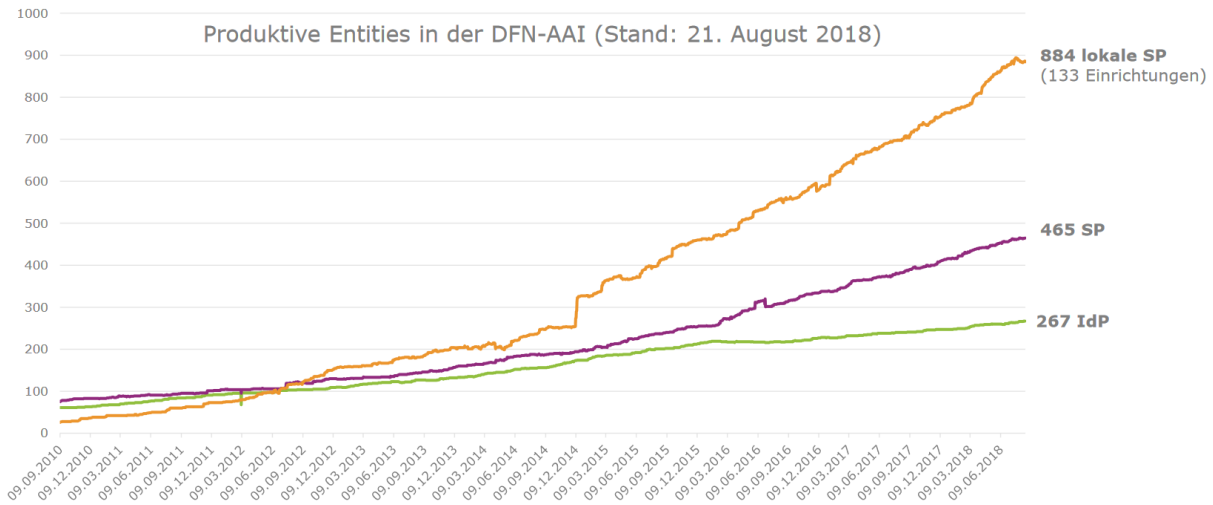

Abbildung 3.5: DFN-AAI – Produktive Entities [Pem18b, S. 35]

In allen Föderationen sind die Metadaten von besonderer Wichtigkeit und stellen sozusagen das technische Rückgrat dar. „Nur wenn auf beiden Seiten (IdP / AA [*Attribute Authority*], SP) die Metadaten des jeweiligen Kommunikationspartners bekannt sind (und ihnen vertraut wird), funktioniert die Kommunikation“ [Pem18b, S. 27]. Der DFN-Verein übernimmt die Aggregation und die Verwaltung der Metadaten. Grob betrachtet ist die Teilnahme in zwei getrennten Föderationen möglich: Test- und Produktivföderation. Auch die Möglichkeit, lokale Metadaten

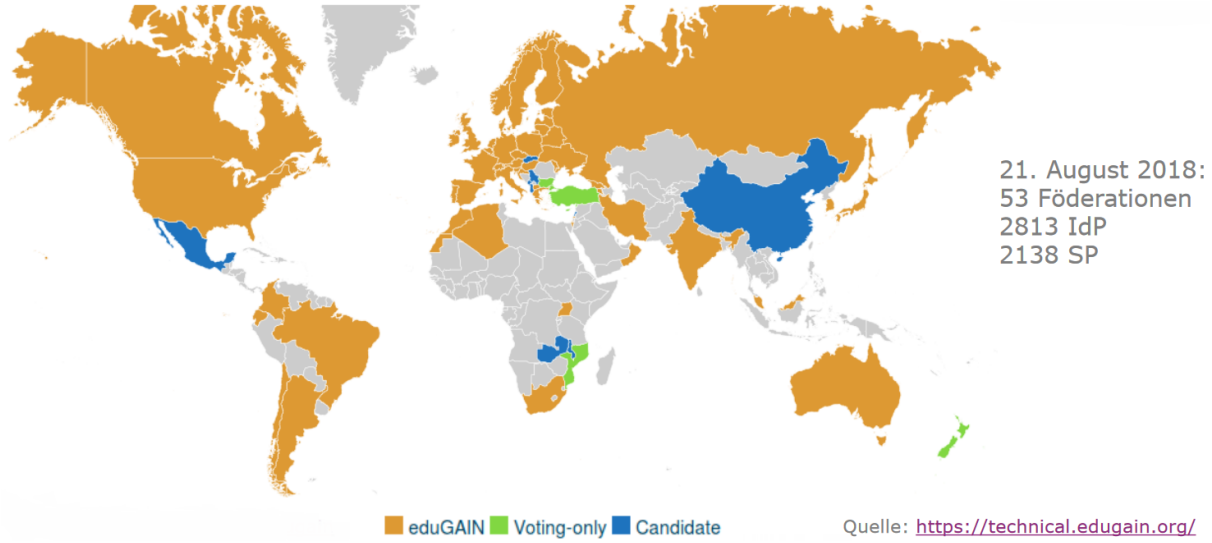

Abbildung 3.6: *eduGAIN* – beteiligte Föderationen [Pem18b, S. 37]

beim DFN-Verein zentral zu positionieren, wird geboten. Das jeweilige Vertrauen beruht auf fest definierten Verlässlichkeitsklassen je Kategorie. Welche Maßnahmen zu treffen sind, werden in der Tabelle 3.3 aufgezeigt [Pem18b, DT18, vgl. S. 27ff].

Verlässlichkeitsklasse	**Identifizierung durch Heimateinrichtung**	**Verfahren zum Ausweis einer Identität**	**Datenhaltung und Prozesse zur Pflege der Identitäten**
Test	Verfahren freigestellt.	Verfahren freigestellt.	Verfahren freigestellt.
Basic	Rückantwort von eindeutiger Adresse (E-Mail, Tel.-Nr., Postanschrift, etc.).	Anhand eindeutig zuzuordnender digitalen Adresse.	Verpflichtung bzgl. Aktualität innerhalb von 3 Monaten.
Advanced	Pers. Vorsprechen gegenüber Vertrauensinstanz unter Vorlage amtlicher Dokumente (alternativ: Post-Ident, eID / nPA). Die an den Hochschulen etablierten Einschreibungs- und Einstellungsprozesse werden als gleichwertig akzeptiert.	Pers. Account bzw. digitales Zertifikat (sichere Vergaberichtlinie).	Verpflichtung bzgl. Aktualität innerhalb von 2 Wochen.

Tabelle 3.3: Verlässlichkeitsklassen in der DFN-AAI [Pem18b, vgl. S. 31]

Damit die Verbindung zwischen SP und dem jeweiligen IdP der Heimateinrichtung komfortabler hergestellt werden kann, wird ein sogenannter *Discovery Service* (DS) bzw. Lokalisierungsdienst eingesetzt. Er ist auch bekannt als WAYF („*Where Are You From*“), der in drei verschiedenen Ausprägungen vorkommen kann. Die erste Variante ist ein zentraler DS, der beispielsweise auch vom DFN-Verein betrieben wird. Vom SP erfolgt eine Weiterleitung auf diesen Dienst und der Nutzer wählt hier seine Heimateinrichtung aus. Die zweite Möglichkeit bietet ein *Embedded Discovery Service* (EDS); dieser wird auch in der Software *Unity IdM* eingesetzt. Hierdurch wird

eine benutzerfreundliche Wahl des Identitätsanbieters am SP selbst vollzogen; die bereitgestellte Liste an Einrichtungen kann bereits gefiltert sein. Als Letztes gibt es noch die *WAYFless URLs*, die schon beim Zugriff auf den SP eine Authentifizierungsanforderung zu einem bestimmten IdP auslösen. Sie sind fest kodiert und daher entfällt die Einrichtungswahl für den Nutzer [Pem18b, Can05, vgl. S. 45ff].

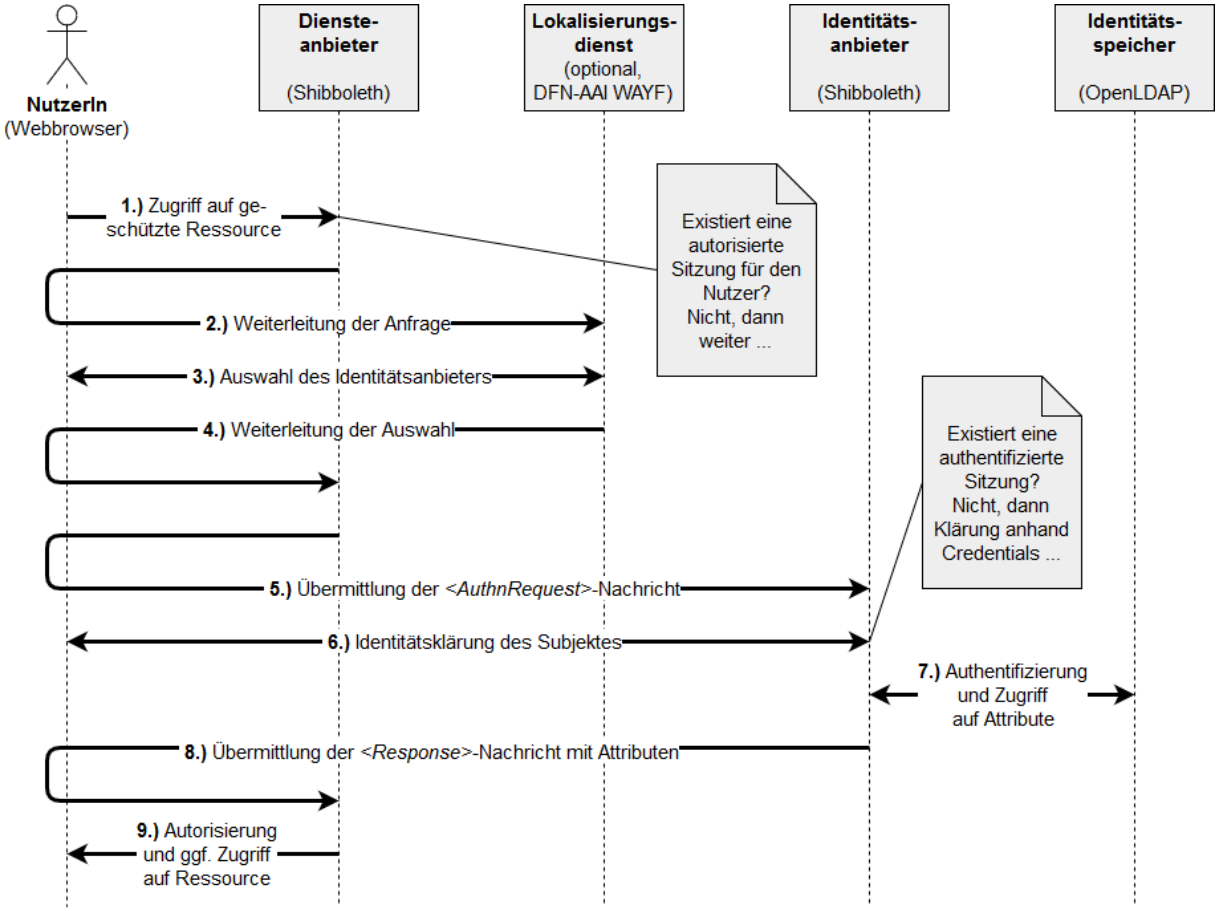

Abbildung 3.7: SAML 2.0 – *Single Sign-on* [HCH+05, WC08, vgl. S. 15, S. 8]

Eine komplette Übersicht des *Single Sign-on*-Ablaufes, wie der Zugriff eines Nutzers via SAML auf einen föderativen Dienst in der DFN-AAI abläuft, ist in der Abbildung 3.7 als abstraktes Sequenzdiagramm abgebildet. Hierbei wird nur das SP-getriebene SSO betrachtet. Innerhalb eines einzelnen Schrittes kann es eine oder mehrere Nachrichtenübermittlungen geben. Der Beginn der Interaktion geht vom Nutzer aus, der mit seinem Webbrowser auf einen Diensteanbieter im DFN-AAI-Verbund zugreift (1.). Handelt es sich um eine geschützte Ressource, so wird eine Authentifizierung / Autorisierung des Nutzers benötigt. Durch eine Weiterleitung zu einem Lokalisierungsdienst bzw. DS (2.) und die anschließende Auswahl des zuständigen IdPs (3.), wird die Heimateinrichtung des Nutzers ermittelt (4.) und die Authentifizierungsanforderung (*AuthnRequest*) kann im Anschluss an den jeweiligen Identitätsanbieter übermittelt werden (5.). Nachdem sich der Nutzer an dem *Identity Provider* authentifiziert hat – beispielsweise mit *Credentials* seine Identität bestätigt hat – (6.), werden alle zur Entität gehörenden und freigegebenen Attribute aus dem Identitätsspeicher geholt (7.) und letztendlich durch eine *Response*-Nachricht verschlüsselt sowie signiert über den Webbrowser des Nutzers wieder dem anfragenden *Service Provider* übermittelt (8.). Dieser prüft die erhaltende *Assertion* und führt eine Autorisierung des Nutzers durch, die ggf. den Zugriff auf den Dienst gestattet (9.). In einem SSO-Kontext wird die Aufforderung zur Eingabe der *Credentials* des Nutzers nicht gefordert, wenn bereits eine aktive und authentifizierte Sitzung am IdP registriert ist. Natürlich können

auch Abläufe realisiert werden, bei denen sich der Nutzer bereits im ersten Schritt am IdP seiner Heimateinrichtung authentifizieren muss. [HCH+05, WC08, RHP+08, vgl. S. 14ff, S. 7f., S. 26ff].

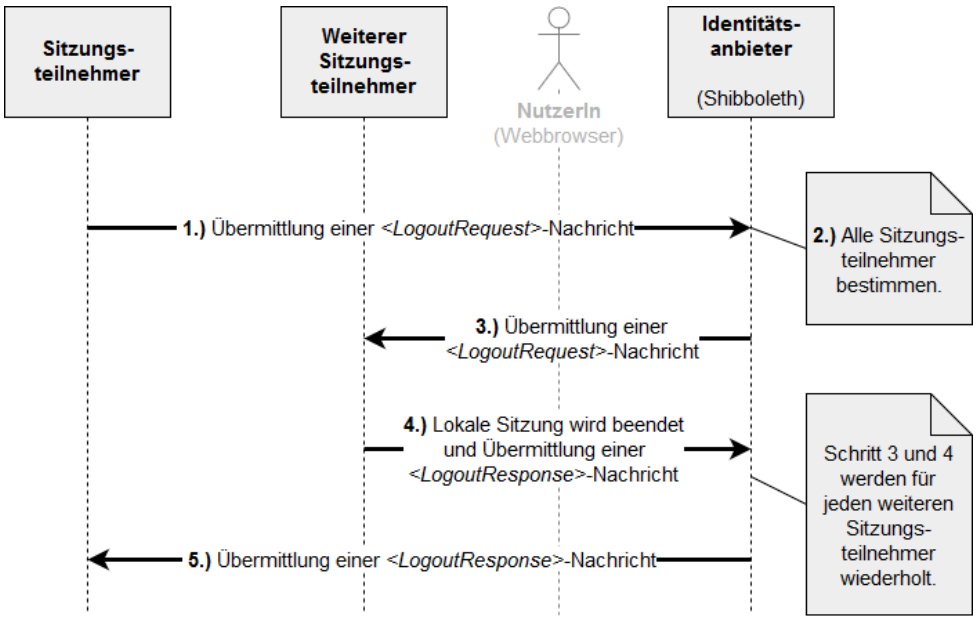

Abbildung 3.8: SAML 2.0 – *Single Log-out* [HCH+05, vgl. S. 33]

In der Abbildung 3.8 wird ein abstraktes Sequenzdiagramm dargestellt, welches den *Single Log-out*-Ablauf demonstrieren soll. Der ausgegraute Nutzer soll versinnbildlichen, dass über ihn der Nachrichtenaustausch vollzogen werden kann, oder aber die Systemakteure (SP, IdP) sich direkt austauschen. Auch hier können innerhalb eines einzelnen Schrittes ein oder mehrere Nachrichtenübermittlungen stattfinden. Zu Beginn wird eine *LogoutRequest*-Nachricht an den Identitätsanbieter übermittelt (1.). Danach werden alle Sitzungsteilnehmer am IdP bestimmt (2.), der als Sitzungsautorität fungiert. Im nächsten Schritt wird eine *LogoutRequest*-Nachricht an einen weiteren Sitzungsteilnehmer geschickt (3.). Daraufhin beendet jener seine lokale Sitzung und übermittelt eine Bestätigung (*LogoutResponse*) an den IdP zurück (4.). Die beiden Schritte zwei und drei werden so lange wiederholt, bis keine Sitzungsteilnehmer mehr übrig sind. Im letzten Schritt wird die ausstehende *LogoutResponse*-Nachricht an den zu Beginn auslösenden Sitzungsteilnehmer überstellt (5.). Der SLO kann übrigens auch vom Identitätsanbieter / Administrator selbst gestartet werden, indem er in einem Schritt an alle ein *LogoutResponse* versendet. Außerdem ist es wichtig, dass die SLO-Funktionalität bei allen involvierten Diensteanbietern konfiguriert sein muss; die Einmalabmeldung wäre sonst nicht vollständig realisierbar [HCH+05, RHP+08, vgl. S. 33ff, S. 37f.].

3.3 Shibboleth

In der DFN-AAI wird u. a. die *Open Source*-Software *Shibboleth* als IdP / SP im SAML-Kontext eingesetzt – so ist es auch beim AWI der Fall. Daher wird *Shibboleth* häufig auch als Synonym für SAML-basiertes Web-SSO verwendet. *Shibboleth* wird in Zusammenarbeit vom *Internet2*-Projekt und OASIS seit 2000 vorangetrieben. Der IdP ist mittlerweile in der dritten Version verfügbar. Die Bezeichnung geht übrigens zurück auf die Bibel (Richter 12, 5-6) [Pem18b, ST18b, vgl. S. 4]. Beim *Alfred-Wegener-Institut* sind zwei Identitätsanbieter im Einsatz:

- Produktivföderation (Advanced) + *eduGAIN*:
  `https://shib-idp.awi.de/idp/shibboleth`

- Testföderation: `https://shib-idp2.awi.de/idp/shibboleth`

Generell werden durch einen IdP Attribute bereitgestellt, die die notwendigen Informationen einer Entität beinhalten und letztendlich die Nutzung eines SSO-Dienstes ermöglichen. Die übermittelten Attribute dienen dabei nicht nur der eindeutigen Identifizierung des Nutzers, sondern gewähren auch die Durchführung einer Autorisierung am SP. Die zulässigen Werte und deren Bedeutungen werden durch allgemein gültige Schemata festgelegt. In der Föderation haben sich dabei *eduPerson* [IT16] und SCHAC (*SCHema for ACademia*, [Fla18]) sowie weitere LDAP-Attributdefinitionen aus *inetOrgPerson* [Smi00, RFC 2798] durchgesetzt [Pem18a, vgl. S. 3ff].

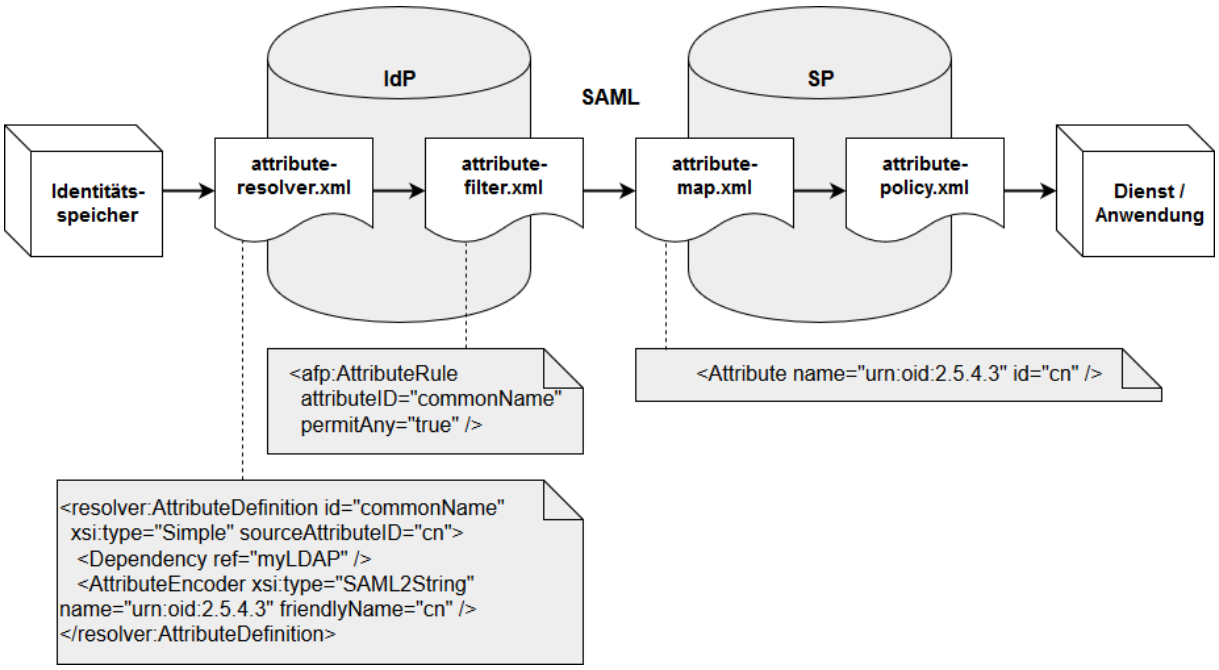

Abbildung 3.9: Attribut-Management in *Shibboleth* [Pem18a, vgl. S. 4]

Anhand der Abbildung 3.9 werden die wichtigsten Konfigurationsdateien von *Shibboleth* sowohl auf der IdP- als auch der SP-Seite bildhaft dargestellt. So ähnlich findet das Zusammenspiel zwischen den Akteuren auch bei anderen Software-Implementierungen, wie beispielsweise *Unity IdM*, statt, die das SAML-*Framework* verwenden. Bevor die in der Abbildung 3.9 aufgezeigte „Pipeline-Verarbeitung“ startet, muss eine Authentifizierungsanforderung (*AuthnRequest*) vom Diensteanbieter eingegangen sein. Des Weiteren muss gewährleistet sein, dass sich die beiden Akteure IdP und SP gegenseitig vertrauen, sonst würde der anschließende SAML-*Response* nicht erstellt werden. Das Vertrauen in einer Föderation beruht, wie bereits weiter oben zu sehen, auf das Vorhandensein signierter Metadaten, die vom Betreiber einer Föderation gestellt werden. Der IdP ist für die Bereitstellung zusätzlicher Attribute eines Subjektes aus einem Identitätsspeicher zuständig. Durch die „*attribute-resolver.xml*“ (s. Listing A.3) werden Attribute geholt, ggf. bearbeitet und für den SP in die richtige Form, beispielsweise in einen *SAML2String*, umgewandelt. Es können hier auch gänzlich neue Attribute erschaffen werden. In der „*attribute-filter.xml*“ (s. Listing A.4) wird anschließend bestimmt, welche Attribute einem bestimmten Diensteanbieter – genauer gesagt dem *Assertion Consumer Service* (ACS) – anhand der *EntityId* übermittelt werden. Der Nutzer hat allerdings immer noch die Möglichkeit über die Weitergabe seiner personenbezogenen Daten in jeder SAML-*Assertion* zu entscheiden (*User Consent*) [Pem18a, Can05, ST18a, vgl. S. 7ff, S. 4ff].

Auf der Seite des *Service Providers* empfängt der ACS die verschlüsselte Bescheinigung des IdPs und extrahiert die Attribute mit Hilfe einer eindeutigen URI (*Uniform Resource Identifier*). Dies

geschieht in der „*attribute-map.xml*“; die Attribute werden zudem auf interne Variablen abgebildet. Durch die im nächsten Schritt agierende „*attribute-policy.xml*“ werden die empfangenden Werte gefiltert, wodurch letztendlich eine Autorisierung durchgeführt wird. Abschließend werden dem Dienst bzw. der Anwendung die Attribute bereitgestellt und eine aktive Sitzung im Webbrowser gehalten [Pem18a, Can05, ST18a, vgl. S. 18ff, S. 4ff].

3.4 Unity IdM

Unity IdM ist eine vielseitige und in *Java* geschriebene Identitätsmanagementlösung für die unterschiedlichsten Anwendungsfälle in einer Organisation oder speziell auch in einem Föderationsumfeld. Das Ziel der Anwendung ist es, dass viele verschiedene Authentifizierungsprotokolle unterstützt werden und eine Integration in einem bestehenden IdM möglichst einfach zu realisieren ist. Es besteht weiterhin die Möglichkeit, die Verwaltung von Anmeldeinformationen und Attributen an einen externen Dienstanbieter auszulagern. Hierfür werden auch mehrere *Upstream*-Protokolle unterstützt. *Unity IdM* wird demzufolge zu einer Brücke – Protokollübersetzung – und zu einem Hub – einzelner Dienst, der verschiedene IdM-Systeme aggregiert [Ben13, UT18b, vgl. S. 8].

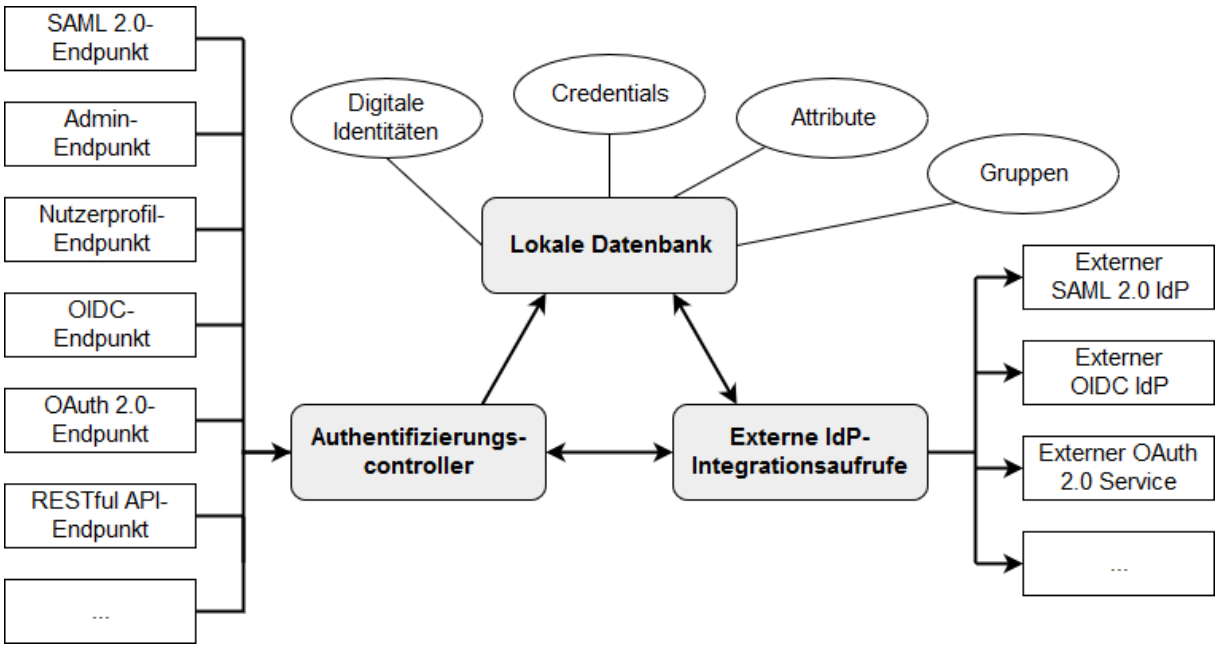

Abbildung 3.10: Systemaufbau von *Unity IdM* [UT18b]

Die Abbildung 3.10 zeigt einen abstrakten Überblick der inneren Bestandteile von *Unity IdM*. Mit Hilfe eines Authentifizierungscontrollers gibt es die Möglichkeit an den unterschiedlichsten Authentifizierungsendpunkten die Anmeldung einer Entität zu gewähren. Im Projekt sind der Admin-, der Nutzerprofil- und der *RESTful* API-Endpunkt sowie ein SAML-Authentifizierungscontroller involviert. Durch die Einbindung des Authentifizierungscontrollers im Nutzerprofil-Endpunkt werden digitale Identitäten (u. a. *Persistent identity*, *Username identity*) in der lokalen Datenbank erzeugt und abgelegt. Des Weiteren werden hier zur Identität dazugehörige Attribute, eine Gruppenzugehörigkeit und ein lokales Passwort (*Password Credentials*) gespeichert. Der Admin-Endpunkt bietet die Möglichkeit, über eine Weboberfläche Konfigurationen an der Software durchzuführen. Gespeicherte Profilinformationen können dagegen im Nutzerprofil-Endpunkt verwaltet werden. Der *RESTful* API-Endpunkt ermöglicht es sämtliche Modifikationen an Entitäten via HTTP vorzunehmen. Eine weitere Hauptkomponente gestattet es auch externe IdP-Integrationsaufrufe

bei Bedarf durchzuführen; dies wird durch den SAML-Authentifizierungscontrollers beansprucht, um mit den jeweiligen Heimat-IdPs in Kontakt zu treten [UT16, UT18b].

Für das Szenario ergibt sich also, dass *Unity IdM* im SAML-Umfeld vorwiegend als SP betrieben wird, um einen Zugang für föderative Entitäten zu schaffen. Da *Unity IdM* als Proxy agiert, nimmt er natürlich auch die Rolle eines Identitätsanbieters ein; daher existiert auch die Bezeichnung *SP-IdP-Proxy*. Verwendet werden die übermittelten Attribute des Nutzers aus dem IdM der Heimateinrichtung, um ein lokales Konto anzulegen. Zusätzlich werden ein Passwort und Gruppeninformationen gespeichert. *Unity IdM* liefert eine große Menge an Schnittstellen für das Identitätsmanagement (s. Abbildung 3.10). Der Authentifizierungsdienst ist folglich sehr flexibel und für die Zukunft ausgerichtet, falls verwendete Protokolle im Szenario gewechselt werden sollten. Der Verzicht auf SAML und die zukünftige Einführung von *OpenID Connect* in der DFN-AAI sind hier zu nennen [Pem18b, vgl. S. 51]. Leider besteht nicht die Möglichkeit aus *Unity IdM* selbst, einen vorhandenen LDAP-Server mit den föderativen Entitäten zu synchronisieren. Hierzu muss ein eigener Dienst entwickelt werden (s. Kapitel 4).

3.5 OpenLDAP

Es gibt einige Implementierungen des LDAP [Ser06, RFC 4511] auf dem Softwaremarkt. Im *Open Source*-Bereich spielt dabei die Referenzimplementierung *OpenLDAP* von der *OpenLDAP Foundation* eine wichtige Rolle und wird daher auch in dieser Bachelorarbeit eingesetzt. Die Software, die in der Programmiersprache *C* geschrieben ist, enthält einen eigenständigen LDAP-Server bzw. -Daemon – dieser wird *slapd* genannt, diverse Bibliotheken und einige weitere nützliche Werkzeuge im Umgang mit dem offenen Verzeichnisprotokoll [OT18b]. Wie LDAP genau funktioniert und welche Chancen das Protokoll bietet, wurde im Unterabschnitt 3.1.5 aufgezeigt. Ferner muss die Möglichkeit bestehen, lokale Entitäten aus dem *Unity IdM* der am AWI agierenden Software *VMware vRealize Automation* zugänglich zu machen. Hierzu wird ein eigener Identitätsspeicher in Ausprägung eines *OpenLDAP*-Verzeichnisdienstes implementiert. Die Synchronisation der Entitäten zwischen dem *Unity IdM* und dem Identitätsspeicher muss durch Zuhilfenahme eines selbst geschriebenen Dienstes erfolgen. Diese konkrete Umsetzung wird im Kapitel 4 thematisiert.

3.6 VMware vRealize Automation

Eine Instanz der lokalen Software *VMware vRealize Automation* (vRA), die im Umfeld der Föderation eingebettet werden soll, muss natürlich auch vorhanden sein. Die kommerzielle Software des US-amerikanischen Unternehmens *VMware* bietet Nutzern eine Plattform, auf der IT-Dienste angefordert und Cloud- bzw. IT-Ressourcen durch Administratoren verwaltet werden können. Bei der Bereitstellung von diesen *On-Demand*-Diensten über die zentrale Plattform (*Marketplace*) an den Endanwender wird die Einhaltung der Unternehmensrichtlinien sichergestellt. Auch spielt eine zugrunde liegende heterogene Infrastruktur keine allzu große Rolle, da ein gemeinsamer Servicekatalog die Anforderungen an alle beteiligten Ressourcen in den Geschäftsprozessen überschaubar hält [VT18b]. Im praktischen Anteil dieser Arbeit soll die Anmeldung der Nutzer, die aus der DFN-AAI-Föderation stammen können, an dieser Plattform durch LDAP realisiert werden. Dazu wird der Zugriff auf den Verzeichnisdienst in Ausprägung eines *OpenLDAP*-Servers in *vRealize Automation* konfiguriert.

4 Praktische Umsetzung

Dieses Kapitel befasst sich mit dem praktischen Anteil der Bachelorarbeit. In den einzelnen Abschnitten wird auf die Vorarbeiten an den beteiligten Systemen (s. Abschnitt 4.1) und die Implementierung der Benutzer-Entitätensynchronisation (s. Abschnitt 4.2) eingegangen. Das zu erwartende Ergebnis der Umsetzung wird in der Abbildung 4.1 deutlich. Hier wird einmal ein Gesamtüberblick des betrachteten Szenarios grafisch dargestellt.

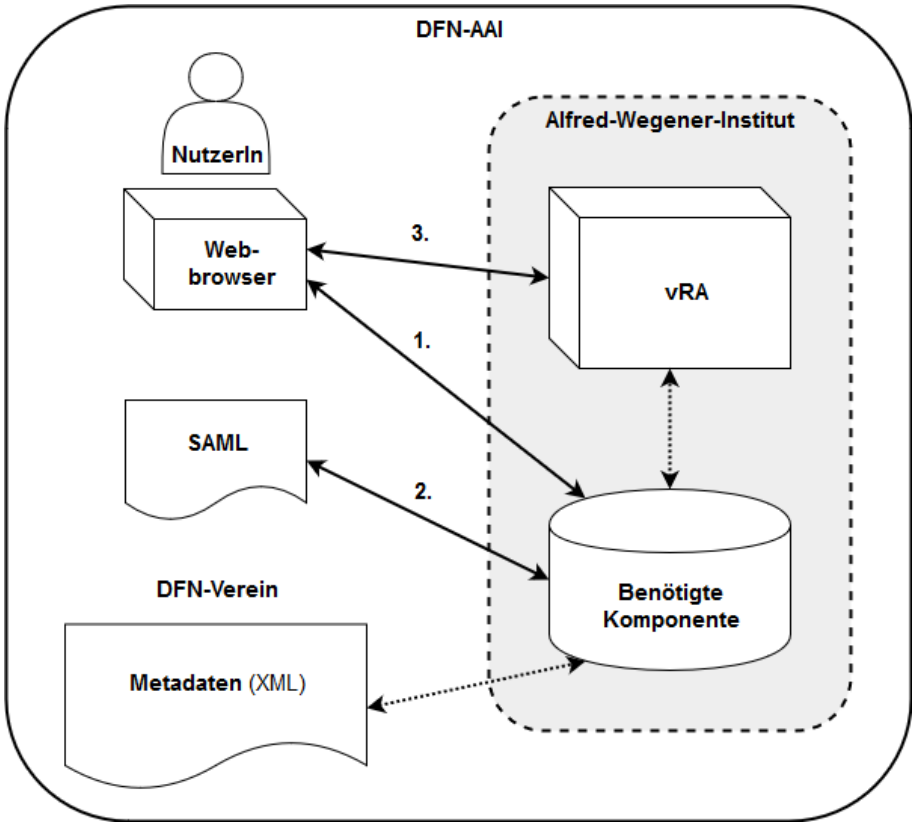

Abbildung 4.1: Kontextdiagramm des betrachteten Szenarios

Ein NutzerIn aus der DFN-AAI kann sich an der benötigten Komponente mit Hilfe eines Webbrowsers anmelden (1.). Zuvor wird der jeweilige IdP der Heimateinrichtung ausgewählt, damit die Identitätsverteilung durch das SAML-*Framework* gestartet werden kann (2.). Nach erfolgreicher Anmeldung bzw. positiver Authentifizierung durch den Abgleich von *Credentials* am Heimat-IdP wird ein lokales Benutzerkonto in der zu erstellenden Komponente erschaffen. Die dafür benötigten Attribute des Subjektes werden der SAML-*Assertion* entnommen. Zusätzlich werden ein lokales Passwort und Gruppeninformationen bei der Erstellung der neuen digitalen Identität hinzugefügt. Die Gruppeninformationen sollen später von berechtigten Gruppenleitern über eine Webschnittstelle der Komponente verwaltet werden. Sobald ein lokales Benutzerkonto durch die ersten beiden Schritte (1., 2.) angelegt wurde, kann schließlich der Zugriff auf die eingebettete Ressource vRa (*VMware vRealize Automation*) durch den NutzerIn vollzogen werden (3.). Die verfügbaren lokalen Benutzerkonten werden durch eine LDAP-Schnittstelle der benötigten Komponente lokal zur Verfügung gestellt; die Synchronisation der Konten in vRA

erfolgt dabei asynchron. Für das notwendige Vertrauensverhältnis zwischen Identitätsanbieter und Diensteanbieter sorgt der DFN-Verein, der signierte Metadaten über die beteiligten Systeme / Entitäten ausstellt und an einer zentralen Stelle bereitstellt. Der Austausch erfolgt dabei zu einem früheren Zeitpunkt und asynchron zum eigentlich betrachteten Anwendungsfall. Die geschützte Ressource kann nun in der Föderation organisationsübergreifend verwendet werden.

Die für das Szenario benötigte Komponente aus dem Kotextdiagramm der Abbildung 4.1 wird in der Abbildung 4.2 verdeutlicht. Mit Hilfe der Bausteinsicht können sowohl die fundamentalen Teilbausteine als auch die zu berücksichtigenden inneren wie äußeren Schnittstellen aufgezeigt werden.

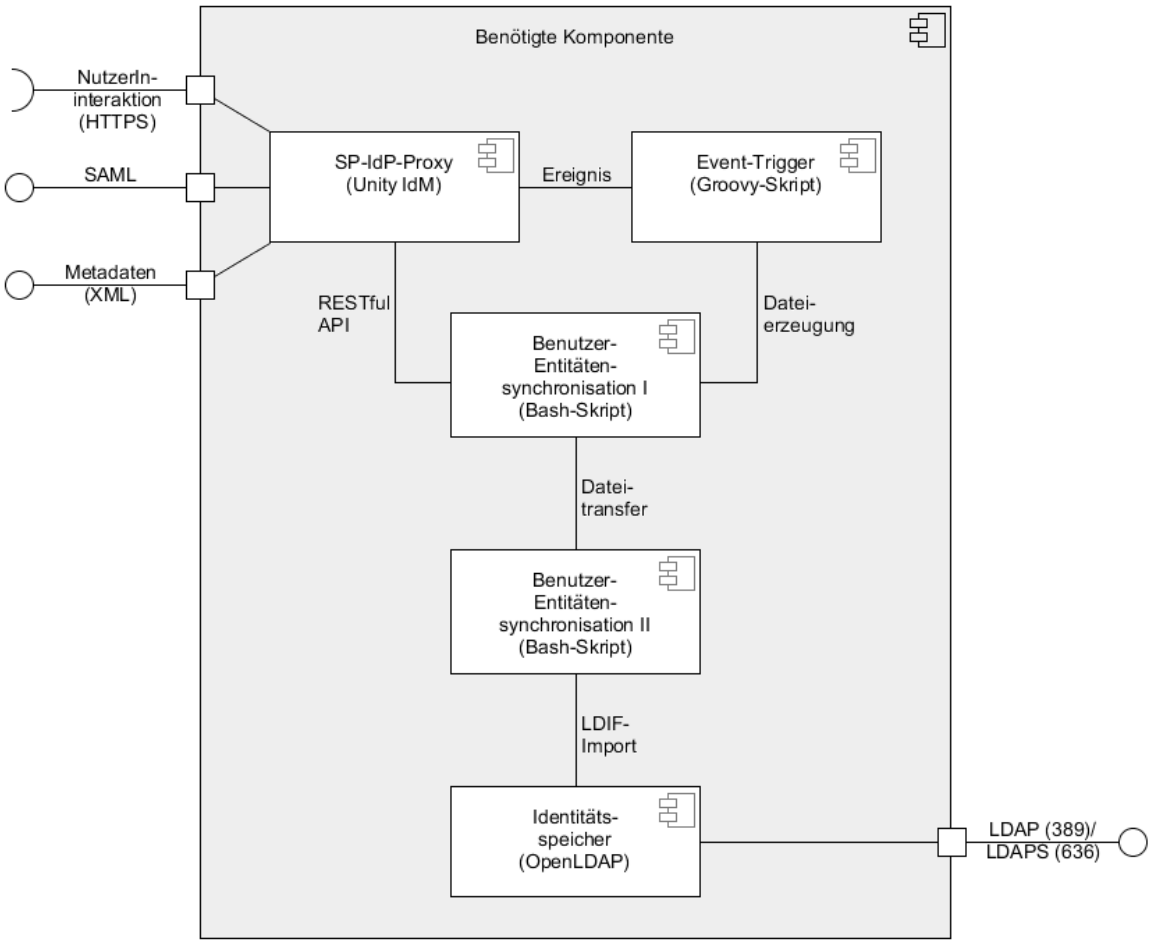

Abbildung 4.2: Bausteinsicht der benötigten Komponente

Die Komponente besteht im Wesentlichen aus einem *SP-IdP-Proxy*, einer Benutzer-Entitätensynchronisation und einem angeschlossenen Identitätsspeicher in Form des Verzeichnisdienstes *OpenLDAP*. Für den Proxy kommt die Software *Unity IdM* zum Einsatz, die einen SAML-Authentifizierungscontroller und einen Nutzerprofil-Endpunkt (SP) mit Weboberfläche zur Interaktion mit dem Nutzer bereitstellt. *Unity IdM* bedient dabei alle außerhalb des AWIs auftretenden externen Schnittstellen. Die Benutzer-Entitätensynchronisation wird durch ein selbst geschriebenes *Bash*-Skript realisiert, welches in zwei Teile aufgeteilt ist. Es baut auf die von *Unity IdM* bereitgestellte *RESTful API* auf, durch die die benötigten Attribute der Föderationsmitglieder entnommen werden können. Auslöser einer anstehenden Synchronisation sind Ereignisse auf der Seite des Proxys, die wiederum Dateien in ein bestimmtes Verzeichnis für ein wartendes Skript erzeugen. Der letzte Teilbaustein beinhaltet den Verzeichnisdienst *OpenLDAP*. Er ist

für die lokale Speicherung der digitalen Identitäten aus der Föderation zuständig und stellt eine LDAP-Schnittstelle in der Organisation bereit. Der Austausch zwischen dem *Bash*-Skripten und dem LDAP-Dienst erfolgt durch Dateien im LDIF-Format. Mit diesem standardisiertem Format können über die Kommandozeile sämtliche Manipulationen im Verzeichnisdienst erfolgen. Der Aufbau der benötigten Komponente wurde möglichst modular und abstrakt gehalten, um zukünftigen strukturellen und sicherheitsrelevanten Veränderungen entgegenzuwirken.

4.1 Vorarbeiten

Damit das eigentliche Vorhaben der Nutzbarmachung einer lokalen Ressource in der DFN-AAI-Föderation verwirklicht werden kann, müssen diverse Vorarbeiten an den beteiligten Systemen geleistet werden. Dies beinhaltet vorwiegend Installations- und Konfigurationsmaßnahmen, aber auch die Bereitstellung von konkreten Systemen in Ausprägung virtueller Maschinen (VMs). Tabellen mit den VM-Spezifikationen liefern in den jeweiligen Unterabschnitten einen Überblick der verwendeten IT-Ressourcen der einzelnen Systeme. Alle VMs sind mit dem Betriebssystem *Ubuntu Linux* in der 64-bit-Architektur und ohne grafische Oberfläche ausgestattet.

4.1.1 Shibboleth

Die SSO-Software *Shibboleth* spielt im Szenario die Rolle des Identitätsanbieters einer entsprechenden Heimateinrichtung in der DFN-AAI. Konkret ist die Testinstanz „*shib-idp2.awi.de*“ mit den VM-Eigenschaften, die der Tabelle 4.1 zu entnehmen sind, involviert. Für die Bachelorarbeit wurde ein vorhandener *Shibboleth* IdP in der Version 3.3.3 verwendet, welcher auch bereits beim DFN-Verein in der Metadatenverwaltung mit der Verlässlichkeitsklasse „*Test*“ registriert war. Der Autor hat diesen Test-IdP vor der Bachelorarbeit im Tätigkeitsumfeld als studentische Hilfskraft eingerichtet. Die Software ist in *Java* geschrieben und wird durch diverse XML-Dateien konfiguriert bzw. administriert. Das Web-SSO wird dabei durch einen *Apache2* HTTP-Server und einem *Apache Tomcat* bereitgestellt. Letztere Software ermöglicht es in *Java* geschriebene Webanwendungen wie den *Shibboleth* IdP auf *Servlet*- und JSP-Basis (*JavaServer Pages*) auszuführen.

Hostname	*shib-idp2.awi.de*
OS	*Ubuntu Linux* (64-bit) 16.04
CPU	1
RAM	4GB
Speicher	20GB + 10GB („/“)
Software	*Shibboleth IdP* 3.3.3, *Java OpenJDK* 8, *Apache Tomcat* 8, *Apache2* HTTP-Server

Tabelle 4.1: VM-Spezifikation – *Shibboleth*

Hauptsächlich wurden als Vorarbeit die SAML-Metadaten der *Unity IdM*-Instanz durch eine lokale Metadatendatei eingepflegt und Filterrichtlinien in Bezug auf die freizugebenden Attribute erstellt. *Unity IdM* agiert im Szenario nämlich als Diensteanbieter (SP) und daher muss ein Vertrauensverhältnis mit dem IdP bestehen, welches durch die abgelegten Metadaten realisiert wird. Natürlich hätten die Metadaten des SP auch beim DFN-Verein registriert werden können, dann wäre das manuelle Hinzufügen der Metadaten auf der IdP-Seite obsolet. Da es sich um ein Testumfeld handelt, wurde der Metadatenaustausch lokal in der Organisation (AWI) gelassen. Alle Metadatenanbieter werden in der Datei „*metadata-providers.xml*“ verwaltet; die Anlaufstellen

für die signierten Metadaten des DFN-Vereins sind hier auch vorzufinden. Im Listing 4.1 ist der vorgenommene Eintrag abgedruckt [ST18a, Pem18a].

```
<MetadataProvider id="LocalMetadata" xsi:type="FilesystemMetadataProvider"
    metadataFile="%{idp.home}/metadata/local-unitysrv1.xml"/>
```

Listing 4.1: Eintragung der lokalen Metadatendatei im *Shibboleth* IdP

Die Filtereinstellungen beim *Shibboleth* IdP werden zentral in der „*attribute-filter.xml*“ zusammengefasst. Für die praktische Umsetzung wurde die Übermittlung von fünf Attributen an den Diensteanbieter „*unitysrv1.awi.de*“ zugelassen. U. a. dürfen der Vorname, Nachname, gebräuchliche Name, dargestellter Name und die E-Mail-Adresse des jeweiligen Subjektes während einer legitimierten SAML-Kommunikation aus dem Identitätsspeicher ausgelesen werden. Die sogenannte „*AttributeFilterPolicy*“ ist dem Listing 4.2 zu entnehmen. Die komplette Filterdatei und auch die notwendige Übersetzungsdatei „*attribute-resolver.xml*“, die LDAP-Attribute auf SAML-Attribute projiziert, sind im Anhang abgebildet (s. Listing A.3, Listing A.4) [ST18a, Pem18a].

```
<AttributeFilterPolicy id="unity">
  <PolicyRequirementRule xsi:type="OR">
    <Rule xsi:type="Requester" value="https://unitysrv1.awi.de" />
  </PolicyRequirementRule>
  <AttributeRule attributeID="mail" permitAny="true" />
  <AttributeRule attributeID="surname" permitAny="true" />
  <AttributeRule attributeID="givenName" permitAny="true" />
  <AttributeRule attributeID="commonName" permitAny="true" />
  <AttributeRule attributeID="displayName" permitAny="true" />
</AttributeFilterPolicy>
```

Listing 4.2: Angewandte Filterrichtlinie im *Shibboleth* IdP

Selbstverständlich hat das involvierte Subjekt immer die Möglichkeit, der Übermittlung bestimmter Attribute zu widersprechen. Dazu wird nach der erfolgreichen Authentifizierung ein bestimmtes Dialogfenster (s. Abbildung A.4) angezeigt, in dem die Attributfreigabe durch Setzen der jeweiligen Haken beeinflusst werden kann. Die Anmeldemaske des *Shibboleth* IdP ist auch im Anhang in der Abbildung A.3 zu sehen; hier werden die *Credentials* des Nutzers für die Authentizitätsklärung abgefragt [ST18a, Pem18a].

4.1.2 Unity IdM

Unity IdM wird als sogenannter *SP-IdP-Proxy* eingesetzt. D. h. die Software agiert als Mittler zwischen einer föderativen Entität und der neu zu erschaffenden lokalen Identität, welche beim AWI verbleibt. Sie stellt sowohl einen SP als auch einen IdP bereit, da sie die Anmeldung eines föderativen Subjektes ermöglicht, daraufhin ein neues Nutzerkonto erschafft und dieses letztendlich wieder als Identitätsanbieter lokal in der Organisation bereitstellt. *Unity IdM* ist in *Java* geschrieben, wird in der Version 2.6.2 eingesetzt und in der VM „*unitysrv1.awi.de*“ gehostet (s. Tabelle 4.2). Für die Interaktion mit dem Nutzer als auch Administrator wird eine Webschnittstelle zur Verfügung gestellt, die mit einem *Jetty* HTTP-Server realisiert wird. Als relationale Datenbank wird standardmäßig *H2* verwendet.

Die Installation des *SP-IdP-Proxys* ist relativ einfach. Das Installationsarchiv muss dafür nur heruntergeladen und anschließend entpackt werden. Für den eigentlichen Betrieb wird dann nur noch eine *Java Runtime* benötigt. Gestartet und beendet wird die Software mit *Bash*-Skripten, welche sich im *bin*-Verzeichnis des Installationsordners befinden. Für eine leichtere

Hostname	*unitysrv1.awi.de*
OS	*Ubuntu Linux* (64-bit) 18.04
CPU	1
RAM	4GB
Speicher	20GB + 10GB („/“)
Software	*Unity IdM Server* 2.6.2, *Java OpenJDK* 10, *Jetty* 9 HTTP-Server, *H2 Database*

Tabelle 4.2: VM-Spezifikation – *Unity IdM*

Bedienung kann das erwähnte Verzeichnis in die *Path*-Umgebungsvariable („*/etc/environment*“) aufgenommen werden. Um einen reibungslosen Betrieb – auch nach einem Neustart des Systems – zu gewährleisten, wurde zudem ein *init*-Skript („*unity-idm-server*“) verwendet und in den einzelnen „*Runlevels*“ des Betriebssystems verankert. Dazu wird das entsprechende Skript in das Verzeichnis „*/etc/init.d/*“ kopiert und durch den Befehl „*update-rc.d unity-idm-server defaults*“ angewandt. Aus Sicherheitsaspekten ist es außerdem ratsam, Serveranwendungen nicht als *root*-Benutzer ausführen zu lassen, daher wurde das Systemkonto „*unity-idm*“ angelegt („*adduser –system –no-create-home –group unity-idm*“) und im *init*-Skript berücksichtigt. Des Weiteren wurde die Anwendung „*authbind*“ konfiguriert, so dass auch ein unprivilegierter Systembenutzer den Webserver auf dem Port 443 starten kann. Das Anlegen der Datei „*/etc/authbind/byport/443*“ und das Zuweisen des Besitzers „*unity-idm*“ waren dafür ausreichend [UT18b].

Eine umfangreiche Anpassung war an dem Standard-Passwort-Hashverfahren von *Unity IdM* notwendig, damit eine anschließende Synchronisation mit einem LDAP-Verzeichnisdienst funktionieren konnte. Standardmäßig wird „*scrypt*“ als Hashverfahren verwendet [TT18, PJ16, RFC 7914], welches allerdings nicht in der Referenzimplementierung *OpenLDAP* ohne allzu großen Aufwand unterstützt wird. Daher fiel die Entscheidung, das Hashverfahren im *Java*-Quellcode von *Unity IdM* anzupassen, um eine aufwendige Umstrukturierung im Verzeichnisdienst zu umgehen und vor allem einen konventionellen Hashalgorithmus zu verwenden. Die Anpassung im Quellcode wurde im *Package* „*pl.edu.icm.unity.stdext.credential.pass*“ der *Library* „*unity-server-std-plugins-2.6.2.jar*“ durchgeführt. Betroffen war hauptsächlich die Klasse „*PasswordEngine*“ und alle hierauf bestehenden Abhängigkeiten. Gewählt wurde das Hashverfahren *SHA-256* mit einem 16-stelligen *Salt* [Hr11, RFC 6234], welches die *crypt(3)*-Funktion in der *glibc-Library* seit der Version 2.7 bereitstellt [Lin18]. In *Java* wird diese Funktion in der *Apache-Library* „*commons-codec-1.11.jar*“ und dort in der Klasse „*org.apache.commons.codec.digest.Crypt*“ implementiert. Die nachstehenden Listings zeigen die hinzugefügten Methoden in der Java-Klasse „*PasswordEngine*“, die die Grundlagen schaffen, die *crypt(3)*-Funktion für das Hashverfahren zu verwenden.

Die Erzeugung eines *crypt(3)*-Passworthashes ist im Listing 4.3 abgebildet. Charakteristisch ist hierbei der Aufbau im folgenden Format: $**ID**$rounds=**X**$**SALT**$**HASH**; dies wird noch einmal im Unterabschnitt 4.1.3 aufgegriffen. Sowohl der Passworthash als auch der *Salt* werden als *Base64*-codierte Zeichenketten in der Datenbank von *Unity IdM* gespeichert. Der entsprechende *Salt* wird durch die Methode „*genSaltCrypt*“ im Listing 4.4 bereitgestellt. Wichtig ist hierbei, dass nur Zeichen verwendet werden dürfen, die im *String* „*B64T*“ enthalten sind. Damit ein gespeichertes Passwort mit den vom Nutzer eingegebenen *Credentials* verglichen werden kann, bedarf es der Methode „*verifyCrypt*“ (s. Listing 4.5), die die notwendige Verifikation durchführt und einen *Boolean* zurückliefert. Zu beachten ist, dass der Austausch der angepassten *Library* vor dem erstmaligen Start des Proxys geschehen muss, da sonst fehlerhafte Einträge in der Datenbank erstellt und *Exceptions* geworfen werden.

```
private String[] crypt(String password, CryptParams params) {
  String saltCrypt = genSaltCrypt(params.getSaltLength());
  String crypt = Crypt.crypt(password,
  "$" + params.getIdHashingAlgorithm() + "$rounds=" + params.getRounds() + "$" +
    saltCrypt);
  return crypt.split("\\$");
}
```

Listing 4.3: *Java* – Erzeugung eines *crypt(3)*-Passworthashes

```
private Random random = new SecureRandom();
private static final String B64T = "./0123456789
    ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz";

private String genSaltCrypt(final int num) {
  final StringBuilder saltString = new StringBuilder();
  for (int i = 1; i <= num; i++) {
    saltString.append(B64T.charAt(random.nextInt(B64T.length())));
  }
  return saltString.toString();
}
```

Listing 4.4: *Java* – Erzeugung eines *crypt(3)-Salts*

```
private static final int CRYPT_HASH_POS = 4;

private boolean verifyCrypt(PasswordInfo stored, String password) {
  CryptParams params = new CryptParams(stored.getMethodParams());
  String crypt = Crypt.crypt(password, "$" + params.getIdHashingAlgorithm() + "
    $rounds=" + params.getRounds()
  + "$" + new String(stored.getSalt(), StandardCharsets.UTF_8));
  String[] cryptParts = crypt.split("\\$");
  byte[] testedHash = cryptParts[CRYPT_HASH_POS].getBytes(StandardCharsets.UTF_8);
  return Arrays.areEqual(testedHash, stored.getHash());
}
```

Listing 4.5: *Java* – Verifikation eines *crypt(3)*-Passworts

Die wesentlichen Konfigurationen von *Unity IdM* werden im Verzeichnis „*conf*“ des Installationsordners durchgeführt. Für den Start der *Java*-Anwendung werden Parameter in der Datei „*startup.properties*“ angegeben. Hier kann z. B. der zu verwendende Arbeitsspeicher angepasst und „*authbind*“ aktiviert werden. Die Hauptkonfigurationsdatei der Anwendung ist aber „*unityServer.conf*“, in der u. a. der Hostname, Zertifikatseinstellungen, bereitzustellende Endpunkte, Authentifizierungscontroller und alle weiteren grundlegenden Kerneinstellungen konfiguriert werden. Für die praktische Umsetzung im Testumfeld wurde das bereits vorkonfigurierte und selbst signierte TLS-Zertifikat aus dem Installationsordner genommen. Im produktiven Betrieb sollte unbedingt ein eigenes Zertifikat durch eine vertrauensvolle CA (*Certificate Authority*) ausgestellt werden. Es werden drei Endpunkte und ein Authentifizierungscontroller für das Szenario gebraucht. Für den Admin- und *Restful* API-Endpunkt mussten keine Konfigurationen vorgenommen werden. Anders sieht es beim Nutzerprofil-Endpunkt und dem SAML-Authentifizierungscontroller aus. Der SAML-Controller „*samlWeb*“ musste aktiviert und mit Hilfe der Datei „*samlAuthenticator.properties*“ angepasst werden. Ein Ausschnitt ist in Listing 4.6 zu sehen, wo die eigene „*EntityId*“, die Metadatenquelle für vertrauenswürdige IdPs und das Übersetzungsprofil „*sys:samlShib*“ angegeben werden [UT18b, UT16].

```
# (Issuer field). This should be unique URI which identifies the server.
unity.saml.requester.requesterEntityId=https://unitysrv1.awi.de

#Federation metadata configured trusted IdPs:
unity.saml.requester.metadataSource.federation.url=http://www.aai.dfn.de/fileadmin
    /metadata/dfn-aai-test-metadata.xml
unity.saml.requester.metadataSource.federation.perMetadataTranslationProfile=sys:
    samlShib
```

Listing 4.6: Einstellungen des SAML-Authentifizierungscontrollers [UT16]

Durch das Übersetzungsprofil (*Translation Profile*) wird es erst möglich die übermittelten Attribute aus der SAML-*Assertion* auf lokale *Unity IdM*-Attribute abzubilden. Das Profil wird über den Admin-Endpunkt im Webbrowser konfiguriert. Einstellungen am Nutzerprofil-Endpunkt werden hingegen in der Datei „*core.module*“, in der der anzuwendende Authentifizierungscontroller „*samlWeb*“ zugewiesen wird, sowie in der *Properties*-Datei „*userhome.properties*“ vorgenommen. In der zuletzt genannten Datei wird der visuelle und funktionale Aufbau des eigentlichen Endpunktes bestimmt. Das Resultat kann im Anhang in der Abbildung A.5, Abbildung A.7 und Abbildung A.8 betrachtet werden. Alle weiteren Einstellungen wurden über die Weboberfläche des Admin-Endpunktes (s. Abbildung A.9) getätigt. Hier wurde im ersten Schritt eine neue Gruppe mit dem Namen „*federation_exchange*“ angelegt, in der alle neuen Entitäten automatisiert hinzugefügt werden. Damit dieser Automatismus greift, wurde das Formular „*attribute_request*“ erstellt, welches nach erstmaliger Anmeldung am Nutzerprofil-Endpunkt modal aufgerufen wird. Hierüber wird auch die Aufforderung erzwungen, ein lokales Passwort „*sys:password*“ zu setzen. Das erstellte Formular ist in der Abbildung A.6 zu sehen und wird in der Weboberfläche unter der Rubrik „Registration & enquiry“ angelegt. Des Weiteren musste im Bereich „Server management“ ein *Translation Profile* mit dem Namen „*sys:samlShib*“ angelegt werden, das die SAML-Attribute auf lokale Attribute übersetzt. Dieses Übersetzungsprofil berücksichtigt die SAML-Attribute „*mail*“, „*sn*“, „*givenName*“, „*cn*“, und „*displayName*“, die auch bereits in der Filterdatei des *Shibboleth* IdPs freigegeben wurden. Meldet sich ein Nutzer an dem bereitgestellten Endpunkt an, wird entweder ein neues Konto in der Datenbank von *Unity IdM* erschaffen oder die gespeicherten und vorhandenen Attribute werden ggf. aktualisiert [UT18b, UT16]. Ein konkreter Ablauf dazu wird in der nachstehenden Abbildung 4.3 gezeigt.

Die Abbildung 4.3 zeigt ein Aktivitätsdiagramm, welches den Ablauf der erstmaligen Anmeldung eines Nutzers aus der DFN-AAI-Föderation am *SP-IdP-Proxy* darstellt. Durch Schwimmbahnen werden sowohl die beteiligten Akteure separiert als auch deren Zuständigkeiten deutlich. Den Anfang macht der Nutzer, indem er auf den Nutzerprofil-Endpunkt zugreift. Der Proxy antwortet mit dem angeforderten Endpunkt und stellt daraufhin einen SAML-Authentifikationscontroller bereit. Nachdem der Nutzer den Identitätsanbieter seiner Heimateinrichtung aus einer Liste ausgewählt hat (s. Abbildung A.5), erstellt *Unity IdM* eine Authentifikationsanfrage (*AuthnRequest*), die über den Webbrowser an den ausgewählten Heimat-IdP übermittelt wird. Dieser verarbeitet die Anfrage und fordert den Nutzer schließlich auf, sich mit seinen *Credentials* zu authentifizieren. Nach erfolgreicher Anmeldung wird der Nutzer aufgefordert einer Attributfreigabe zuzustimmen (s. Abbildung A.4). Wird diese genehmigt, wird durch den IdP eine Authentifikationsantwort (*Response*) mit allen freigegebenen Attributen über den Webbrowser des Nutzers an den *SP-IdP-Proxy* versendet. Danach wird auf Grundlage der Antwort eine neue Identität erschaffen, welche lokal in die Datenbank abgelegt wird. Damit auch ein lokales Passwort der digitalen Identität zugeordnet werden kann, füllt der Nutzer das Formular „*attribute_request*“ aus und schickt dieses ab (s. Abbildung A.6). Das bewirkt wiederum, dass der Proxy das Passwort zur Identität zusätzlich abspeichert und das Nutzerkonto automatisiert in die Gruppe „*federation_exchange*“ verschiebt; dies sind die Gruppeninformationen. Im letzten Schritt wird der Zugriff auf den

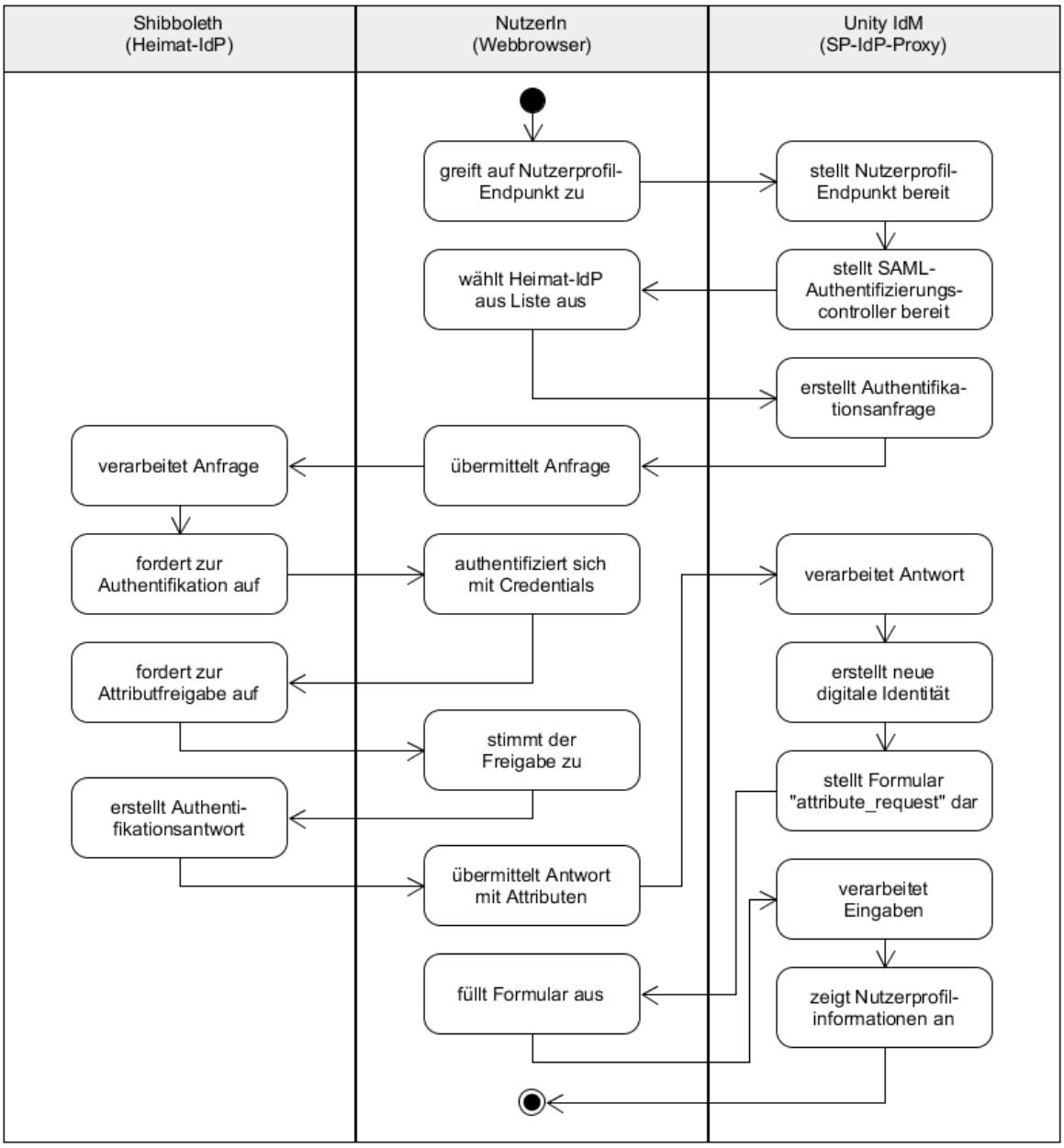

Abbildung 4.3: Erstmalige Anmeldung am *SP-IdP-Proxy* (*Unity IdM*)

Nutzerprofil-Endpunkt autorisiert und die Profilinformationen werden dem Nutzer zugänglich (s. Abbildung A.7, Abbildung A.8).

4.1.3 OpenLDAP

Die neu geschaffenen Entitäten aus dem *SP-IdP-Proxy* werden in ein *OpenLDAP*-Verzeichnis synchronisiert. Dafür wurde eine eigene VM (s. Tabelle 4.3) mit dem Hostnamen „*openldaps-rv1.awi.de*“ geschaffen, auf der der eigenständige LDAP-Dienst „*slapd*“ in der Version 2.4.45 installiert wurde. Außerdem werden hier auch Hilfsanwendungen aus dem Paket „*ldap-utils*“ bereitgestellt. Die Pakete wurden aus dem offiziellen *Ubuntu Repository* bezogen, was die Wartung und Installation vereinfacht. Für die Konfiguration des LDAP-Servers wurde der Befehl „*dpkg-reconfigure slapd*“ angewendet. Hier wurde auch der *Base DN* „*dc=unity,dc=awi,dc=de*“

des DITs angegeben. Der Aufbau des Namensraumes wurde bereits in der Abbildung 3.3 deutlich.

Hostname	*openldapsrv1.awi.de*
OS	*Ubuntu Linux* (64-bit) 18.04
CPU	1
RAM	1GB
Speicher	20GB + 10GB („/“)
Software	*OpenLDAP server (slapd)* 2.4.45, *OpenLDAP utilities (ldap-utils)* 2.4.45

Tabelle 4.3: VM-Spezifikation – *OpenLDAP*

Generell werden in der praktischen Umsetzung alle Modifikationen im LDAP-Verzeichnisdienst mit sogenannten LDIF-Dateien (*LDAP Data Interchange Format*) durchgeführt [Goo00, RFC 2849]. Dies sind einfache Textdateien, die nach einem speziellen Format aufgebaut sind. Wichtig ist vor allem, dass in der obersten Zeile eines Blockes immer der eindeutige Pfad zu einem Objekt (DN) steht, damit die Modifikationen zugeordnet werden können. Mit den Befehlen „*ldapadd*“ oder „*ldapmodify*“ können die LDIF-Dateien verwendet werden. Letzterer benötigt zusätzlich die Angabe eines „*changetype*“ und hat daher die Möglichkeit alle verfügbaren Modifikationen (*Modify*, *Add*, *Delete*) an einem Objekt durchzuführen. „*ldapadd*“ hingegen kann nur neue Objekte dem Verzeichnis hinzufügen. Für eine bessere Zuordnung von Gruppen und Nutzerkonten im Verzeichnis wurden die zwei Organisationseinheiten (*organizationalUnit*) „*People*“ und „*Groups*“ geschaffen. Mit Hilfe der LDIF-Datei „*add_basic.ldif*“ im Listing 4.7 und dem Befehl „*ldapadd -x -D cn=admin,dc=unity,dc=awi,dc=de -W -f add_basic.ldif*“ konnten diese „Zweige“ erstellt werden. Außerdem wurde die Gruppe „*federation_exchange*“, der alle Entitäten aus *Unity IdM* hinzugefügt werden, und das Testnutzerkonto „*wwichtig*“ angelegt.

```
dn: ou=People,dc=unity,dc=awi,dc=de
objectClass: organizationalUnit
ou: People

dn: ou=Groups,dc=unity,dc=awi,dc=de
objectClass: organizationalUnit
ou: Groups

dn: cn=federation_exchange,ou=Groups,dc=unity,dc=awi,dc=de
objectClass: posixGroup
cn: federation_exchange
gidNumber: 5000

dn: uid=wwichtig,ou=People,dc=unity,dc=awi,dc=de
objectClass: inetOrgPerson
objectClass: posixAccount
objectClass: shadowAccount
uid: wwichtig
sn: Wichtig
givenName: Willi
cn: Willi Wichtig
displayName: Willi Wichtig
uidNumber: 10000
gidNumber: 5000
userPassword: {CRYPT}$5$rounds=5000$aTJ.LJ9kmO2B8ils$noooGBQ.
    mLtFyL4DmfQrK5NHpcAxE8n9LyIOhdE69r9
loginShell: /bin/bash
homeDirectory: /home/wwichtig
```

Listing 4.7: LDIF – Erweiterung des DITs

ID	KDF	SALT (#Zeichen)	HASH (#Zeichen)
	DES (Data Encryption Standard)	2	11
1	MD5 (Message Digest 5)	≤ 16	22
5	SHA-256 (seit glibc 2.7)	≤ 16	43
6	SHA-512 (seit glibc 2.7)	≤ 16	86

Tabelle 4.4: Unterstützte KDFs in der *crypt(3)*-Funktion [Lin18]

Damit im späteren Verlauf der Synchronisation keine Schwierigkeiten bezüglich des Passworts auftreten, musste ein auf allen beteiligten Systemen vorhandener und sicherer Hashalgorithmus zum Einsatz kommen. Nach einiger Recherche wurde der *SHA-256* mit einem 16-stelligen *Salt* aus der *crypt(3)*-Funktion genommen (s. Tabelle 4.4). *SHA-256* [Hr11, RFC 6234] entspricht noch den momentan gültigen Standards und kann auf allen involvierten Plattformen benutzt werden. Die Verwendung eines *Salts* und die Möglichkeit den Hashalgorithmus durch Angabe von „*rounds*“ öfters zu durchlaufen, geben noch einmal zusätzliche Sicherheit. Sollte *SHA-256* in Zukunft nicht mehr den Sicherheitsanforderungen genügen, könnte im betrachteten Szenario mit Leichtigkeit auf *SHA-512* umgestiegen werden. Folgendes Format muss der LDAP-Passworteintrag eines Nutzers im Verzeichnisdienst, bestimmt durch die *crypt(3)*-Funktion, haben [OT18b, Lin18, s. Password Storage]:

- userPassword: {CRYPT}$**ID**$rounds=**X**$**SALT**$**HASH**
- Dabei gilt: **ID** $\in$ {1,5,6}, $1000 \leq$ **X** ≤ 999999999,
  SALT, **HASH** $\in$ {[a-z]*,[A-Z]*,[0-9]*,[.]*,[/]*}

Die richtige Reihenfolge der Bestandteile „ID“, „X“, „SALT“, „HASH“ und das Integrieren des Trennungszeichens „$“ sind hier besonders zu beachten. Vor der eigentlichen Zeichenkette wird in geschweiften Klammern angegeben, um welches Schema es sich handelt. Außerdem müssen die oben erwähnten Bedingungen erfüllt sein und auf eine konkrete *Key Derivation Function* (KDF) aus der Tabelle 4.4 abzielen [Lin18]. Das oben definierte Passwortschema kann durch die LDIF-Datei „*defaultPasswordConf.ldif*“ in Listing 4.8 und dem Befehl „*ldapmodify -Q -Y EXTERNAL -H ldapi:/// -f defaultPasswordConf.ldif*“ als Standard gesetzt werden. Wenn nun ein Passwort durch die Anwendung „*ldappasswd*“ angelegt werden sollte, wird ein *SHA-256*-Hash mit einem zufälligen 16-stelligen *Salt* erstellt [OT18b].

```
dn: cn=config
changetype: modify
add: olcPasswordHash
olcPasswordHash: {CRYPT}
-
add: olcPasswordCryptSaltFormat
olcPasswordCryptSaltFormat: $5$rounds=5000$%.16s
```

Listing 4.8: LDIF – Änderung des Standard-Passwortschemas

4.1.4 VMware vRealize Automation

Die Software vRA in der Version 7.4.0 soll organisationsübergreifend in der DFN-AAI-Föderation bereitgestellt werden. Damit der Zugriff auf die Anwendung möglich ist, wird das zuvor konfigurierte LDAP-Verzeichnis als Authentifizierungsquelle in der Administrationsoberfläche von vRA hinzugefügt (s. Abbildung A.11, Abbildung A.12) [VT18a]; zum Erstaunen wird ein zusätzliches Attribut (Zweckentfremdung von „*description*“) für den DN eines Eintrags benötigt. Allzu große Einstellungen lassen sich aus Systemadministrationssicht nicht vornehmen, da es sich um eine *Virtual Appliance* von *VMware* handelt [VT18b]. Sobald der Verzeichnisdienst integriert ist, können

sich föderative Nutzer an der *Marketplace*-Software durch Auswählen der Authentifizierungsquelle „*unity.awi.de*“ anmelden (s. Abbildung A.10). Die vorherige Erstellung einer lokalen Identität durch die Anmeldung an dem *SP-IdP-Proxy* ist allerdings eine Voraussetzung dafür.

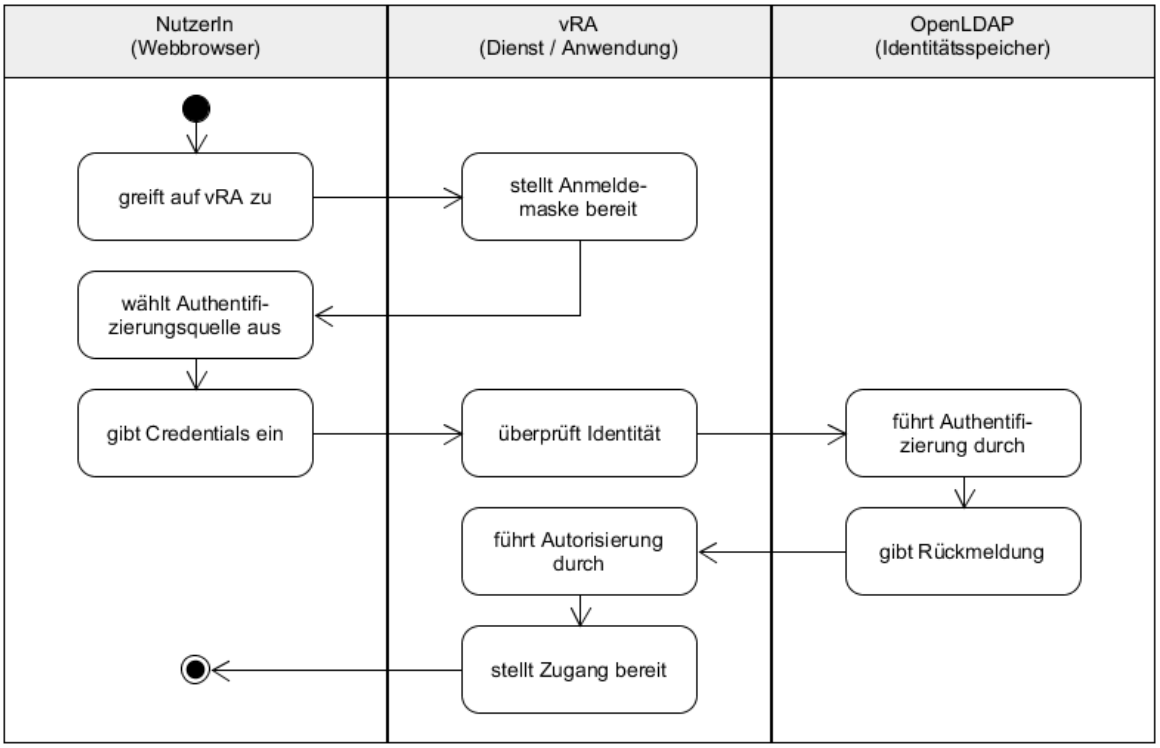

Abbildung 4.4: Anmeldung eines (föderativen) Nutzers an lokaler Anwendung (vRA)

Die Abbildung 4.4 verdeutlicht den Ablauf der Anmeldung eines Nutzers an der Plattform *vRealize Automation*, der aus der DFN-AAI-Föderation stammen kann. Initial greift der Nutzer auf das Portal zu und wählt seine Authentifizierungsquelle aus. Nach der Eingabe seiner *Credentials* wird die Identität durch vRA überprüft. Dafür wird eine Authentifizierung an dem entsprechenden Verzeichnisdienst (*OpenLDAP*) durchgeführt. Auf Grundlage der empfangenen Rückmeldung des LDAP-Servers erfolgt schließlich eine Autorisierung des Nutzers durch vRA und dieser erlangt, wenn ausreichende Berechtigungen vorhanden sind, den Zugriff auf die *Marketplace*-Software. Natürlich muss in einem vorherigen Schritt ein lokales Nutzerkonto mit einem Passwort über *Unity IdM* erstellt worden sein; hier erhält der Nutzer auch den notwendigen Benutzernamen (*userName*). Ist die Anmeldung geglückt, können diverse Angebote, z. B. VMs mit vorkonfigurierter Betriebssystem- und Anwendungssoftware, aus dem Servicekatalog angefordert werden. Die Einbettung der lokalen Anwendung in einem Föderationsumfeld ist somit durchgeführt. Auch andere Dienste, die LDAP „sprechen“, könnten über das betrachtete Szenario bereitgestellt werden.

4.2 Implementierung

Beide Teile der Benutzer-Entitätensynchronisation („*sync-process-one*“, „*sync-process-two*“) wurden auf zwei verschiedenen VMs implementiert. Der erste Prozess läuft auf „*unitysrv1.awi.de*“ und der zweite auf „*openldapsrv1.awi.de*“. Es wurden einige grundlegende Prinzipien aufgestellt, die unbedingt erfüllt werden sollten. Sämtliche auftretende Kommunikation im Netzwerk und der Dateitransport mussten über einen verschlüsselten Transportkanal erfolgen. Um dies zu

bewerkstelligen, wurde auf SSH, SCP und HTTPS zurückgegriffen. Außerdem sollten keine Klartextpasswörter bei der Kommandoausführung und im zugehörigen Prozess ersichtlich sein. Dazu wurden die Passwörter in Dateien ausgelagert und der Zugriff auf diese nur dem jeweiligen Prozessbenutzer gestattet. Des Weiteren sollte eine Protokollierung in einer zentralen Protokolldatei möglich sein, damit Prozessfehler im Nachgang ermittelt werden können. Auch die Bedienung der beiden *Bash*-Skripte sollte komfortabel gelöst sein. Durch die Verwendung von sogenannten *init*-Skripten im Verzeichnis „*/etc/init.d/*“ wird dies mit Hilfe des Befehls „*service {sync-process-one|sync-process-two} {start|stop|status|restart|help}*“ gelöst. Zuvor wurden die Skripte mit „*update-rc.d {sync-process-one|sync-process-two} defaults*“ in den einzelnen „*Runlevels*“ des Betriebssystems verankert. Zudem ermöglicht das Vorhandensein eines *init*-Skripts den automatischen Start des Prozesses nach beispielsweise einem Neustart des Systems. Im *init*-Skript selbst werden dann die zuvor erstellten Start- bzw. Stopp-Skripte des jeweiligen Synchronisierungsprozesses je nach Aufforderung ausgeführt. Damit die erstellten *Bash*-Skripte im Hintergrund der VMs laufen und auch ein Abmeldevorgang des Nutzers keine Auswirkung hat, wird das Programm „*nohup*“ verwendet (s. Listing 4.9). Alle Ausgabekanäle werden zudem in eine Protokolldatei umgeleitet.

```
1 nohup "${BIN_DIR}/$SCRIPT" >>"${LOG_DIR}/$LOG_FILE" 2>&1 &
```

Listing 4.9: Ausführung der *Bash*-Skripte mit „*nohup*“

Vor der Ausführung der *Bash*-Skripte mussten einige vorbereitende Maßnahmen durchgeführt werden. Dies beinhaltete sowohl die Erstellung von Systembenutzern („*sync-user-one*“, „*sync-user-two*“) und deren Verzeichnisse durch den Befehl „*adduser –system –group {sync-user-one|sync-user-two}*“, als auch die Erstellung („*ssh-keygen -t rsa -b 4096*“) und der Austausch, durch die Eintragung des öffentlichen Schlüssels des Senders in die Datei „*.ssh/authorized_keys*“ des Empfängers, von SSH-Schlüsseln. Damit der Zugriff via *Secure Shell* (SSH) funktionieren konnte, musste eine *Login-Shell* („*/bin/bash*“) in der Datei „*/etc/passwd*“ dem jeweiligen Systembenutzer hinzugefügt werden. Außerdem wurden Passwort- und *Properties*-Dateien sowie eine rudimentäre Verzeichnisstruktur in den Benutzerverzeichnissen angelegt. Bei der Erstellung der Skripte wurde darauf geachtet, dass wiederkehrende Logikbestandteile in Funktionen ausgelagert werden und die Nutzung von Variablen – teilweise durch die *Properties*-Dateien belegt – forciert wird.

Als zusätzliche Sicherheitsmaßnahme werden Dateien, die über Rechnergrenzen hinweg übermittelt werden, mit einem sogenannten *One-time Password* (OTP) verschlüsselt [Eck18, vgl. S. 451ff]. Dieser Zusatz wurde gewählt, da sogar Nutzerpassworte und der zugehörige *Salt* in den LDIF-Dateien übertragen werden. Das Passwort liegt zwar nur in Form eines Hashes vor, allerdings könnte mit Hilfe des *Salts* und durch Kenntnis der Hashmethode auf das Passwort durch unzähliges Ausprobieren geschlossen werden. Ein solches Verfahren wird auch als *Brute-Force*-Angriff oder erschöpfende Suche verstanden [Eck18, vgl. S. 358]. Das OTP gewährt die Anwendung einer symmetrischen Verschlüsselung der zu übertragenen Dateien. Charakteristisch ist dabei, dass der Schlüssel für die Ver- und Entschlüsselung dabei gleich ist. Die Kommunikationspartner müssen sich folglich vorab über einen gemeinsamen Schlüssel verständigen und diesen austauschen [Eck18, vgl. S. 288f.]. Für den sicheren Austausch des OTPs kommt ein weiteres Verfahren zum Einsatz, die asymmetrische Verschlüsselung. Bei ihr ist es charakteristisch, dass jeder Kommunikationsteilnehmer ein Schlüsselpaar bestehend aus einem geheimen Schlüssel (*Private Key*) und einem öffentlich bekannten Schlüssel (*Public Key*) besitzt. Auch hier existieren geheime Schlüssel, allerdings müssen diese nicht wie bei der symmetrischen Verschlüsselung sicher ausgetauscht werden [Eck18, vgl. S. 289f.]. In Listing 4.10 ist dargestellt, wie der private Schlüssel (Zeile 1, „*private_key.pem*“) und der dazugehörige öffentliche Schlüssel (Zeile 2, „*public_key.pem*“) auf beiden Seiten generiert wurden. Anschließend musste nur noch der öffentliche Schlüssel des Empfängers dem jeweiligen Sender bekannt gemacht werden.

```
openssl genpkey -algorithm RSA -out private_key.pem -pkeyopt rsa_keygen_bits:4096
openssl rsa -pubout -in private_key.pem -out public_key.pem
```

Listing 4.10: Initiale Schlüsselpaargenerierung mit „*openssl*“ [OT18c]

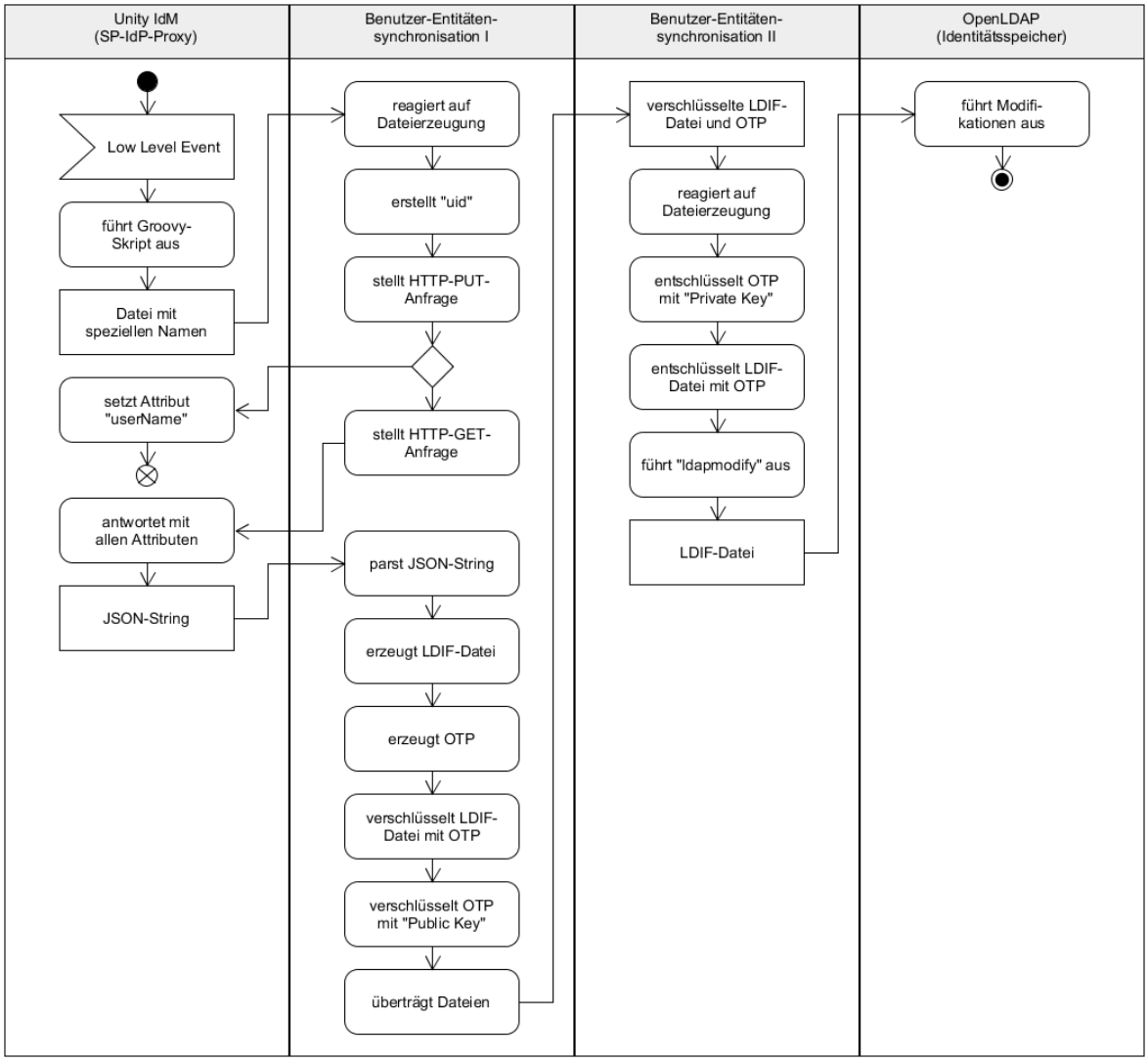

Abbildung 4.5: Entitäten-Synchronisation nach erstmaliger Anmeldung am *SP-IdP-Proxy*

Das Aktivitätsdiagramm in der Abbildung 4.5 stellt den Ablauf der Benutzer-Entitätensynchronisation nach der erstmaligen Anmeldung eines Nutzers an *Unity IdM* dar. Es soll das Zusammenspiel aller Teilkomponenten in der zu erstellenden Komponente verdeutlichen. Ausgelöst wird die Aktivität durch ein sogenanntes „*Low Level Event*“ – hier ist es konkret das Abschicken des Formulars „attribute_request“ durch den Nutzer – am *SP-IdP-Proxy*, das wiederum die Ausführung eines *Groovy*-Skripts bewirkt. In diesem Skript werden verschiedene Ereignisse behandelt und daraufhin Dateien mit einem speziellen Aufbau erzeugt, u. a. sind der Ereignisname und die „*entityId*“ Bestandteile; dies wird im Unterabschnitt 4.2.1 noch einmal genauer aufgegriffen. Wird nun so eine Datei in ein speziell vorkonfiguriertes Verzeichnis erstellt, wird das erste Synchronisierungsskript aktiv. Im ersten Schritt wird eine auf die Entität referenzierende „*uid*“ erstellt, die im Verzeichnisdienst noch eine wesentliche Rolle einnehmen wird (Schlagwort: DN). Danach wird diese „*uid*“ als ein neues Attribut („*userName*“) der Entität in *Unity IdM* durch eine HTTP-PUT-Anfrage integriert. Anschließend wird eine HTTP-GET-Anfrage an den

RESTful API-Endpunkt des Proxys gestellt, mit der alle momentan gespeicherten Attribute zu einer Entität im JSON-Format erlangt werden. Nachdem die JSON-Zeichenkette eingegangen ist, wird diese geparst und die Erstellung einer LDIF-Datei beginnt. Ist die LDIF-Datei durch das erste Synchronisierungsskript fertiggestellt, wird mit der Verschlüsselung derselben gestartet. Dafür wird ein OTP erzeugt und anschließend für die Verschlüsselung der LDIF-Datei verwendet. Abschließend wird das OTP mit Hilfe eines öffentlichen Schlüssels des Empfängers verschlüsselt und die Übertragung der beiden kryptographischen Dateien zum Empfänger durchgeführt. Jetzt wird das zweite Synchronisierungsskript aktiv, indem es mit der Entschlüsselung des OTPs durch den eigenen privaten Schlüssel beginnt. Danach wird die LDIF-Datei mit Hilfe des OTPs wieder entschlüsselt und ein „*ldapmodify*“ wird auf das lokale *OpenLDAP*-Verzeichnis abgesetzt. Die letzte Aktivität führt der Verzeichnisdienst aus, der die in der LDIF-Datei befindlichen Modifikationen umsetzt. Durch dieses Zusammenspiel der Teilkomponenten konnte schließlich eine föderative Entität auf ein lokales Benutzerkonto abgebildet werden.

4.2.1 Benutzer-Entitätensynchronisation I

Der erste Teil der Benutzer-Entitätensynchronisation beschäftigt sich hauptsächlich damit bei einem bestimmten Ereignis am *Unity IdM*, eine LDIF-Datei zu erstellen, dann zu verschlüsseln und abschließend an einen nachfolgenden Prozess (s. Unterabschnitt 4.2.2) weiterzureichen. Das komplette *Bash*-Skript ist im Anhang und dort im Listing A.8 zu finden. Für die Auslösung der sogenannten „*Low Level Events*“ muss die Konfiguration, wie im Listing 4.11 zu sehen, in der Datei „*unityServer.conf*“ am *SP-IdP-Proxy* vorgenommen werden [UT18b, *Enhancement scripts*]. Hier werden ausgewählte Ereignisse aktiviert und einem *Groovy*-Skript zugeordnet, welches bei Eintreten eines registrierten Vorfalls ausgeführt wird.

```
unityServer.core.enableLowLevelEvents=true
unityServer.core.script.n.file=${CONF}/scripts/myGroovy_basicEvents.groovy
unityServer.core.script.n.trigger=methodInvocation.submitEnquiryResponse
```

Listing 4.11: Aktivierung der „*Low Level Events*“ in *Unity IdM* [UT18b]

Das erwähnte *Groovy*-Skript „*myGroovy_basicEvents.groovy*“ ist vollständig im Listing A.7 abgedruckt. Hier wird auf die verschiedensten Ereignisse reagiert, indem Dateien mit einem speziellen Namensformat erstellt werden. Die nachstehende Auflistung enthält alle relevanten Ereignisse mit den dazugehörigen Formaten der Dateinamen:

- Eine Entität wurde erstellt bzw. ein Formular wurde abgeschickt – entityId_<entityId>_methodInvocation.submitEnquiryResponse_<timestamp>.unity
- Die *Credentials* einer Entität wurden neu erstellt – entityId_<entityId>_methodInvocation.setEntityCredential_<timestamp>.unity
- Eine Entität wurde entfernt – entityId_<entityId>_methodInvocation.removeEntity_<timestamp>.unity
- Ein Attribut wurde neu erstellt – entityId_<entityId>_methodInvocation.createAttribute_<attributeName>_<timestamp>.unity
- Ein Attribut wurde neu gesetzt – entityId_<entityId>_methodInvocation.setAttribute_<attributeName>_<timestamp>.unity

- Ein Attribut wurde entfernt – entityId_<entityId>_methodInvocation.removeAttribute_<attributeName>_<timestamp>.unity

Gestartet wird immer mit der „*entityId*“, auf die sich die Transaktion bezieht. Danach folgt der Ereignisname, ggf. ein Attributname, ein Zeitstempel um Dopplungen zu vermeiden, und als Letztes die Dateiendung „*.unity*“.

Damit auf das Erstellen von Dateien reagiert werden kann, wird das Programm „*inotifywait*“ aus dem Paket „*inotify-tools*“ verwendet. Hierdurch wird eine speicherbasierte Interprozesskommunikation zwischen *Unity IdM* und dem *Bash*-Skript hergestellt. Der Aufruf des „Wächters“ endet in einer Endlosschleife wie im Listing 4.12 zu sehen ist. Es wird sowohl auf die Erstellung als auch auf eine Metadatenveränderung einer Datei gewartet und der betroffene Dateiname in der Variable „*$FILE*“ gespeichert. Sollte der erste Teil der Benutzer-Entitätensynchronisation ausfallen, könnte zu einem späteren Zeitpunkt beispielsweise durch das Programm „*touch*“ liegengebliebene Dateien wieder aktiviert werden.

```
1 inotifywait -mq -e create -e attrib --format %f $MONITORING_DIR | while read FILE
```

Listing 4.12: Überwachung eines Verzeichnisses mit „*inotifywait*“

Um mit den *RESTful API*-Endpunkt von *Unity IdM* zu kommunizieren, kommt das Programm „*curl*“ zum Einsatz (s. Listing 4.13). Es wird sowohl eine HTTP-PUT-Anfrage (Zeile 1) als auch eine HTTP-GET-Anfrage (Zeile 3) im Laufe der Skriptlogik übermittelt. Mit Hilfe des ersten Statements und der Erstellung einer JSON-Zeichenkette wird die neu geschaffene „*uid*“ als „*userName*“ im Proxy eingetragen. Die „*uid*“ wird dabei immer nach folgendem Format aufgebaut: „*awiFed*“ + „*$entityId*“. Alle zu einer Entität gehörenden Attribute können hingegen mit dem zweiten Aufruf erhalten und in eine temporäre Datei im JSON-Format gespeichert werden. Wie eine solche Datei aussieht, kann dem Listing A.6 im Anhang entnommen werden.

```
1 curl -s -k --netrc-file $CONF_DIR/.netrc -X PUT -H "Content-Type: application/json
    " -d "{\"values\":[\"$uid\"],\"name\": \"userName\",\"groupPath\": \"/\"}" "
    https://unitysrv1.awi.de/rest-admin/v1/entity/$entityId/attribute"

3 curl -s -k --netrc-file $CONF_DIR/.netrc "https://unitysrv1.awi.de/rest-admin/v1/
    entity/$entityId/attributes" | sed -e 's/\\//g' -e 's/"{/{/g' -e 's/}"/}/g' >
    ${WORK_DIR}/$FILE.tmp
```

Listing 4.13: HTTP-Anfragen an die *RESTful API* mit „*curl*“

Nachdem die Attributinformationen einer Entität erhalten wurden, müssen diese noch entsprechend geparst werden. Da es sich um das JSON-Format handelt, in der die temporäre Datei vorliegt, wird hier das Programm „*jq*“ mit seiner speziellen Syntax angewendet; dies wird anhand des Listing 4.14 dargestellt, indem der „*surname*“ einer Entität entnommen wird. Beim Passworthash und *Salt* ist außerdem zu beachten, dass diese in einer *Base64*-kodierten Form übermittelt werden. Sie müssen daher noch mit Hilfe des Befehls „*base64 -d*“ dekodiert werden.

```
1 result=$(cat ${WORK_DIR}/$FILE.tmp | jq -c '.[] | select(.["name"] == "surname") |
    .["values"][0]' | sed 's/"//g')
```

Listing 4.14: „*surname*“ – Parsen der JSON-Datei mit „*jq*“

Anschließend findet das eigentliche Erzeugen einer LDIF-Datei statt. Eine Übersicht aller möglichen Dateiinhalte anhand konkreter Beispiele sind im Abschnitt A.7 des Anhangs zu finden. Das Listing 4.15 demonstriert den schrittweisen Aufbau einer LDIF-Datei, der aufgrund neuer

Credentials einer Entität veranlasst wird. Durch die Funktion „*writeInLdif*“ werden zeilenweise Einträge in die neue Datei hinzugefügt. Wichtig ist hier die LDIF-Syntax zu beachten, da es sonst zu Fehlern im Synchronisierungsprozess kommt. Die Funktion „*getJsonValue*“ ermöglicht es Attributwerte aus der temporären Datei im JSON-Format auszulesen und für den weiteren Aufbau der LDIF-Datei zu verwenden.

```
writeInLdif "dn:" "uid=$uid,ou=People,dc=unity,dc=awi,dc=de"
writeInLdif "changetype:" "modify"
writeInLdif "replace:" "userPassword"
passwdHash=$(getJsonValue "hash")
passwdSalt=$(getJsonValue "salt")
writeInLdif "userPassword:" '{CRYPT}$5$rounds=5000$'$passwdSalt'$'$passwdHash
```

Listing 4.15: LDIF-Erstellung aufgrund neuer *Credentials* einer Entität

```
openssl rand -base64 -out ${WORK_DIR}/$FILE.otp 128

openssl enc -aes-256-cbc -salt -in ${WORK_DIR}/$FILE.ldif -out ${WORK_DIR}/$FILE.
    ldif.enc -pass file:${WORK_DIR}/$FILE.otp

openssl rsautl -encrypt -inkey ${CONF_DIR}/public_key.pem -pubin -in ${WORK_DIR}/
    $FILE.otp -out ${WORK_DIR}/$FILE.otp.enc
```

Listing 4.16: Verschlüsselung der zu übertragenden Dateien mit „*openssl*“ [OT18c]

Sobald die Erstellung einer LDIF-Datei beendet ist, wird die Verschlüsselung der zu übertragenden Dateien mit Hilfe der Anwendung „*openssl*“ gestartet. Alle hierfür benötigten Befehle sind dem Listing 4.16 zu entnehmen. Als Erstes wird ein zufällig generiertes OTP erstellt (Zeile 1, „*$FILE.otp*“). Daraufhin wird die zuvor erzeugte LDIF-Datei mit diesem OTP symmetrisch verschlüsselt (Zeile 3, „*$FILE.ldif.enc*“) und im letzten Schritt wird zudem das OTP verschlüsselt (Zeile 5, „*$FILE.otp.enc*“); allerdings erfolgt dies asymmetrisch mit dem öffentlichen Schlüssel („*public_key.pem*“) des Empfängers [OT18c].

Als abschließende Maßnahme im ersten Teil der Benutzer-Entitätensynchronisation wird die Übertragung der kryptographischen Dateien („*$FILE.otp.enc*“, „*$FILE.ldif.enc*“) durchgeführt. Sie erfolgt via SSH / SCP und durch den vorherigen Austausch des öffentlichen SSH-Schlüssels passwortlos. Die konkreten Befehle sind im Listing 4.17 abgebildet. Es ist auch zu überlegen, die Übertragung der beiden Dateien zeitlich voneinander zu separieren, falls die Sicherheitsbestimmungen dies vorsehen sollten.

```
scp ${WORK_DIR}/$FILE.otp.enc openldap:~/work/.
scp ${WORK_DIR}/$FILE.ldif.enc openldap:~/monitoring/.
```

Listing 4.17: Übertragung der kryptographischen Dateien mit „*scp*“

4.2.2 Benutzer-Entitätensynchronisation II

Der zweite Teil der Benutzer-Entitätensynchronisation schließt nahtlos an den ersten Prozess an, entschlüsselt die erhaltenen Dateien und führt die entsprechenden Modifikationen in den entschlüsselten LDIF-Dateien im Verzeichnisdienst *OpenLDAP* aus. Das komplette *Bash*-Skript wird im Listing A.9 dargestellt.

Für das automatisierte Entgegennehmen der verschlüsselten Dateien kommt wieder das Programm „*inotifywait*“ aus dem Paket „*inotify-tools*“ wie im ersten Teil des Prozesses zum Einsatz (s. Listing 4.12). Der Aufbau des Dateinamens ist hier für die speicherbasierte Kommunikation über

Dateien abermals entscheidend. Das Format muss dem Ausdruck „*entityId_*_methodInvocation.*.unity.ldif.enc*“ entsprechen, damit die gewünschte Verarbeitung angestoßen wird. Des Weiteren muss sich zeitgleich das dazugehörige verschlüsselte OTP („*$FILE.otp.enc*“) im Arbeitsverzeichnis des Synchronisierungsskripts befinden.

```
openssl rsautl -decrypt -inkey ${CONF_DIR}/private_key.pem -in ${WORK_DIR}/$FILE.
    otp.enc -out ${WORK_DIR}/$FILE.otp
2
openssl enc -d -aes-256-cbc -in ${WORK_DIR}/$FILE.ldif.enc -out ${WORK_DIR}/$FILE.
    ldif -pass file:${WORK_DIR}/$FILE.otp
```

Listing 4.18: Entschlüsselung der empfangenen Dateien mit „*openssl*“ [OT18c]

Nach dem Auslösen des „Wächters“ im überwachten Verzeichnis wird die Entschlüsselung der übertragenen Dateien begonnen. Dies wird im Listing 4.18 mit Hilfe der Anwendung „*openssl*“ aufgegriffen. Zuallererst wird das OTP mit dem privaten Geheimnis („*private_key.pem*“) des Empfängers entschlüsselt (Zeile 1, „*$FILE.otp*“). Danach wird die verschlüsselte LDIF-Datei durch das zuvor erhaltene OTP wieder lesbar gemacht (Zeile 3, „*$FILE.ldif*“) und für den nächsten Schritt abgelegt [OT18c].

Abschließend werden die erstellten Modifikationen in der nun zugänglichen LDIF-Datei auf das *OpenLDAP*-Verzeichnis durch das Programm „ldapmodify“ und den Ausdruck im Listing 4.19 angewendet. Kommt es hier zu Fehlern, wird durch das Abfragen des Beendigungsstatus in der Variable „*$?*“ ein entsprechender Vermerk im Protokoll getätigt; dies erfolgt an mehreren entscheidenden Stellen in den beiden *Bash*-Skripten.

```
1 ldapmodify -x -D cn=admin,dc=unity,dc=awi,dc=de -y $CONF_DIR/.passwd -f $WORK_DIR/
    $FILE.ldif
```

Listing 4.19: Ausführung von „*ldapmodify*“

Wie bereits an den erzeugten Dateinamen mehr oder weniger ersichtlich wird, können im betrachteten Szenario über die LDIF-Dateien Entitäten erschaffen, entfernt und zugehörige Attribute erstellt, bearbeitet und wieder gelöscht werden. Konkrete Beispiele des Aufbaus mit den charakteristischen Dateinamen sind im Anhang und dort im Abschnitt A.7 zu finden. Alle hier behandelten Attribute müssen natürlich durch entsprechende Schemata im Verzeichnisdienst definiert sein (s. Tabelle 3.1). Die im ersten Skript erzeugten Attributwerte wurden bereits mit Sonderzeichen und Umlauten getestet und konnten dem Verzeichnisdienst erfolgreich hinzugefügt werden; es bedarf keiner vorherigen *Base64*-Kodierung der Zeichenketten in den LDIF-Dateien. Fehler würden nur bei leeren Attributen auftreten, die aber durch die Skriptlogik grundsätzlich vermieden werden.

5 Fazit und Ausblick

Wesentlich hat sich diese Ausarbeitung mit dem vielschichtigen Themenkomplex Identitätsmanagement (IdM) befasst. In den ersten Kapiteln wurden die in diesem Zusammenhang stehenden Grundlagen und Verfahren vermittelt (s. Kapitel 2, Kapitel 3). Es hat sich gezeigt, dass das föderative Identitätsmanagement (FIM) eine logische Weiterentwicklung des lokal-begrenzten Authentifizierungs- und Berechtigungswesen in Organisationen darstellt. Der Bedarf an verteilter digitaler Identitäten wird voraussichtlich in der Zukunft weiter zunehmen und stellt die Unternehmen und Einrichtungen im Bezug auf das Management dieser Entitäten vor bekannte und neue Herausforderungen; dies ist nicht zu Letzt am Zustandekommen des HDF-Projekts zu erkennen [HT18]. Die wichtige Basis der Identitätsverteilung in einer entsprechenden AAI ist dabei aber immer noch ein vorhandenes und funktionstüchtiges IdM im eigenen Organisationsumfeld (Schlagwort: Verteiltes FIM). In Föderationen wie der DFN-AAI werden digitale Identitäten über Identitätsanbieter (IdPs) der jeweiligen Heimateinrichtungen bereitgestellt und sind somit die zentrale Schnittstelle im FIM für die Diensteanbieter (SPs). Natürlich bleiben die wesentlichen Konzepte der Authentifizierung und Autorisierung erhalten, um domänenfremden Nutzern föderative Dienste anzubieten. Die Autorisierung des einzelnen Nutzers erfolgt dabei vermehrt an einem sogenannten *SP-IdP-Proxy* [ABB+18, SFH+18, vgl. S. 5f.], welcher die verschiedensten Protokolle und Verfahren beherrscht; so wie es auch im praktischen Anteil dieser Arbeit zu sehen ist.

Die praktische Umsetzung aus Kapitel 4 ermöglicht es, eine lokale Software (*VMware vRealize Automation*) in einem Föderationsumfeld (DFN-AAI) einzubetten. Dies wurde mit Hilfe des SAML-*Frameworks* zur Identitätsverteilung, einem *SP-IdP-Proxy* (*Unity IdM*) als Vermittler zwischen dem *Alfred-Wegener-Institut* und den Mitgliedern der AAI, sowie einem Identitätsspeicher in Form eines Verzeichnisdienstes (*OpenLDAP*) umgesetzt. Für die Synchronisation der einzelnen Benutzerkonten wurde außerdem ein zweiteiliges *Bash*-Skript entwickelt. Erste Ansätze der Entitätensynchronisation in den lokalen Verzeichnisdienst sahen vor, ein regelmäßiges *cronjob*-gesteuertes Abarbeiten jeder einzelnen Entität durchzuführen. Dieses Verfahren wurde schnell verworfen und ein ereignisbasierter Ansatz forciert, um unnötige Lasten zu vermeiden.

Für eine Produktivnahme der erstellten Komponente sollten zukünftig noch einige Bestandteile angepasst werden. So müsste unbedingt ein eigenes TLS-Zertifikat für *Unity IdM* ausgestellt und anschließend die SP-Metadaten beim DFN-Verein registriert werden, damit das Vertrauensverhältnis in der AAI nicht auf lokale Metadaten – wie im Testumfeld geschehen – beschränkt bleibt. Außerdem müssten umfangreichere Tests – auch mit anderen Mitgliedern der Föderation – durchgeführt werden. Zudem wäre es ratsam eine Überwachung des Synchronisationsdienstes und der beteiligten VMs durch *Monitoring-Tools* zu veranlassen, falls der Dienst ausfallen oder andere Komplikationen auftreten sollten. Ein redundanter Aufbau zur Lastverteilung und Ausfallsicherheit des zentralen Dienstes müsste auch in Zukunft angedacht werden. Zusätzliche Maßnahmen wie die Integration einer Mehr-Faktor-Authentifikation oder einer E-Mail-Bestätigung des Nutzers könnten in den nächsten Entwicklungsschritten ebenfalls interessant werden. Außerdem könnte eine Umstellung des Nutzernamens auf die E-Mail-Adresse einen größeren Komfort für den Endanwender haben, da dieser sich keinen neu generierten Anmeldenamen (s. „*uid*“ im Unterabschnitt 4.2.1) merken müsste.

Die im oberen Absatz genannten Maßnahmen / Probleme sind relativ überschaubar und mehr oder weniger einfach umzusetzen. Trotzdem gibt es noch zwei gravierendere Problematiken, die es zukünftig zu lösen gilt. Die erste Fragestellung zielt auf die Laufzeit eines durch den Proxy erstellten lokalen Kontos ab. Es muss gewährleistet sein, dass die digitale Identität in dem jeweiligen Heimat-IdP zur Laufzeit der lokalen Referenz (Benutzerkonto am AWI) stetig vorhanden ist. In der DFN-AAI existieren keine Benachrichtigungen über die Löschung oder das Deaktivieren eines föderativen Nutzerkontos; eine SSO-Infrastruktur sieht ein solches Verhalten auch gar nicht vor. Ein Lösungsansatz, der sich anbieten würde, wäre das zusätzliche Abspeichern eines *Session-Timeouts* zu jeder lokalen Identitätsreferenz. Nach Überschreiten des Zeitstempels würde das Konto deaktiviert werden und eine erneute Anmeldung und Verifikation am *SP-IdP-Proxy* nach sich ziehen. Das „richtige“ Setzen des Zeitraums – beispielsweise 24 Stunden – liegt dabei im Ermessensspielraum des Proxybetreibers. Das zweite Problem stellen die Gruppenberechtigungen dar. Nicht alle Nutzer sollen in einer AAI auf „alles“ Zugriff erlangen. Hierfür müssen die Konzepte anhand der die Berechtigungen gesetzt werden, ausgebaut und in der Föderation vereinheitlicht werden. Momentan wird für eine Autorisierung in der Föderation das Attribut „*eduPersonEntitlement*“ aus der Objektklasse „*eduPerson*“ eingesetzt [IT16]. Die Berechtigungen müssen am Proxy anhand von Attributen gesetzt und anschließend verteilt werden.

Letztendlich zeigt diese Bachelorarbeit einen Ansatz, wie nur lokal zugängliche Ressourcen – losgelöst vom Web-SSO – in einem Föderationsumfeld bereitgestellt werden können. Gewiss gibt es bei der implementierten Lösung Verbesserungs- und Diskussionsbedarf in der AAI, was die vorherigen zwei Absätze verdeutlichen. Des Weiteren wird eine lokale Referenz (Benutzerkonto) zu einer föderativen digitalen Identität erzeugt, wodurch das Konzept des *Single Sign-on* (SSO) vernachlässigt wird. Es ist zu hoffen, dass in absehbarer Zeit durch die Verwendung des Protokolls *OpenID Connect* (OIDC) in der DFN-AAI [Pem18a, vgl. S. 51] oder anderen Entwicklungen in diesem verteilten Kontext auch SSO realisiert werden kann. Die Nutzung des zentralen *SP-IdP-Proxys* im HDF-Umfeld, über den mehrere Einrichtungen die Autorisierung ihrer Dienste durchführen, steht ebenfalls im Raum.

Literaturverzeichnis

[ABB+18] Atherton, Christopher J. ; Barton, Thomas ; Basney, Jim ; Broeder, Daan ; Costa, Alessandro ; Daalen, Mirjam V. ; Dyke, Stephanie ; Elbers, Willem ; Enell, Carl-Fredrik ; Fasanelli, Enrico Maria V. ; Fernandes, João ; Florio, Licia ; Gietz, Peter ; Groep, David L. ; Junker, Matthias B. ; Kanellopoulos, Christos ; Kelsey, David ; Kershaw, Philip ; Knapic, Cristina ; Kollegger, Thorsten ; Koranda, Scott ; Linden, Mikael ; Marinic, Filip ; Matyska, Ludek ; Nyrönen, Tommi H. ; Paetow, Stefan ; Paglione, Laura A D. ; Parlati, Sandra ; Phillips, Christopher ; Prochazka, Michal ; Rees, Nicholas ; Short, Hannah ; Stevanovic, Uros ; Tartakovsky, Michael ; Venekamp, Gerben ; Vitez, Tom ; Wartel, Romain ; Whalen, Christopher ; White, John ; Zwölf, Carlo M.: Federated Identity Management For Research Collaborations. In: *FIM4R* (2018), Juli. http://dx.doi.org/10.5281/zenodo.1307551. – DOI 10.5281/zenodo.1307551. – abgerufen am 10.10.2018

[AT17] AWI-Team: *Eis. Meer. Klima. Das Alfred-Wegener-Institut (AWI).* https://www.awi.de/ueber-uns/organisation/profil.html. Version: Dezember 2017. – abgerufen am 01.10.2018

[AT18] Auth0-Team: *Get Started with JSON Web Tokens.* https://auth0.com/learn/json-web-tokens/. Version: 2018. – abgerufen am 28.10.2018

[Ben13] Benedyczak, Krzysztof: *UNIfied identiTY management.* http://www.unity-idm.eu/wp-content/uploads/2016/07/unity-fzj-2013.pdf. Version: Juli 2013. – abgerufen am 01.10.2018

[BJK+13] Broeder, Daan ; Jones, Bob ; Kelsey, David ; Kershaw, Philip ; Lüders, Stefan ; Lyall, Andrew ; Nyrönen, Tommi ; Wartel, Romain ; Weyer, Heinz J.: Federated Identity Management for Research Collaborations. In: *CERN-OPEN-2012-006* (2013), August. https://cdsweb.cern.ch/record/1442597/files/CERN-OPEN-2012-006.pdf. – abgerufen am 01.10.2018

[Bur18] Burgey, Lukas: Portal for Federated User Management in the Helmholtz Data Federation. In: *Bachelor Thesis, Karlsruher Institut für Technologie* (2018)

[Can05] Cantor, Scott: Shibboleth Architecture - Protocols and Profiles / Internet2. Version: September 2005. https://wiki.shibboleth.net/confluence/download/attachments/2162702/internet2-mace-shibboleth-arch-protocols-200509.pdf. 2005. – techreport. – abgerufen am 01.10.2018

[Die16] Diez, Maximilian: Usable Security Models for the Internet of Things and Hybrid Cloud Solutions. In: *Ubiquitäre Systeme (Seminar) und Mobile Computing (Proseminar) SS 2016. Mobile und Verteilte Systeme. Ubiquitous Computing. Teil XIV. Hrsg.: M.A. Neumann* Bd. 2016, Karlsruhe, 2016 (Karlsruhe Reports in Informatics 13). – ISSN 2190–4782, 92-110. – abgerufen am 01.10.2018

[DT18] DFN-Team ; DFN (Hrsg.): *Online-Dokumentation der DFN Trust und Identity Dienste DFN-AAI, eduroam und DFN PKI.* DFN, 2018. https://wiki.aai.dfn.de/:de:start. – abgerufen am 01.10.2018

[Eck18] Eckert, Claudia: *IT-Sicherheit: Konzepte - Verfahren - Protokolle*. Walter de Gruyter GmbH, 2018 (De Gruyter Studium). https://www.degruyter.com/viewbooktoc/product/490352. – ISBN 978–3–11–056390–0. – abgerufen am 07.10.2018

[Eur15] Europäische Kommission: *The 7th Framework Programme funded European Research and Technological Development from 2007 until 2013*. http://collections.internetmemory.org/haeu/20161215122208/http://cordis.europa.eu/fp7/. Version: November 2015. – abgerufen am 01.10.2018

[Fla18] Flanagan, Heather: *SCHAC - SCHema for ACademia*. https://wiki.refeds.org/display/STAN/SCHAC. Version: Mai 2018. – abgerufen am 26.10.2018

[Goo00] Good, Gordon: *The LDAP Data Interchange Format (LDIF) - Technical Specification*. RFC 2849. http://dx.doi.org/10.17487/RFC2849. Version: Juni 2000 (Request for Comments). – abgerufen am 27.10.2018

[Hö11] Höllrigl, Thorsten: *Informationskonsistenz im föderativen Identitätsmanagement: Modellierung und Mechanismen*, Karlsruher Institut für Technologie (KIT), Diss., 2011. http://dx.doi.org/10.5445/IR/1000022470. – DOI 10.5445/IR/1000022470. – abgerufen am 15.10.2018

[Har12] Hardt, Dick: *The OAuth 2.0 Authorization Framework*. RFC 6749. http://dx.doi.org/10.17487/RFC6749. Version: Oktober 2012 (Request for Comments). – abgerufen am 27.10.2018

[HCH+05] Hughes, John ; Cantor, Scott ; Hodges, Jeff ; Hirsch, Frederick ; Mishra, Prateek ; Philpott, Rob ; Maler, Eve: Profiles for the OASIS Security Assertion Markup Language (SAML) V2.0 / OASIS. Version: März 2005. https://docs.oasis-open.org/security/saml/v2.0/saml-profiles-2.0-os.pdf. 2005. – techreport. – abgerufen am 25.10.2018

[Hr11] Hansen, Tony ; 3rd, Donald E. E.: *US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)*. RFC 6234. http://dx.doi.org/10.17487/RFC6234. Version: Mai 2011 (Request for Comments). – abgerufen am 11.11.2018

[HT18] HDF-Team: *Helmholtz-Datenföderation (HDF)*. https://www.helmholtz.de/forschung/information_data_science/helmholtz_data_federation/. Version: 2018. – abgerufen am 01.10.2018

[IT16] Internet2-Team: eduPerson Object Class Specification (201602) / Internet2. Version: März 2016. http://software.internet2.edu/eduperson/internet2-mace-dir-eduperson-201602.html. 2016. – Forschungsbericht. – abgerufen am 26.10.2018

[JBS15] Jones, Michael ; Bradley, John ; Sakimura, Nat: *JSON Web Token (JWT)*. RFC 7519. http://dx.doi.org/10.17487/RFC7519. Version: Mai 2015 (Request for Comments). – abgerufen am 27.10.2018

[JD08] Jochen Dinger, Hannes H.: *Netzwerk- und IT-Sicherheitsmanagement : eine Einführung*. Universitätsverlag Karlsruhe, Karlsruhe, 2008 https://publikationen.bibliothek.kit.edu/1000007400/142064. – ISBN 978–3–86644–209–2. – abgerufen am 06.10.2018

[KK11] Karunanithi, M. D. ; Kiruthika, B.: Single sign-on and single log out in identity. In: *International Conference on Nanoscience, Engineering and Technology (ICONSET 2011)*, 2011, 607-611. – abgerufen am 01.10.2018

[KLDS15] Kanellopoulos, Christos ; Liampotis, Nicolas ; Dijk, Niels van ; Solagna, Peter: Analysis of user community and service provider requirements. In: *DJRA1.1* (2015), Oktober. https://aarc-project.eu/wp-content/uploads/2015/10/AARC-DJRA1.1.pdf. – abgerufen am 06.10.2018

[LH13] Lodderstedt, Torsten ; Hiller, Jochen: *Flexible und sichere Internetdienste mit OAuth 2.0.* https://www.heise.de/developer/artikel/Flexible-und-sichere-Internetdienste-mit-OAuth-2-0-2068404.html?seite=all. Version: Dezember 2013. – abgerufen am 28.10.2018

[Lin18] Linux Programmer's Manual: *crypt, crypt_r - password and data encryption.* manpage. http://man7.org/linux/man-pages/man3/crypt.3.html. Version: April 2018. – abgerufen am 06.10.2018

[LSM05] Leach, Paul J. ; Salz, Rich ; Mealling, Michael H.: *A Universally Unique IDentifier (UUID) URN Namespace.* RFC 4122. http://dx.doi.org/10.17487/RFC4122. Version: Juli 2005 (Request for Comments). – abgerufen am 07.10.2018

[MD11] Mansfield-Devine, Steve: Single Sign-On: matching convenience with security. In: *Biometric Technology Today* 2011 (2011), Nr. 7, 7 - 11. http://dx.doi.org/https://doi.org/10.1016/S0969-4765(11)70134-5. – DOI https://doi.org/10.1016/S0969–4765(11)70134–5. – ISSN 0969–4765. – abgerufen am 01.10.2018

[MMS16] Mladenov, Vladislav ; Mainka, Christian ; Schwenk, Jörg: On the security of modern Single Sign-On Protocols: Second-Order Vulnerabilities in OpenID Connect. In: *N/A* (2016), Januar. https://arxiv.org/pdf/1508.04324v2.pdf. – abgerufen am 27.10.2018

[NA07] Ngo, L. ; Apon, A.: Using Shibboleth for Authorization and Authentication to the Subversion Version Control Repository System. In: *Information Technology, 2007. ITNG '07. Fourth International Conference on*, 2007, 760-765. – abgerufen am 01.10.2018

[OT18a] OpenID-Team: *OpenID Connect FAQ and Q&As.* https://openid.net/connect/faq/. Version: 2018. – abgerufen am 27.10.2018

[OT18b] OpenLDAP-Team: *OpenLDAP Software 2.4 Administrator's Guide*, März 2018. https://www.openldap.org/doc/admin24/. – abgerufen am 01.10.2018

[OT18c] OpenSSL-Team: *openssl manpage – commands.* https://www.openssl.org/docs/man1.1.0/apps/. Version: September 2018. – abgerufen am 27.10.2018

[Pem18a] Pempe, Wolfgang: *Attribute: Attribut-Schemata, -Generierung, -Übertragung und Verarbeitung am SP.* https://download.aai.dfn.de/ws/2018_fhdo/attributes.pdf. Version: August 2018. – abgerufen am 19.10.2018

[Pem18b] Pempe, Wolfgang: *Grundlagen: AAI, Web-SSO, Metadaten und Föderationen.* https://download.aai.dfn.de/ws/2018_fhdo/grundlagen.pdf. Version: August 2018. – abgerufen am 19.10.2018

[PJ16] Percival, Colin ; Josefsson, Simon: *The scrypt Password-Based Key Derivation Function.* RFC 7914. http://dx.doi.org/10.17487/RFC7914. Version: August 2016 (Request for Comments). – abgerufen am 06.10.2018

[RHP+08] Ragouzis, Nick ; Hughes, John ; Philpott, Rob ; Maler, Eve ; Madsen, Paul ; Scavo, Tom: Security Assertion Markup Language (SAML) V2.0 Technical Overview / OASIS. Version: März 2008. http://docs.oasis-open.org/security/saml/Post2.

0/sstc-saml-tech-overview-2.0-cd-02.pdf. 2008. – techreport. – abgerufen am 25.10.2018

[SBJ+14] Sakimura, N. ; Bradley, J. ; Jones, M. ; Medeiros, B. de ; Mortimore, C.: OpenID Connect Core 1.0 incorporating errata set 1 / OpenID Foundation. Version: November 2014. https://openid.net/specs/openid-connect-core-1_0.html. 2014. – techreport. – abgerufen am 27.10.2018

[Sci06] Sciberras, Andrew: *Lightweight Directory Access Protocol (LDAP): Schema for User Applications*. RFC 4519. http://dx.doi.org/10.17487/RFC4519. Version: Juni 2006 (Request for Comments). – abgerufen am 21.10.2018

[Ser06] Sermersheim, Jim: *Lightweight Directory Access Protocol (LDAP): The Protocol*. RFC 4511. http://dx.doi.org/10.17487/RFC4511. Version: Juni 2006 (Request for Comments). – abgerufen am 21.10.2018

[SFH+18] Scardaci, Diego ; Florio, Licia ; Hübner, David ; Jankowski, Michał ; Jensen, Jens ; Kanellopoulos, Christos ; Kouril, Daniel ; Liampotis, Nicolas ; Linden, Mikael ; Memon, Shiraz ; Salle, Mischa ; Terpstra, Arnout: Use-Cases for Interoperable Cross-Infrastructure AAI. In: *DJRA1.1* (2018), September. https://aarc-project.eu/wp-content/uploads/2018/09/AARC2-DJRA1.1-V3-v3FINAL.pdf. – abgerufen am 10.10.2018

[Smi00] Smith, Mark C.: *Definition of the inetOrgPerson LDAP Object Class*. RFC 2798. http://dx.doi.org/10.17487/RFC2798. Version: April 2000 (Request for Comments). – abgerufen am 21.10.2018

[ST18a] Shibboleth-Team ; Shibboleth (Hrsg.): *Shibboleth Manual*. Shibboleth, 2018. https://wiki.shibboleth.net/. – abgerufen am 01.10.2018

[ST18b] Shibboleth-Team: *What's Shibboleth?* https://www.shibboleth.net/index/. Version: 2018. – abgerufen am 01.10.2018

[TT18] Tarsnap-Team: *The scrypt key derivation function*. https://www.tarsnap.com/scrypt.html. Version: Oktober 2018. – abgerufen am 06.10.2018

[UT16] Unity-Team ; Unity (Hrsg.): *Unity SAML HowTo Manual*. Unity, April 2016. http://www.unity-idm.eu/documentation/unity-1.9.0/saml-howto.html. – abgerufen am 01.10.2018

[UT18a] Unity-Team ; Unity (Hrsg.): *RESTful Administration API*. Unity, August 2018. http://www.unity-idm.eu/documentation/unity-2.6.2/rest-api-v1.html. – abgerufen am 01.10.2018

[UT18b] Unity-Team ; Unity (Hrsg.): *Unity Manual*. Unity, August 2018. http://www.unity-idm.eu/documentation/unity-2.6.2/manual.html. – abgerufen am 01.10.2018

[VT18a] VMware-Team: *Konfigurieren einer OpenLDAP Directory-Verbindung*. https://docs.vmware.com/de/vRealize-Automation/7.4/com.vmware.vra.prepare.use.doc/GUID-2FE80EB5-BBC9-4AB0-B70F-34610E74FD1B.html. Version: Juli 2018. – abgerufen am 16.11.2018

[VT18b] VMware-Team ; VMware (Hrsg.): *VMware vRealize Automation Manual*. VMware, 2018. https://docs.vmware.com/de/vRealize-Automation/index.html. – abgerufen am 01.10.2018

[WC08] Widdowson, Rod ; Cantor, Scott: Identity Provider Discovery Service Protocol and Profile / OASIS. Version: März 2008. `http://docs.oasis-open.org/security/saml/Post2.0/sstc-saml-idp-discovery.pdf`. 2008. – techreport. – abgerufen am 25.10.2018

[Zei06a] Zeilenga, Kurt: *Lightweight Directory Access Protocol (LDAP): Directory Information Models*. RFC 4512. `http://dx.doi.org/10.17487/RFC4512`. Version: Juni 2006 (Request for Comments). – abgerufen am 21.10.2018

[Zei06b] Zeilenga, Kurt: *Lightweight Directory Access Protocol (LDAP): Technical Specification Road Map*. RFC 4510. `http://dx.doi.org/10.17487/RFC4510`. Version: Juni 2006 (Request for Comments). – abgerufen am 21.10.2018

[ZM06] Zeilenga, Kurt ; Melnikov, Alexey: *Simple Authentication and Security Layer (SASL)*. RFC 4422. `http://dx.doi.org/10.17487/RFC4422`. Version: Juni 2006 (Request for Comments). – abgerufen am 21.10.2018

A Anhang

A.1 SAML-Kommunikation

In den nachstehenden Listings wird eine SAML-Kommunikation zwischen der *Relying Party* „*unitysrv1.awi.de*“ und der *Asserting Party* „*shib-idp2.awi.de*“ deutlich. Für das Mitschneiden der Interaktion wurde das *Firefox-Plugin* „*SAML-tracer*“ von Olav Morken in der Version 1.5.1 verwendet.

Das Listing A.1 zeigt die Authentifizierungsanfrage (*AuthnRequest*) des SP „*unitysrv1.awi.de*“ an den Test-IdP „*shib-idp2.awi.de*“. Im Listing A.2 wird daraufhin die SAML-Antwort (*Response*) des IdP „*shib-idp2.awi.de*“ dargestellt. Aus Demonstrationszwecken wurde die SAML-*Assertion* entschlüsselt und liegt somit in Klartext vor. Zuvor war dieser Bereich der XML-Datei durch *XML-Encryption* („*saml2:EncryptedAssertion*“) nicht zugänglich.

```
<urn:AuthnRequest IssueInstant="2018-10-20T14:00:51.036Z"
                  ID="
    SAMLY2lib_msg_3723f93554f1d3c3bf08dfa6432cd911a6221db9745d86a7"
                  Version="2.0"
                  Destination="https://shib-idp2.awi.de/idp/profile/SAML2/Redirect
    /SSO"
                  AssertionConsumerServiceURL="https://unitysrv1.awi.de:443/
    unitygw/spSAMLResponseConsumer"
                  xmlns:urn="urn:oasis:names:tc:SAML:2.0:protocol"
                  >
  <urn1:Issuer Format="urn:oasis:names:tc:SAML:2.0:nameid-format:entity"
               xmlns:urn1="urn:oasis:names:tc:SAML:2.0:assertion"
               >https://unitysrv1.awi.de</urn1:Issuer>
  <urn:NameIDPolicy AllowCreate="true" />
</urn:AuthnRequest>
```

Listing A.1: saml_authn_request.xml

```
<saml2p:Response xmlns:saml2p="urn:oasis:names:tc:SAML:2.0:protocol" Destination="
    https://unitysrv1.awi.de:443/unitygw/spSAMLResponseConsumer" ID="
    _bae2d89e81e43be6ec12eecc31c5679a" InResponseTo="
    SAMLY2lib_msg_3723f93554f1d3c3bf08dfa6432cd911a6221db9745d86a7" IssueInstant="
    2018-10-20T14:09:09.410Z" Version="2.0">
  <saml2:Issuer xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion">https://shib-
    idp2.awi.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/2001/04/xmldsig-more#rsa-
    sha256"/>
      <ds:Reference URI="#_bae2d89e81e43be6ec12eecc31c5679a">
        <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/2001/04/xmlenc#sha256"/>
        <ds:DigestValue>...</ds:DigestValue>
      </ds:Reference>
    </ds:SignedInfo>
    <ds:SignatureValue>...</ds:SignatureValue>
    <ds:KeyInfo>
      <ds:X509Data>
        <ds:X509Certificate>...</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:Assertion xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion" ID="
    _a94b98047f2dc07f3ff1b955cecffc50" IssueInstant="2018-10-20T14:09:09.410Z"
    Version="2.0">
    <saml2:Issuer>https://shib-idp2.awi.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/2001/04/xmldsig-more#rsa-
    sha256"/>
        <ds:Reference URI="#_a94b98047f2dc07f3ff1b955cecffc50">
          <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/2001/04/xmlenc#sha256"/>
          <ds:DigestValue>...</ds:DigestValue>
        </ds:Reference>
      </ds:SignedInfo>
      <ds:SignatureValue>...</ds:SignatureValue>
      <ds:KeyInfo>
        <ds:X509Data>
          <ds:X509Certificate>...</ds:X509Certificate>
        </ds:X509Data>
      </ds:KeyInfo>
    </ds:Signature>
    <saml2:Subject>
      <saml2:NameID Format="urn:oasis:names:tc:SAML:2.0:nameid-format:persistent"
    NameQualifier="https://shib-idp2.awi.de/idp/shibboleth" SPNameQualifier="
    https://unitysrv1.awi.de">SWPEbjnFUQcPOwe0eU5/wNcxgKg=</saml2:NameID>
      <saml2:SubjectConfirmation Method="urn:oasis:names:tc:SAML:2.0:cm:bearer">
        <saml2:SubjectConfirmationData Address="..." InResponseTo="
    SAMLY2lib_msg_3723f93554f1d3c3bf08dfa6432cd911a6221db9745d86a7" NotOnOrAfter="
    2018-10-20T14:14:09.461Z" Recipient="https://unitysrv1.awi.de:443/unitygw/
    spSAMLResponseConsumer"/>
      </saml2:SubjectConfirmation>
    </saml2:Subject>
    <saml2:Conditions NotBefore="2018-10-20T14:09:09.410Z" NotOnOrAfter="
    2018-10-20T14:14:09.410Z">
      <saml2:AudienceRestriction>
        <saml2:Audience>https://unitysrv1.awi.de</saml2:Audience>
      </saml2:AudienceRestriction>
    </saml2:Conditions>
    <saml2:AuthnStatement AuthnInstant="2018-10-20T14:08:18.569Z" SessionIndex="
    _9bd53c9f0fc8a2255966ae187d4b36a5">
```

```
60        <saml2:SubjectLocality Address="..."/>
          <saml2:AuthnContext>
62          <saml2:AuthnContextClassRef>urn:oasis:names:tc:SAML:2.0
      :ac:classes:PasswordProtectedTransport</saml2:AuthnContextClassRef>
          </saml2:AuthnContext>
64      </saml2:AuthnStatement>
        <saml2:AttributeStatement>
66        <saml2:Attribute FriendlyName="cn" Name="urn:oid:2.5.4.3" NameFormat="
      urn:oasis:names:tc:SAML:2.0:attrname-format:uri">
            <saml2:AttributeValue>Fabian Mangels</saml2:AttributeValue>
68        </saml2:Attribute>
          <saml2:Attribute FriendlyName="mail" Name="urn:oid:0.9.2342.19200300.100.1.3
      " NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri">
70          <saml2:AttributeValue>fabian.mangels@awi.de</saml2:AttributeValue>
          </saml2:Attribute>
72        <saml2:Attribute FriendlyName="displayName" Name="urn:oid:2
      .16.840.1.113730.3.1.241" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-
      format:uri">
            <saml2:AttributeValue>Fabian Mangels</saml2:AttributeValue>
74        </saml2:Attribute>
          <saml2:Attribute FriendlyName="sn" Name="urn:oid:2.5.4.4" NameFormat="
      urn:oasis:names:tc:SAML:2.0:attrname-format:uri">
76          <saml2:AttributeValue>Mangels</saml2:AttributeValue>
          </saml2:Attribute>
78        <saml2:Attribute FriendlyName="givenName" Name="urn:oid:2.5.4.42" NameFormat
      ="urn:oasis:names:tc:SAML:2.0:attrname-format:uri">
            <saml2:AttributeValue>Fabian</saml2:AttributeValue>
80        </saml2:Attribute>
        </saml2:AttributeStatement>
82    </saml2:Assertion>
</saml2p:Response>
```

Listing A.2: saml_response.xml

A.2 DFN-AAI

Abbildung A.1: DFN-AAI – Dienste und Nutzergruppen [Pem18b, S. 8]

Die DFN-AAI ermöglicht es auf nationaler Ebene (Deutschland) Identitätsverteilungen zwischen verschiedenen Einrichtungen vorzunehmen und so föderativ Dienste miteinander zu teilen bzw.

bereitzustellen. Die Abbildung A.1 zeigt einige Dienste und die dazugehörigen Nutzergruppen im DFN-AAI-Umfeld. Dabei sind die Dienste der DFN-AAI zeitlich seit dem offiziellen Betrieb im Oktober 2007 bis heute angeordnet.

A.3 eduGAIN

eduGAIN ist eine auf der europäischen Ebene agierende Föderation, die es national tätigen Föderationen ermöglicht auch weltweit eine AAI zu betreiben. Das Diagramm in der Abbildung A.2

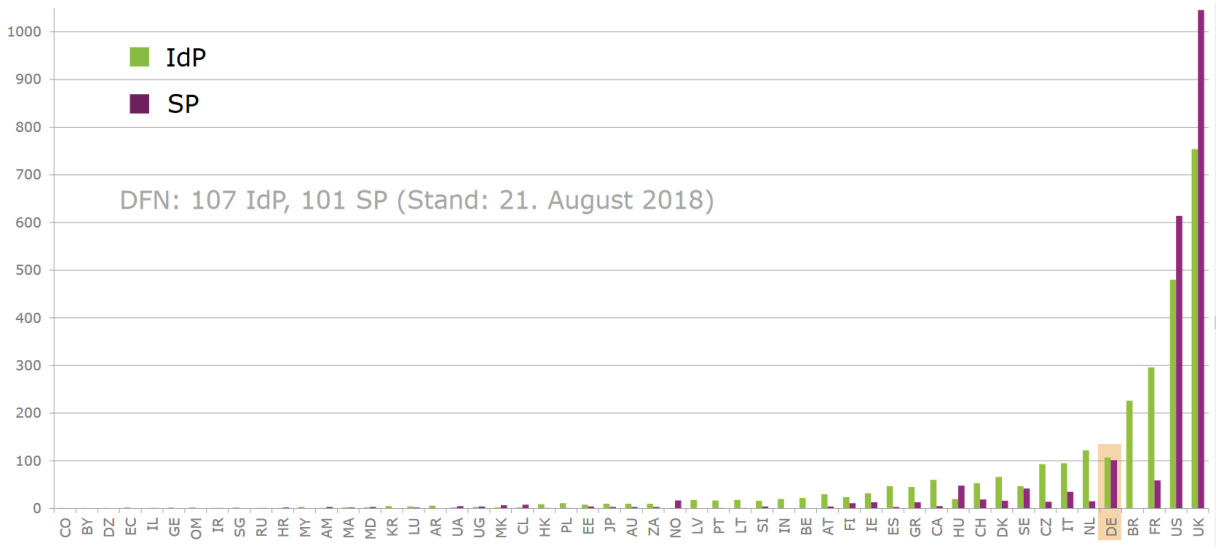

Abbildung A.2: *eduGAIN* – Beteiligung je Föderation [Pem18b, S. 38]

zeigt die Beteiligung je Föderation in *eduGAIN*. Sowohl die Briten als auch die Vereinigten Staaten stechen hier besonders hervor. Deutschland nimmt den fünften Platz mit jeweils ungefähr hundert IdPs und SPs ein. Hier werden nur Entitäten gezählt, die tatsächlich in dem *eduGAIN*-Verbund freigegeben sind. Daher gibt es einen deutlichen Unterschied zwischen den Anzahlen in diesem Diagramm und in der Abbildung 3.5.

A.4 Shibboleth

Das abgebildete Listing A.3 stellt die Übersetzungsregeln für Attribute zwischen Identitätsanbieter und Identitätsspeicher dar. Im Falle einer SAML-Kommunikation werden hier Attribute aus dem LDAP-Verzeichnis in gültige SAML-Attribute übersetzt oder ggf. auch zusammengesetzt. Das Listing A.4 enthält die komplette Filterrichtliniendatei, die bei der Attributfreigabe an die Entität (*Requester*) „*unitysrv1.awi.de*“ Anwendung findet.

```
<?xml version="1.0" encoding="UTF-8"?>
<AttributeResolver xmlns="urn:mace:shibboleth:2.0:resolver" xmlns:xsi="http://www.
    w3.org/2001/XMLSchema-instance" xsi:schemaLocation="urn:mace:shibboleth:2.0
    :resolver http://shibboleth.net/schema/idp/shibboleth-attribute-resolver.xsd">

  <AttributeDefinition id="mail" xsi:type="Simple" sourceAttributeID="mail">
    <Dependency ref="myLDAP" />
    <DisplayName xml:lang="en">E-mail</DisplayName>
    <DisplayName xml:lang="de">E-Mail</DisplayName>
    <DisplayDescription xml:lang="en">E-mail address</DisplayDescription>
    <DisplayDescription xml:lang="de">E-Mail Adresse</DisplayDescription>
    <AttributeEncoder xsi:type="SAML1String" name="urn:mace:dir:attribute-def:mail
    " encodeType="false" />
    <AttributeEncoder xsi:type="SAML2String" name="urn:oid:0
    .9.2342.19200300.100.1.3" friendlyName="mail" encodeType="false" />
  </AttributeDefinition>

  <AttributeDefinition id="surname" xsi:type="Simple" sourceAttributeID="sn">
    <Dependency ref="myLDAP" />
    <DisplayName xml:lang="en">Surname</DisplayName>
    <DisplayName xml:lang="de">Nachname</DisplayName>
    <DisplayDescription xml:lang="en">Surname or family name</DisplayDescription>
    <DisplayDescription xml:lang="de">Familienname des Nutzers bzw. der Nutzerin</
    DisplayDescription>
    <AttributeEncoder xsi:type="SAML1String" name="urn:mace:dir:attribute-def:sn"
    encodeType="false" />
    <AttributeEncoder xsi:type="SAML2String" name="urn:oid:2.5.4.4" friendlyName="
    sn" encodeType="false" />
  </AttributeDefinition>

  <AttributeDefinition id="givenName" xsi:type="Simple" sourceAttributeID="
    givenName">
    <Dependency ref="myLDAP" />
    <DisplayName xml:lang="en">Given name</DisplayName>
    <DisplayName xml:lang="de">Vorname</DisplayName>
    <DisplayDescription xml:lang="en">Given name of a person</DisplayDescription>
    <DisplayDescription xml:lang="de">Vorname des Nutzers bzw. der Nutzerin</
    DisplayDescription>
    <AttributeEncoder xsi:type="SAML1String" name="urn:mace:dir:attribute-
    def:givenName" encodeType="false" />
    <AttributeEncoder xsi:type="SAML2String" name="urn:oid:2.5.4.42" friendlyName=
    "givenName" encodeType="false" />
  </AttributeDefinition>

  <AttributeDefinition id="commonName" xsi:type="Simple" sourceAttributeID="cn">
    <Dependency ref="myLDAP"/>
    <DisplayName xml:lang="en">Common name</DisplayName>
    <DisplayName xml:lang="de">Gebraeuchlicher Name</DisplayName>
    <DisplayDescription xml:lang="en">Common name of a person</DisplayDescription>
    <DisplayDescription xml:lang="de">Gebraeuchlicher Name des Nutzers bzw. der
    Nutzerin</DisplayDescription>
```

```
    <AttributeEncoder xsi:type="SAML1String" name="urn:mace:dir:attribute-def:cn"
    encodeType="false"/>
41    <AttributeEncoder xsi:type="SAML2String" name="urn:oid:2.5.4.3" friendlyName="
    cn" encodeType="false"/>
  </AttributeDefinition>
43
  <AttributeDefinition id="displayName" xsi:type="Simple" sourceAttributeID="
    displayName">
45    <Dependency ref="myLDAP"/>
    <DisplayName xml:lang="en">Display name</DisplayName>
47    <DisplayName xml:lang="de">Anzeigename</DisplayName>
    <DisplayDescription xml:lang="en">Display name of a person</DisplayDescription
    >
49    <DisplayDescription xml:lang="de">Anzeigename des Nutzers bzw. der Nutzerin</
    DisplayDescription>
    <AttributeEncoder xsi:type="SAML1String" name="urn:mace:dir:attribute-
    def:displayName" encodeType="false"/>
51    <AttributeEncoder xsi:type="SAML2String" name="urn:oid:2
    .16.840.1.113730.3.1.241" friendlyName="displayName" encodeType="false"/>
  </AttributeDefinition>
53
  <DataConnector id="myLDAP" xsi:type="LDAPDirectory"
55    ldapURL="%{idp.attribute.resolver.LDAP.ldapURL}"
    baseDN="%{idp.attribute.resolver.LDAP.baseDN}">
57    <FilterTemplate>
      <![CDATA[
59        %{idp.attribute.resolver.LDAP.searchFilter}
      ]]>
61    </FilterTemplate>
  </DataConnector>
63</AttributeResolver>
```

Listing A.3: attribute_resolver.xml

```
1<?xml version="1.0" encoding="UTF-8"?>
<AttributeFilterPolicyGroup xmlns="urn:mace:shibboleth:2.0:afp" xmlns:xsi="http://
    www.w3.org/2001/XMLSchema-instance" id="ShibbolethFilterPolicy"
    xsi:schemaLocation="urn:mace:shibboleth:2.0:afp http://shibboleth.net/schema/
    idp/shibboleth-afp.xsd">
3
  <AttributeFilterPolicy id="unity">
5    <PolicyRequirementRule xsi:type="OR">
      <Rule xsi:type="Requester" value="https://unitysrv1.awi.de" />
7    </PolicyRequirementRule>
    <AttributeRule attributeID="mail" permitAny="true" />
9    <AttributeRule attributeID="surname" permitAny="true" />
    <AttributeRule attributeID="givenName" permitAny="true" />
11    <AttributeRule attributeID="commonName" permitAny="true" />
    <AttributeRule attributeID="displayName" permitAny="true" />
13  </AttributeFilterPolicy>
</AttributeFilterPolicyGroup>
```

Listing A.4: attribute_filter.xml

Die Abbildung A.3 zeigt die Anmeldemaske am *Shibboleth* IdP, an dem im dargestellten Szenario die Authentifizierung des Nutzers stattfindet. Ist die Anmeldung erfolgreich, wird ein Dialogfenster über die Zustimmung zur Attributfreigabe dargestellt, welches in der Abbildung A.4 zu sehen ist.

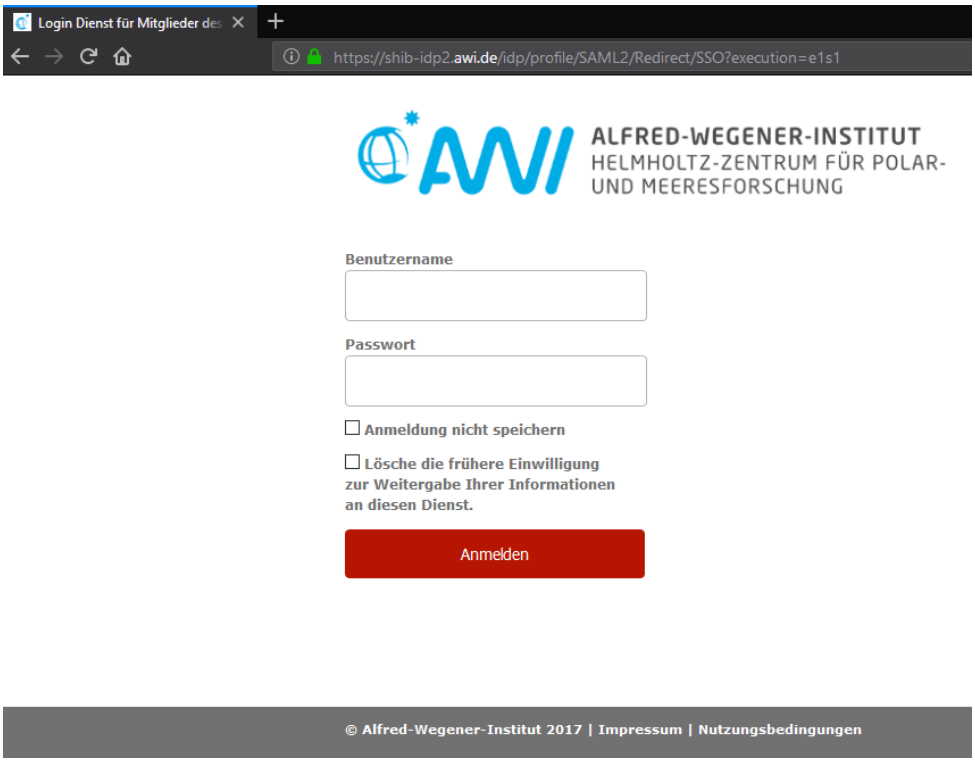

Abbildung A.3: Anmeldemaske des *Shibboleth* IdPs

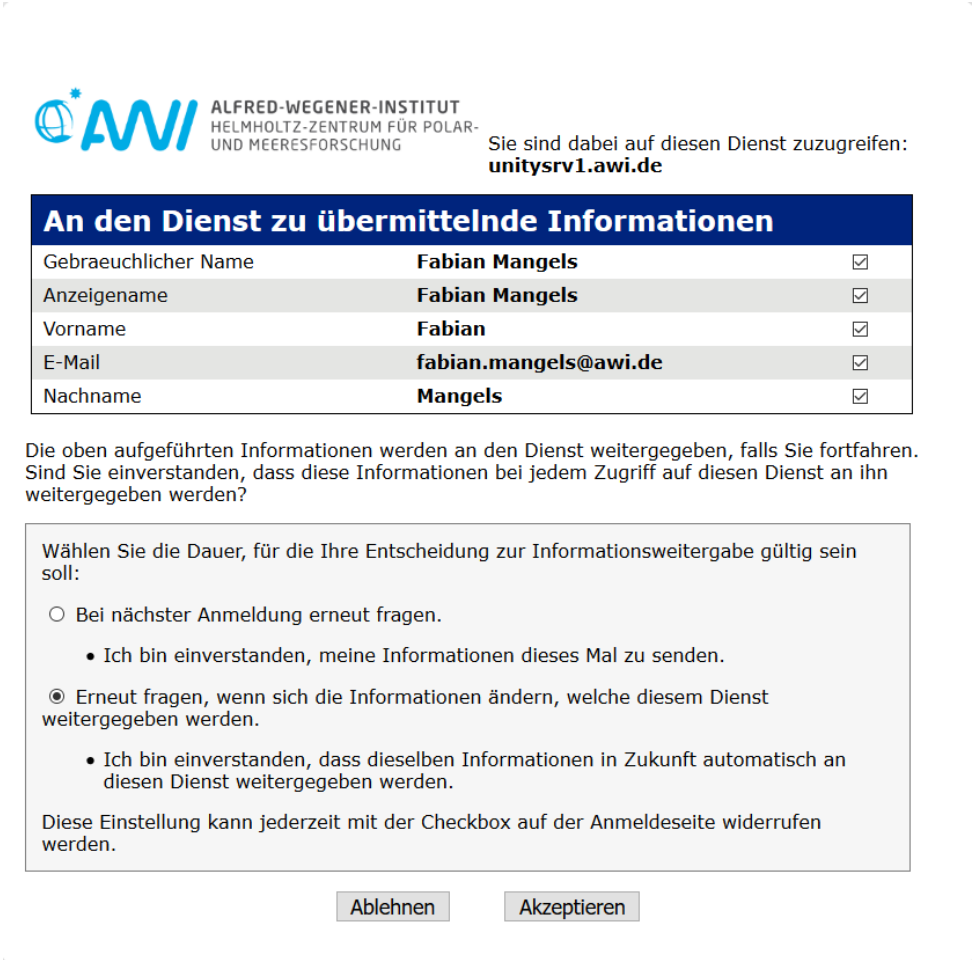

Abbildung A.4: Zustimmung zur Attributfreigabe am *Shibboleth* IdP

A.5 Unity IdM

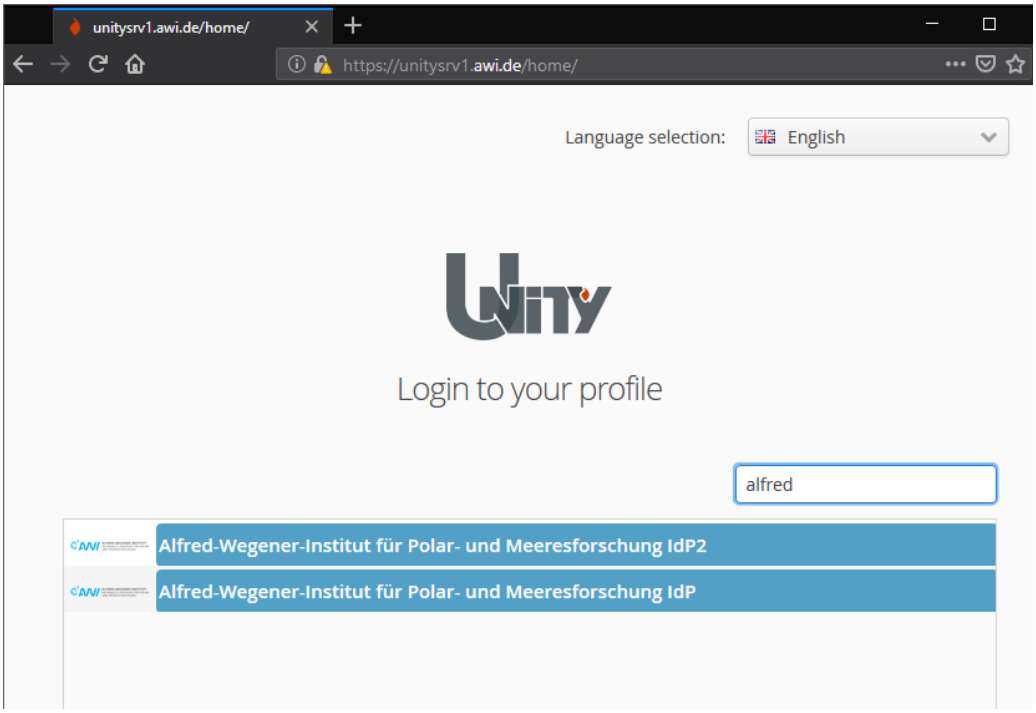

Abbildung A.5: Anmeldemaske des Nutzerprofil-Endpunkts

Die Abbildung A.5 zeigt die Anmeldemaske des Nutzerprofil-Endpunkts von *Unity IdM*. Hier kann der Nutzer den Identitätsanbieter seiner Heimateinrichtung in einer vordefinierten Liste auswählen und damit die Authentifizierung starten. Nach erfolgreicher Anmeldung wird bei erstmaligem Zugriff ein Formular aufgerufen (s. Abbildung A.6), welches vom Nutzer ausgefüllt werden muss. Hier wird ein lokales Passwort erstellt, welches der Entität zugeordnet wird.

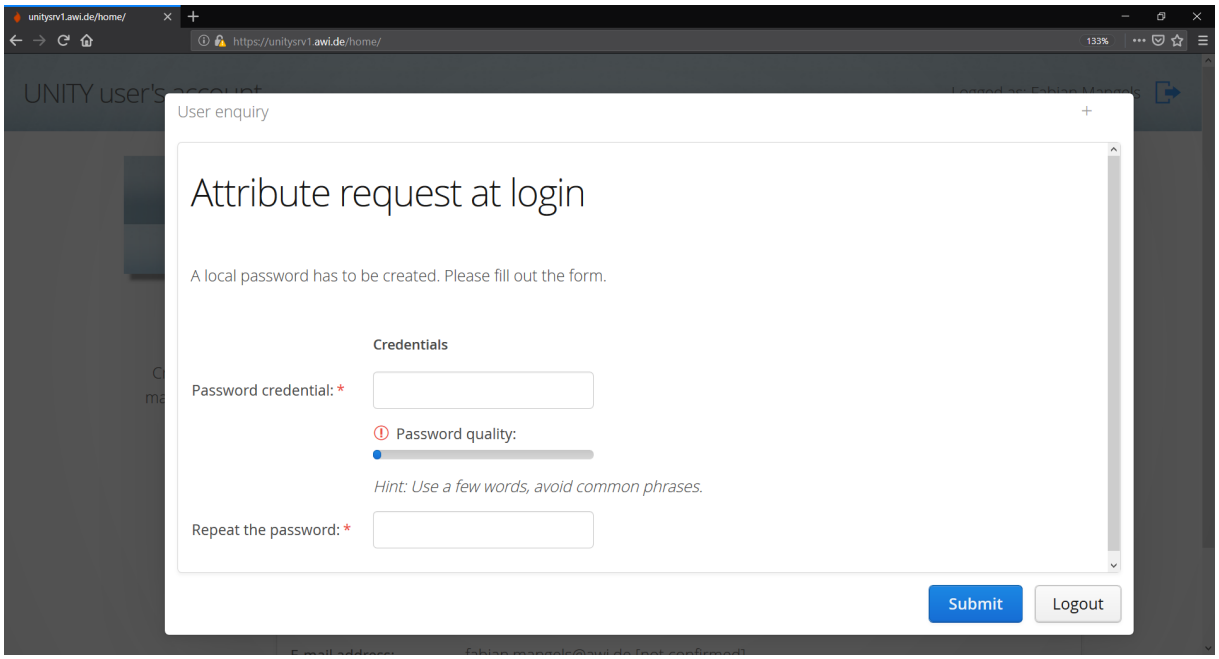

Abbildung A.6: Formularaufruf am Nutzerprofil-Endpunkt

Werden die Eingaben erfolgreich vom Nutzer bestätigt, gelangt dieser zur Übersichtsseite, auf der alle gespeicherten Attribute aufgezeigt werden; dies wird in der Abbildung A.7 dargestellt. Der zweite Reiter – zu sehen in Abbildung A.8 – bietet die Möglichkeit an, das Passwort zu ändern.

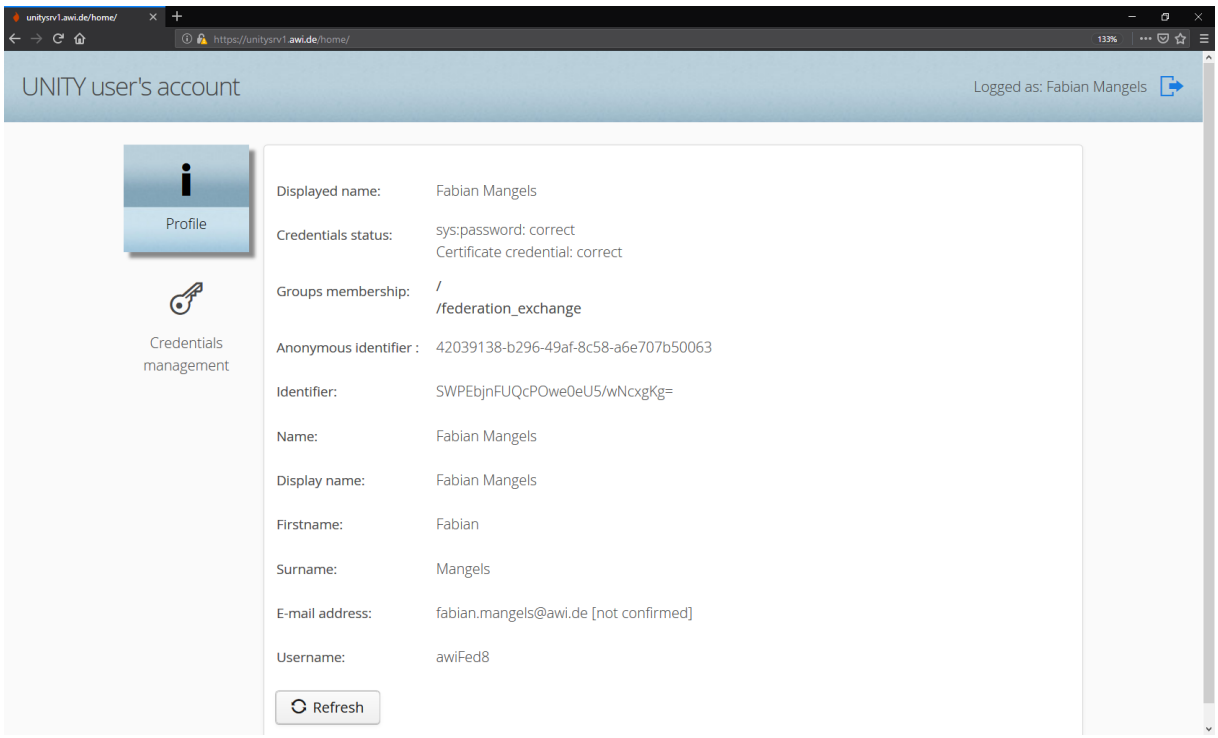

Abbildung A.7: Übersicht gespeicherter Attribute am Nutzerprofil-Endpunkt

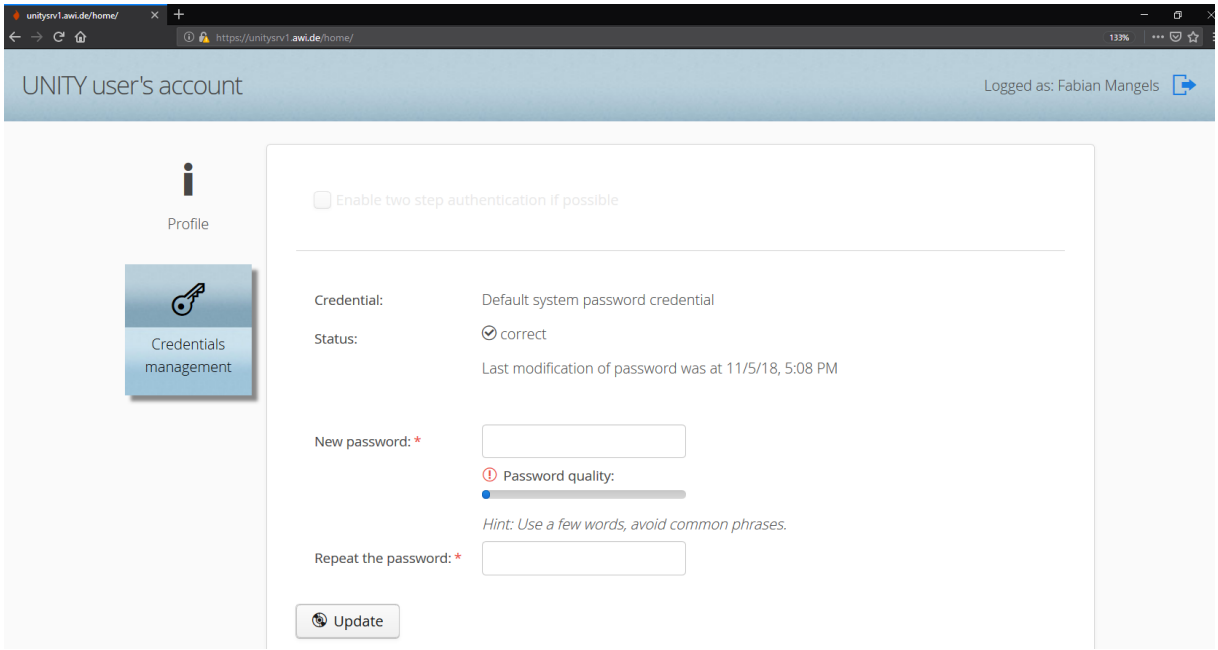

Abbildung A.8: Passwortänderung am Nutzerprofil-Endpunkt

In der Abbildung A.9 ist die Startseite der Administrationsoberfläche des Admin-Endpunkts dargestellt. Von hier aus können vorhandene Nutzerkonten und Gruppen verwaltet werden.

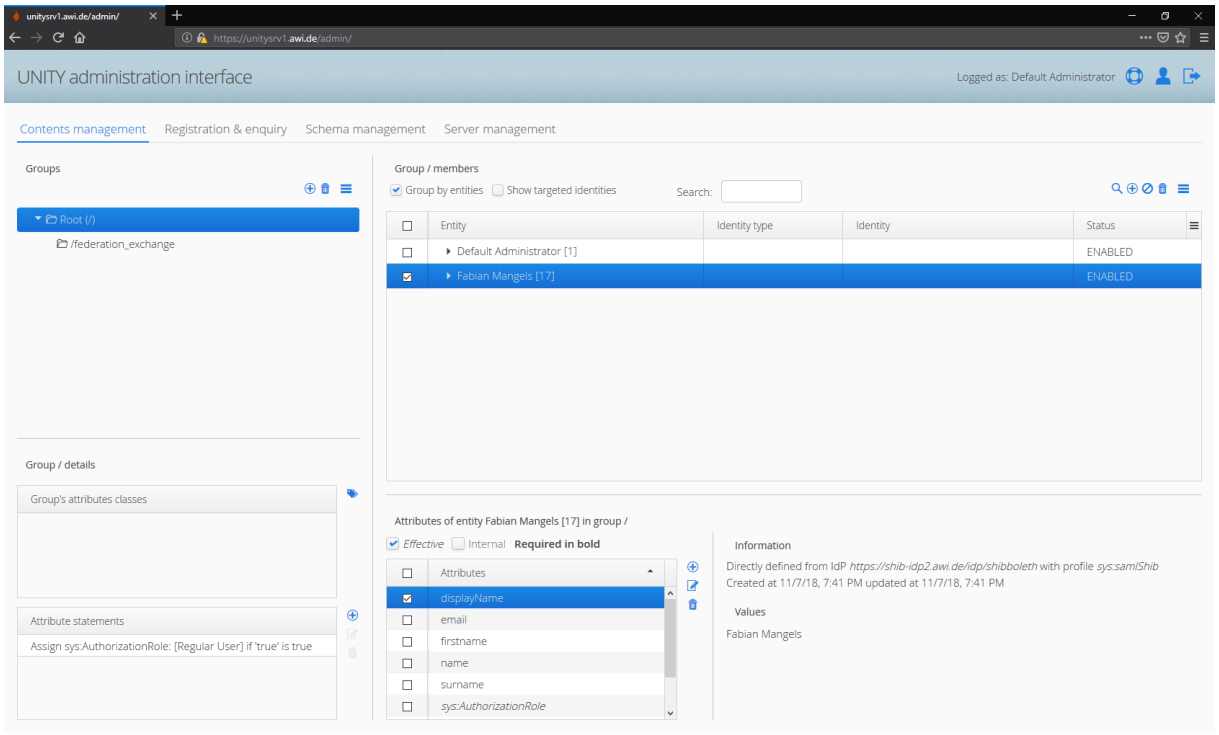

Abbildung A.9: Einstiegspunkt der Weboberfläche des Admin-Endpunkts

Außerdem sind viele weitere Einstellungsmöglichkeiten zu den einzelnen Endpunkten und Authentifizierungscontrollern über diverse Unterseiten möglich. Es können u. a. neue Attribut-Objekte, Formulare oder auch *Translation Profiles* erschafft werden.

```
/*
 * Copyright (c) 2017 ICM Uniwersytet Warszawski All rights reserved.
 * See LICENCE.txt file for licensing information.
 */
package pl.edu.icm.unity.stdext.credential.pass;

import java.nio.charset.StandardCharsets;
import java.security.SecureRandom;
import java.util.Map;
import java.util.Random;

import org.apache.commons.codec.digest.Crypt;
import org.bouncycastle.crypto.digests.SHA256Digest;
import org.bouncycastle.crypto.generators.SCrypt;
import org.bouncycastle.util.Arrays;

/**
 * Low level password handling. Allows for initial obfuscation of a given
 * password (PasswordInfo is generated, ready to be stored in DB) and for
 * checking a given password against the one loaded.
 *
 * @author K. Benedyczak & F. Mangels
 */
public class PasswordEngine {
  private static final int SALT_LENGTH = 32;
  private Random random = new SecureRandom();

  /**
   * Table with characters for Base64 transformation.
```

```
30     */
       private static final String B64T = "./0123456789
         ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz";
32
       /**
34      * Salt value position in the crypt string.
        */
36     private static final int CRYPT_SALT_POS = 3;

38     /**
        * Hash value position in the crypt string.
40      */
       private static final int CRYPT_HASH_POS = 4;
42
       /**
44      * Default Hash method is now CRYPT with SHA-256 &lt;crypt(3)&gt;.
        *
46      * @param credentialSettings - contains all parameter settings of the
         credentials.
        * @param password - password in plain text.
48      * @return PasswodInfo - for storing the credentials in DB.
        */
50     public PasswordInfo prepareForStore(PasswordCredential credentialSettings,
         String password) {
        // byte[] salt = genSalt();
52      // ScryptParams scryptParams = credentialSettings.getScryptParams();
        // byte[] hash = scrypt(password, salt, scryptParams);
54      // return new PasswordInfo(PasswordHashMethod.SCRYPT,
        // hash,
56      // salt,
        // scryptParams.toMap(),
58      // System.currentTimeMillis());

60      CryptParams cryptParams = credentialSettings.getCryptParams();
        String[] cryptParts = crypt(password, cryptParams);
62      byte[] salt = cryptParts[CRYPT_SALT_POS].getBytes(StandardCharsets.UTF_8);
        byte[] hash = cryptParts[CRYPT_HASH_POS].getBytes(StandardCharsets.UTF_8);
64
        return new PasswordInfo(PasswordHashMethod.CRYPT, hash, salt, cryptParams.
       toMap(), System.currentTimeMillis());
66     }

68     /**
        * Select the method for verifying the password hashes.
70      *
        * @param stored - stored credentials.
72      * @param password - password in plain text.
        * @return boolean - verification worked or not.
74      */
       public boolean verify(PasswordInfo stored, String password) {
76      PasswordHashMethod method = stored.getMethod();
        switch (method) {
78      case SCRYPT:
          return verifySCrypt(stored, password);
80      case SHA256:
          return verifySHA2(stored, password);
82      case CRYPT:
          return verifyCrypt(stored, password);
84      }
        throw new IllegalStateException("Shouldn't happen: " + "unsupported password
       hash method: " + method);
```

```
86    }
88    private boolean verifySHA2(PasswordInfo stored, String password) {
        Map<String, Object> methodParams = stored.getMethodParams();
90      int rehashNumber = (Integer) methodParams.getOrDefault("rehashNumber", 1);
        String salt = stored.getSalt() == null ? "" : new String(stored.getSalt(),
        StandardCharsets.UTF_8);
92      byte[] interim = (salt + password).getBytes(StandardCharsets.UTF_8);
        SHA256Digest digest = new SHA256Digest();
94      int size = digest.getDigestSize();
96      for (int i = 0; i < rehashNumber; i++)
          interim = sha2hash(interim, size, digest);
98
        return Arrays.areEqual(interim, stored.getHash());
100   }
102   private boolean verifySCrypt(PasswordInfo stored, String password) {
        ScryptParams params = new ScryptParams(stored.getMethodParams());
104     byte[] testedHash = scrypt(password, stored.getSalt(), params);
        return Arrays.areEqual(testedHash, stored.getHash());
106   }
108   public boolean checkParamsUpToDate(PasswordCredential credentialSettings,
        PasswordInfo stored) {
        if (stored.getMethod() != PasswordHashMethod.CRYPT)
110       return credentialSettings.isAllowLegacy();
112     CryptParams params = new CryptParams(stored.getMethodParams());
        return credentialSettings.getCryptParams().equals(params);
114   }
116   private byte[] scrypt(String password, byte[] salt, ScryptParams params) {
        return SCrypt.generate(password.getBytes(StandardCharsets.UTF_8), salt, 1 <<
        params.getWorkFactor(),
118         params.getBlockSize(), params.getParallelization(), params.getLength());
        // Memory use = 128 bytes x cost x blockSize
120   }
122   @SuppressWarnings("unused")
      private byte[] genSalt() {
124     byte[] salt = new byte[SALT_LENGTH];
        random.nextBytes(salt);
126     return salt;
      }
128
      private byte[] sha2hash(byte[] current, int size, SHA256Digest digest) {
130     digest.update(current, 0, current.length);
        byte[] hashed = new byte[size];
132     digest.doFinal(hashed, 0);
        return hashed;
134   }
136   /**
       * Method for creating the password hashes in CRYPT format &lt;crypt(3)&gt;.
138    *
       * @param password - password in plain text.
140    * @param params - parameters for the hash algorithm.
       * @return String[] - All components of the crypt string as array.
142    *         $&lt;id&gt;$&lt;rounds&gt;$&lt;salt&gt;$&lt;hash&gt;
       */
```

```
144   private String[] crypt(String password, CryptParams params) {
        String saltCrypt = genSaltCrypt(params.getSaltLength());
146     String crypt = Crypt.crypt(password,
            "$" + params.getIdHashingAlgorithm() + "$rounds=" + params.getRounds() + "
        $" + saltCrypt);
148     return crypt.split("\\$");
      }
150
      /**
152    * Method for verifying the password hashes in CRYPT format &lt;crypt(3)&gt;.
       *
154    * @param stored - stored credentials.
       * @param password - password in plain text.
156    * @return boolean - crypt verification worked or not.
       */
158   private boolean verifyCrypt(PasswordInfo stored, String password) {
        CryptParams params = new CryptParams(stored.getMethodParams());
160     String crypt = Crypt.crypt(password, "$" + params.getIdHashingAlgorithm() + "
        $rounds=" + params.getRounds()
            + "$" + new String(stored.getSalt(), StandardCharsets.UTF_8));
162     String[] cryptParts = crypt.split("\\$");
        byte[] testedHash = cryptParts[CRYPT_HASH_POS].getBytes(StandardCharsets.UTF_8
        );
164
        return Arrays.areEqual(testedHash, stored.getHash());
166   }

168   /**
       * Generates a string of random chars from the B64T set.
170    *
       * @param num - Number of chars to generate.
172    * @return String - random salt.
       */
174   private String genSaltCrypt(final int num) {
        final StringBuilder saltString = new StringBuilder();
176     for (int i = 1; i <= num; i++) {
          saltString.append(B64T.charAt(random.nextInt(B64T.length())));
178     }
        return saltString.toString();
180   }
    }
```

Listing A.5: PasswordEngine.java

Im Listing A.5 ist die vollständige *Java*-Klasse „*PasswordEngine*“ abgedruckt. Sie dient dazu ein valides Passwort in *Unity IdM* zu erzeugen und zu verifizieren. Die JSON-Datei einer GET-Anfrage, welche dem Listing A.6 zu entnehmen ist, beinhaltet alle Attribute, die über die REST API von *Unity IdM* zu erhalten sind.

```
1  [
     {
3      "values": [
         "2018-11-06T14:04:23"
5      ],
       "creationTs": 1541509463208,
7      "updateTs": 1541509463208,
       "direct": true,
9      "name": "sys:LastAuthentication",
       "groupPath": "/",
11     "valueSyntax": "string"
```

```
  },
  {
    "values": [
      {
        "passwords": [
          {
            "method": "CRYPT",
            "methodParams": {
              "idHashingAlgorithm": 5,
              "hashLength": 43,
              "saltLength": 16,
              "rounds": 5000
            },
            "hash": "eTQzVXlpTUxrSEd2SnhMM3N6c3BNM3J2UWdhSy9yNTEuWEttSGhTL3poQw=="
,
            "salt": "QTlqU1J2TVROcjZlUjZxUA==",
            "time": 1541509476850
          }
        ],
        "outdated": false,
        "outdatedReason": null,
        "securityQuestion": null,
        "answer": null
      }
    ],
    "creationTs": 1541509476880,
    "updateTs": 1541509476880,
    "direct": true,
    "name": "sys:Credential:sys:password",
    "groupPath": "/",
    "valueSyntax": "string"
  },
  {
    "remoteIdp": "https://shib-idp2.awi.de/idp/shibboleth",
    "translationProfile": "sys:samlShib",
    "values": [
      "Fabian"
    ],
    "creationTs": 1541509463058,
    "updateTs": 1541509463074,
    "direct": true,
    "name": "firstname",
    "groupPath": "/",
    "valueSyntax": "string"
  },
  {
    "remoteIdp": "https://shib-idp2.awi.de/idp/shibboleth",
    "translationProfile": "sys:samlShib",
    "values": [
      "Mangels"
    ],
    "creationTs": 1541509463059,
    "updateTs": 1541509463070,
    "direct": true,
    "name": "surname",
    "groupPath": "/",
    "valueSyntax": "string"
  },
  {
    "remoteIdp": "https://shib-idp2.awi.de/idp/shibboleth",
    "translationProfile": "sys:samlShib",
```

```
    "values": [
      "Fabian Mangels"
    ],
    "creationTs": 1541509463078,
    "updateTs": 1541509463078,
    "direct": true,
    "name": "displayName",
    "groupPath": "/",
    "valueSyntax": "string"
  },
  {
    "remoteIdp": "https://shib-idp2.awi.de/idp/shibboleth",
    "translationProfile": "sys:samlShib",
    "values": [
      "Fabian Mangels"
    ],
    "creationTs": 1541509463057,
    "updateTs": 1541509463075,
    "direct": true,
    "name": "name",
    "groupPath": "/",
    "valueSyntax": "string"
  },
  {
    "values": [
      "attribute_request"
    ],
    "creationTs": 1541509476854,
    "updateTs": 1541509476854,
    "direct": true,
    "name": "sys:FilledEnquires",
    "groupPath": "/",
    "valueSyntax": "string"
  },
  {
    "values": [
      "Regular User"
    ],
    "creationTs": 1541509697602,
    "updateTs": 1541509697602,
    "direct": false,
    "name": "sys:AuthorizationRole",
    "groupPath": "/",
    "valueSyntax": "enumeration"
  },
  {
    "values": [
      "awiFed12"
    ],
    "creationTs": 1541509477162,
    "updateTs": 1541509477162,
    "direct": true,
    "name": "userName",
    "groupPath": "/",
    "valueSyntax": "string"
  },
  {
    "values": [
      "sys:all"
    ],
    "creationTs": 1541509463045,
```

```
    "updateTs": 1541509463045,
    "direct": true,
    "name": "sys:CredentialRequirements",
    "groupPath": "/",
    "valueSyntax": "string"
  },
  {
    "remoteIdp": "https://shib-idp2.awi.de/idp/shibboleth",
    "translationProfile": "sys:samlShib",
    "values": [
      {
        "value": "fabian.mangels@awi.de",
        "confirmationData": {
          "confirmed": false,
          "confirmationDate": 0,
          "sentRequestAmount": 0
        },
        "tags": []
      }
    ],
    "creationTs": 1541509463067,
    "updateTs": 1541509463067,
    "direct": true,
    "name": "email",
    "groupPath": "/",
    "valueSyntax": "verifiableEmail"
  }
]
```

Listing A.6: unity_attributes.json

Damit auf sogenannte „*Low Level Events*“ reagiert werden kann, kommt das *Groovy*-Skript „*myGroovy_basicEvents.groovy*“ aus dem Listing A.7 zum Einsatz [UT18b, *Developing Groovy script*].

```
import groovy.json.JsonSlurper;
import java.net.URLEncoder;
import java.time.Instant;

import pl.edu.icm.unity.engine.api.authn.InvocationContext;

try {
        final String FILE_PATH = '/home/sync-user-one/monitoring/';
        String FILE_NAME = "";

        log.info("##### Event --> " + event + " --- OK #####");
        //log.info("EVENT: " + event);
        //log.info("CONTEXT: " + context);

        def slurper = new groovy.json.JsonSlurper();
        def result = slurper.parseText(context);

        String typeId = null;
        long entityId = -999;
        String attributeName = null;
        String identifier = null;
        String identifierURL = null;

        switch(event) {
                /*case "methodInvocation.addEntity":
```

```
26                  typeId = result.args[0].typeId;
                    identifier = result.args[0].value;
28                  identifierURL = URLEncoder.encode(identifier, "UTF-8");
                    break;*/
30              case "methodInvocation.submitEnquiryResponse":
                        if((result.args[0].FormId).equals("attribute_request") &&
    InvocationContext.hasCurrent()) {
32                          entityId = InvocationContext.getCurrent().
    getLoginSession().getEntityId();
                            typeId = "entityId";
34                      }
                        break;
36              case "methodInvocation.removeEntity":
                        if(result.args[0].entityId != null) {
38                          typeId = "entityId";
                            entityId = result.args[0].entityId;
40                      }
                        break;
42              case "methodInvocation.setEntityCredential":
                        if(result.exception == null && result.args[0].entityId !=
    null) {
44                          typeId = "entityId";
                            entityId = result.args[0].entityId;
46
                            // Plaintext Password ...
48                          //String password = slurper.parseText(result.args
    [2]).password;
                            //log.info("New plaintext password is: " +
    password);
50                      }
                        break;
52              case "methodInvocation.setAttribute":
                        log.info(result.args[1].name);
54                      if(result.args[0].entityId != null && result.args[1].name
    != null) {
                            typeId = "entityId";
56                          entityId = result.args[0].entityId;

58                          // Attribute ...
                            attributeName = result.args[1].name;
60                          //log.info("New attribute is: " + attributeName);
                        }
62                      break;
                case "methodInvocation.createAttribute":
64                      if(result.args[0].entityId != null && result.args[1].name
    != null) {
                            typeId = "entityId";
66                          entityId = result.args[0].entityId;

68                          // Attribute ...
                            attributeName = result.args[1].name;
70                          //log.info("Create new attribute is: " +
    attributeName);
                        }
72                      break;
                case "methodInvocation.removeAttribute":
74                      if(result.args[0].entityId != null && result.args[1].name
    != null) {
                            typeId = "entityId";
76                          entityId = result.args[0].entityId;
```

```
78                                  // Attribute ...
                                    attributeName = result.args[1].name;
80                                  //log.info("Remove attribute is: " + attributeName
    );
                            }
82                          break;
                    default :
84                          //log.warn("##### Event ---> " + event + " --- NO ACTION
    #####");
            }
86
            // DEBUG INFO
88          /*log.info("typeId is: " + typeId);
              log.info("entityId is: " + entityId);
90            log.info("identifier is: " + identifier);
              log.info("identifierURL is: " + identifierURL);*/
92
            if(typeId != null && typeId != "") {
94                  FILE_NAME = typeId + "_";
                    FILE_NAME += typeId.equals("identifier") ? identifierURL :
    entityId;
96                  FILE_NAME += "_" + event + "_";
                    FILE_NAME += event.equals("methodInvocation.setAttribute") ||
    event.equals("methodInvocation.createAttribute") || event.equals("
    methodInvocation.removeAttribute") ? attributeName + "_" : "";
98                  FILE_NAME += Instant.now().getEpochSecond() + ".unity"

100                         new File(FILE_PATH, FILE_NAME).withWriter('utf-8') {
                                writer -> writer.writeLine ' '
102                         }
            }
104
} catch (Exception e) {
106         log.warn("##### Event --> " + event + " --- ERROR #####", e);
}
```

Listing A.7: myGroovy_basicEvents.groovy

A.6 Benutzer-Entitätensynchronisation

In den nachstehenden Listings sind die vollständigen *Bash*-Skripte zu den beiden Synchronisationsprozessen abgedruckt. Der erste Teil der Benutzer-Entitätensynchronisation wird durch die Anweisungen im Listing A.8 realisiert. Entsprechend wird dies für den zweiten Teil im Listing A.9 verwirklicht.

```
1 #!/bin/bash
  # Sync Process One
3 # Fabian Mangels
  # Alfred-Wegener-Institut
5 #-----------------------------------------#

7 . /home/sync-user-one/conf/sync-process-one.properties

9 cd "$BASE_DIR"

11 function log() {
    if [ -n "$LOG_DIR" ] && [ -n "$LOG_FILE" ]
```

```
13    then
        NOW=$(date +"%d.%m.%Y-%H:%M:%S")
15      echo "$NOW> $1" >>${LOG_DIR}/$LOG_FILE
      fi
17  }

19  function log_debug() {
      if [ -n "$LOG_DIR" ] && [ -n "$DEBUG_FILE" ] && [ -n "$FILE" ] && [ -e ${
        WORK_DIR}/$FILE.ldif ]
21    then
        NOW=$(date +"%d.%m.%Y-%H:%M:%S")
23      echo "$NOW> $FILE.ldif - debug - start" >> ${LOG_DIR}/$DEBUG_FILE
        cat ${WORK_DIR}/$FILE.ldif >> ${LOG_DIR}/$DEBUG_FILE
25      echo "$NOW> $FILE.ldif - debug - end" >> ${LOG_DIR}/$DEBUG_FILE
      fi
27  }

29  function getJsonValue() {
      if [ -n "$1" ] && [ -n "$WORK_DIR" ] && [ -n "$FILE" ]
31    then
        if [ ! -e ${WORK_DIR}/$FILE.tmp ] && [ -n "$entityId" ] && [ -n "$CONF_DIR" ]
33      then
          curl -s -k --netrc-file $CONF_DIR/.netrc "https://unitysrv1.awi.de/rest-
        admin/v1/entity/$entityId/attributes" | sed -e 's/\\//g' -e 's/"{/{/g' -e 's
        /}"/}/g' > ${WORK_DIR}/$FILE.tmp
35
          if [[ $? == 0 ]]
37        then
            log "$FILE.tmp - tmp file created"
39        else
            log "$FILE.tmp - tmp file not created"
41        fi
        fi
43
        case "$1" in
45        "surname") result=$(cat ${WORK_DIR}/$FILE.tmp | jq -c '.[] | select(.["name
        "] == "surname") | .["values"][0]' | sed 's/"//g')
            ;;
47        "firstname") result=$(cat ${WORK_DIR}/$FILE.tmp | jq -c '.[] | select(.["
        name"] == "firstname") | .["values"][0]' | sed 's/"//g')
            ;;
49        "name") result=$(cat ${WORK_DIR}/$FILE.tmp | jq -c '.[] | select(.["name"]
        == "name") | .["values"][0]' | sed 's/"//g')
            ;;
51        "displayName") result=$(cat ${WORK_DIR}/$FILE.tmp | jq -c '.[] | select(.["
        name"] == "displayName") | .["values"][0]' | sed 's/"//g')
            ;;
53        "email") result=$(cat ${WORK_DIR}/$FILE.tmp | jq -c '.[] | select(.["name"]
        == "email") | .["values"][0]["value"]' | sed 's/"//g')
            ;;
55        "hash") result=$(cat ${WORK_DIR}/$FILE.tmp | jq -c '.[] | select(.["name"]
        == "sys:Credential:sys:password") | .["values"][]["passwords"][] | select(.["
        method"] == "CRYPT") | .["hash"]' | sed 's/"//g' | base64 -d)
            ;;
57        "salt") result=$(cat ${WORK_DIR}/$FILE.tmp | jq -c '.[] | select(.["name"]
        == "sys:Credential:sys:password") | .["values"][]["passwords"][] | select(.["
        method"] == "CRYPT") | .["salt"]' | sed 's/"//g' | base64 -d)
            ;;
59        *) result="ERROR"
            ;;
61      esac
```

```
63      echo "$result"
64    fi
65  }
66
67  function putAttributeRequest() {
68    if [ -n "$1" ] && [ -n "$2" ] && [ -n "$CONF_DIR" ] && [ -n "$FILE" ]
69    then
70      curl -s -k --netrc-file $CONF_DIR/.netrc -X PUT -H "Content-Type: application/
        json" -d "$2" "https://unitysrv1.awi.de/rest-admin/v1/entity/$1/attribute"
71      if [[ $? == 0 ]]
72      then
73        log "$FILE - put attribute request successful"
74      else
75        log "$FILE - put attribute request failed"
76      fi
77    fi
78  }
79
80  function writeInLdif() {
81    if [ -n "$1" ] && [ "$2" != "ERROR" ] && [ -n "$WORK_DIR" ] && [ -n "$FILE" ]
82    then
83      if [ -n "$2" ]
84      then
85        echo "$1 $2" >> ${WORK_DIR}/$FILE.ldif
86      else
87        echo "$1 n.a." >> ${WORK_DIR}/$FILE.ldif
88      fi
89    fi
90  }
91
92  function transAttrUnity2Ldap() {
93    if [ -n "$1" ]
94    then
95      case "$1" in
96        "surname") result="sn"
97          ;;
98        "firstname") result="givenName"
99          ;;
100       "name") result="cn"
101         ;;
102       "displayName") result="displayName"
103         ;;
104       "email") result="mail"
105         ;;
106       *) result="ERROR"
107         ;;
108     esac
109
110     echo "$result"
111   fi
112 }
113
114 function encryptLdif() {
115   if [ -n "$WORK_DIR" ] && [ -n "$FILE" ] && [ -e ${WORK_DIR}/$FILE.ldif ] && [ -n
        "$CONF_DIR" ] && [ -e ${CONF_DIR}/public_key.pem ]
116   then
117     # create OTP
118     openssl rand -base64 -out ${WORK_DIR}/$FILE.otp 128
119
120     # encrypt LDIF with OTP
```

```
121     openssl enc -aes-256-cbc -salt -in ${WORK_DIR}/$FILE.ldif -out ${WORK_DIR}/
        $FILE.ldif.enc -pass file:${WORK_DIR}/$FILE.otp

123     # encrypt OTP with public_key.pem
        openssl rsautl -encrypt -inkey ${CONF_DIR}/public_key.pem -pubin -in ${
        WORK_DIR}/$FILE.otp -out ${WORK_DIR}/$FILE.otp.enc
125   fi
    }
127

129 log "`basename $0` - executed"

131 watchnames=''
    [ -d $MONITORING_DIR ] && watchnames="$watchnames $MONITORING_DIR"
133
    inotifywait -mq -e create -e attrib --format %f $watchnames | while read FILE
135 do
      if [[ $FILE == entityId_*_methodInvocation.*.unity ]]
137   then
        log "$FILE - file detected - do action"
139
        entityId=$(echo $FILE | cut -d"_" -f2)
141     uid="awiFed$entityId"
        log "entityId: $entityId, uid: $uid"
143
        rm -f ${WORK_DIR}/$FILE.tmp
145     rm -f ${WORK_DIR}/$FILE.ldif

147     log "$FILE - file moved to work dir"
        mv $MONITORING_DIR/$FILE $WORK_DIR/.
149
        if [[ $FILE == *_methodInvocation.submitEnquiryResponse_* ]]
151     then
          log "$FILE - create entity - start"
153
          log "$FILE - create entity - put userName to unity"
155       putAttributeRequest "$entityId" "{\"values\":[\"$uid\"],\"name\": \"userName
        \",\"groupPath\": \"/\"}"

157       writeInLdif "dn:" "uid=$uid,ou=People,dc=unity,dc=awi,dc=de"
          writeInLdif "changetype:" "add"
159       writeInLdif "objectClass:" "inetOrgPerson"
          writeInLdif "objectClass:" "posixAccount"
161       writeInLdif "objectClass:" "shadowAccount"
          writeInLdif "uid:" "$uid"
163       writeInLdif "loginShell:" "/bin/bash"
          writeInLdif "homeDirectory:" "/home/$uid"
165       writeInLdif "gidNumber:" "5000"
          writeInLdif "uidNumber:" "$((10000 + $entityId))"
167       writeInLdif "description:" "uid=$uid,ou=People,dc=unity,dc=awi,dc=de"

169       sn=$(getJsonValue "surname")
          writeInLdif "sn:" "$sn"
171
          givenName=$(getJsonValue "firstname")
173       writeInLdif "givenName:" "$givenName"

175       cn=$(getJsonValue "name")
          writeInLdif "cn:" "$cn"
177
          displayName=$(getJsonValue "displayName")
```

```
179         writeInLdif "displayName:" "$displayName"

181         mail=$(getJsonValue "email")
            writeInLdif "mail:" "$mail"
183
            passwdHash=$(getJsonValue "hash")
185         passwdSalt=$(getJsonValue "salt")
            writeInLdif "userPassword:" '{CRYPT}$5$rounds=5000$'$passwdSalt'$'
        $passwdHash
187
            log "$FILE - create entity - end"
189
        elif [[ $FILE == *_methodInvocation.removeEntity_* ]]
191     then
            log "$FILE - remove entity - start"
193
            writeInLdif "dn:" "uid=$uid,ou=People,dc=unity,dc=awi,dc=de"
195         writeInLdif "changetype:" "delete"

197         log "$FILE - remove entity - end"

199     elif [[ $FILE == *_methodInvocation.setEntityCredential_* ]]
        then
201         log "$FILE - new credentials - start"

203         writeInLdif "dn:" "uid=$uid,ou=People,dc=unity,dc=awi,dc=de"
            writeInLdif "changetype:" "modify"
205         writeInLdif "replace:" "userPassword"

207         passwdHash=$(getJsonValue "hash")
            passwdSalt=$(getJsonValue "salt")
209         writeInLdif "userPassword:" '{CRYPT}$5$rounds=5000$'$passwdSalt'$'
        $passwdHash

211         log "$FILE - new credentials - end"

213     elif [[ $FILE == *_methodInvocation.setAttribute_* ]]
        then
215         log "$FILE - set attribute - start"

217         attributeName=$(echo $FILE | cut -d"_" -f4)
            attributeValue=$(getJsonValue "$attributeName")
219         attributeName=$(transAttrUnity2Ldap "$attributeName")

221         writeInLdif "dn:" "uid=$uid,ou=People,dc=unity,dc=awi,dc=de"
            writeInLdif "changetype:" "modify"
223         writeInLdif "replace:" "$attributeName"
            writeInLdif "$attributeName:" "$attributeValue"
225
            log "$FILE - set attribute - end"
227
        elif [[ $FILE == *_methodInvocation.createAttribute_* ]]
229     then
            log "$FILE - create attribute - start"
231
            attributeName=$(echo $FILE | cut -d"_" -f4)
233         attributeValue=$(getJsonValue "$attributeName")
            attributeName=$(transAttrUnity2Ldap "$attributeName")
235
            writeInLdif "dn:" "uid=$uid,ou=People,dc=unity,dc=awi,dc=de"
237         writeInLdif "changetype:" "modify"
```

```
      writeInLdif "add:" "$attributeName"
239   writeInLdif "$attributeName:" "$attributeValue"

241   log "$FILE - create attribute - end"

243 elif [[ $FILE == *_methodInvocation.removeAttribute_* ]]
    then
245   log "$FILE - remove attribute - start"

247   attributeName=$(echo $FILE | cut -d"_" -f4)
      attributeName=$(transAttrUnity2Ldap "$attributeName")
249
      writeInLdif "dn:" "uid=$uid,ou=People,dc=unity,dc=awi,dc=de"
251   writeInLdif "changetype:" "modify"
      writeInLdif "delete:" "$attributeName"
253
      log "$FILE - remove attribute - end"
255
    fi
257
    if [ -e ${WORK_DIR}/$FILE.ldif ]
259 then
      log "$FILE.ldif - ldif file created"
261
      log_debug
263
      encryptLdif
265
      if [ -e ${WORK_DIR}/$FILE.ldif.enc ] && [ -e ${WORK_DIR}/$FILE.otp.enc ]
267   then
        log "$FILE.ldif.enc - ldif.enc file created"
269     log "$FILE.otp.enc - otp.enc file created"

271     scp ${WORK_DIR}/$FILE.otp.enc openldap:~/work/.
        scp ${WORK_DIR}/$FILE.ldif.enc openldap:~/monitoring/.
273
        if [[ $? == 0 ]]
275     then
          log "$FILE.ldif.enc - ldif.enc file transferred"
277     else
          log "$FILE.ldif.enc - ldif.enc file not transferred"
279     fi

281   else
        log "$FILE.ldif - encryption failed"
283   fi

285 else
      log "$FILE.ldif - ldif file not created"
287 fi

289 log "$FILE - files (org, tmp, ldif, otp, enc) deleted"
    rm -f ${WORK_DIR}/$FILE
291 rm -f ${WORK_DIR}/$FILE.tmp
    rm -f ${WORK_DIR}/$FILE.ldif
293 rm -f ${WORK_DIR}/$FILE.ldif.enc
    rm -f ${WORK_DIR}/$FILE.otp
295 rm -f ${WORK_DIR}/$FILE.otp.enc

297 else
    log "$FILE - file detected - do no action"
```

```
299
        log "$FILE - file deleted"
301     rm -f ${MONITORING_DIR}/$FILE
    fi
303 done
```

Listing A.8: sync-process-one.sh

```
1  #!/bin/bash
   # Sync Process Two
3  # Fabian Mangels
   # Alfred-Wegener-Institut
5  #-------------------------------------------#

7  . /home/sync-user-two/conf/sync-process-two.properties

9  cd "$BASE_DIR"

11 function log() {
     if [ -n "$LOG_DIR" ] && [ -n "$LOG_FILE" ]
13   then
       NOW=$(date +"%d.%m.%Y-%H:%M:%S")
15     echo "$NOW> $1" >>${LOG_DIR}/$LOG_FILE
     fi
17 }

19 function log_debug() {
     if [ -n "$LOG_DIR" ] && [ -n "$DEBUG_FILE" ] && [ -n "$FILE" ] && [ -e ${
       WORK_DIR}/$FILE.ldif ]
21   then
       NOW=$(date +"%d.%m.%Y-%H:%M:%S")
23     echo "$NOW> $FILE.ldif - debug - start" >> ${LOG_DIR}/$DEBUG_FILE
       cat ${WORK_DIR}/$FILE.ldif >> ${LOG_DIR}/$DEBUG_FILE
25     echo "$NOW> $FILE.ldif - debug - end" >> ${LOG_DIR}/$DEBUG_FILE
     fi
27 }

29 function decryptLdif() {
     if [ -n "$WORK_DIR" ] && [ -n "$FILE" ] && [ -e ${WORK_DIR}/$FILE.ldif.enc ] &&
       [ -e ${WORK_DIR}/$FILE.otp.enc ] && [ -n "$CONF_DIR" ] && [ -e ${CONF_DIR}/
       private_key.pem ]
31   then
       # decrypt encrypted OTP with private_key.pem
33     openssl rsautl -decrypt -inkey ${CONF_DIR}/private_key.pem -in ${WORK_DIR}/
       $FILE.otp.enc -out ${WORK_DIR}/$FILE.otp

35     # decrypt encrypted LDIF with OTP
       openssl enc -d -aes-256-cbc -in ${WORK_DIR}/$FILE.ldif.enc -out ${WORK_DIR}/
       $FILE.ldif -pass file:${WORK_DIR}/$FILE.otp
37   fi
   }
39

41 log "`basename $0` - executed"

43 watchnames=''
   [ -d $MONITORING_DIR ] && watchnames="$watchnames $MONITORING_DIR"
45

   inotifywait -mq -e create -e attrib --format %f $watchnames | while read FILE
47 do
     if [[ $FILE == entityId_*_methodInvocation.*.unity.ldif.enc ]]
```

```
49   then
       log "$FILE - file detected - do action"
51
       log "$FILE - file moved to work dir"
53     mv $MONITORING_DIR/$FILE $WORK_DIR/.

55     FILE=$(echo $FILE | sed 's/.ldif.enc//g')

57     decryptLdif

59     if [ -e ${WORK_DIR}/$FILE.ldif ] && [ -e ${WORK_DIR}/$FILE.otp ]
       then
61       log "$FILE.otp - otp file created"
         log "$FILE.ldif - ldif file created"
63
         log_debug
65
         ldapmodify -x -D cn=admin,dc=unity,dc=awi,dc=de -y $CONF_DIR/.passwd -f
     $WORK_DIR/$FILE.ldif
67
         if [[ $? == 0 ]]
69       then
           log "$FILE - ldapmodify successful"
71       else
           log "$FILE - ldapmodify failed"
73       fi

75     else
         log "$FILE.ldif.enc - decryption failed"
77     fi

79     log "$FILE - files (ldif, otp, enc) deleted"
       rm -f $WORK_DIR/$FILE.ldif
81     rm -f $WORK_DIR/$FILE.ldif.enc
       rm -f $WORK_DIR/$FILE.otp
83     rm -f $WORK_DIR/$FILE.otp.enc

85   else
       log "$FILE - file detected - do no action"
87
       log "$FILE - file deleted"
89     rm -f $MONITORING_DIR/$FILE
     fi
91 done
```

Listing A.9: sync-process-two.sh

A.7 LDIF-Beispiele

In diesem Abschnitt werden einige konkrete LDIF-Dateien vorgestellt, deren Erstellung durch unterschiedliche Ereignisse („*Low Level Events*“) am *SP-IdP-Proxy* (*Unity IdM*) ausgelöst wurden.

Im ersten Listing A.10 wird die Erstellung einer neuen Entität dargestellt. Die LDIF-Datei wird auf das Ereignis „*submitEnquiryResponse*“, bedingt durch das Abschicken des Formulars „*attribute_request*“ am Proxy, erstellt.

```
dn: uid=awiFed2,ou=People,dc=unity,dc=awi,dc=de
changetype: add
objectClass: inetOrgPerson
objectClass: posixAccount
objectClass: shadowAccount
uid: awiFed2
loginShell: /bin/bash
homeDirectory: /home/awiFed2
gidNumber: 5000
uidNumber: 10002
description: uid=awiFed2,ou=People,dc=unity,dc=awi,dc=de
sn: Mangels
givenName: Fabian
cn: Fabian Mangels
displayName: Fabian Mangels
mail: fabian.mangels@awi.de
userPassword: {CRYPT}$5$rounds=5000$SVb/TG.2
    e9PnJLDJ$hsW4EChqtxFTb9clt6wAxN5P1azYKfo1ig.JgkoU4P.
```

Listing A.10: entityId_2_methodInvocation.submitEnquiryResponse_1542024212.unity.ldif

Im Listing A.11 werden die *Credentials* bzw. das Passwort einer zugehörigen Entität neu gesetzt. Für die Erstellung wird auf das Ereignis „*setEntityCredential*“ gewartet. Das Passwortformat wird dabei maßgebend durch die *crypt(3)*-Funktion in der *glibc-Library* bestimmt.

```
dn: uid=awiFed2,ou=People,dc=unity,dc=awi,dc=de
changetype: modify
replace: userPassword
userPassword: {CRYPT}$5$rounds=5000$1xFq0/GHmt4QTkIm$zo7XVwNH30VhPPq7aEA.
    miJ7LxWpwk3wYZa8oBH./N3
```

Listing A.11: entityId_2_methodInvocation.setEntityCredential_1542024274.unity.ldif

Natürlich besteht auch die Möglichkeit, dass eine Entität entfernt wird; das wird durch die Anweisung im Listing A.12 bewirkt.

```
dn: uid=awiFed2,ou=People,dc=unity,dc=awi,dc=de
changetype: delete
```

Listing A.12: entityId_2_methodInvocation.removeEntity_1542024374.unity.ldif

Alle weiteren LDIF-Dateien befassen sich mit der Modifikation eines Attributs einer Entität. Zu Demonstrationszwecken wurde hier das LDAP-Attribut „*mail*“ verwendet. Im Listing A.13 wird die Erstellung, im Listing A.14 die Veränderung und im Listing A.15 das endgültige Entfernen des jeweiligen Attributes ersichtlich.

```
dn: uid=awiFed2,ou=People,dc=unity,dc=awi,dc=de
changetype: modify
add: mail
mail: fabian.mangels@awi.de
```

Listing A.13: entityId_2_methodInvocation.createAttribute_email_1542024307.unity.ldif

```
dn: uid=awiFed2,ou=People,dc=unity,dc=awi,dc=de
changetype: modify
replace: mail
mail: fabian.mangels@awi.de
```

Listing A.14: entityId_2_methodInvocation.setAttribute_email_1542024325.unity.ldif

```
dn: uid=awiFed2,ou=People,dc=unity,dc=awi,dc=de
changetype: modify
delete: mail
```

Listing A.15: entityId_2_methodInvocation.removeAttribute_email_1542024343.unity.ldif

A.8 VMware vRealize Automation

Die Abbildung A.10 zeigt die Anmeldemaske der einzubettenden Software *VMware vRealize Automation*. Wichtig ist, dass für eine erfolgreiche Anmeldung mit einem zuvor erstellten lokalen Benutzerkonto am *SP-IdP-Proxy* die Auswahl der Authentifizierungsquelle „*unity.awi.de*“ erfolgen muss.

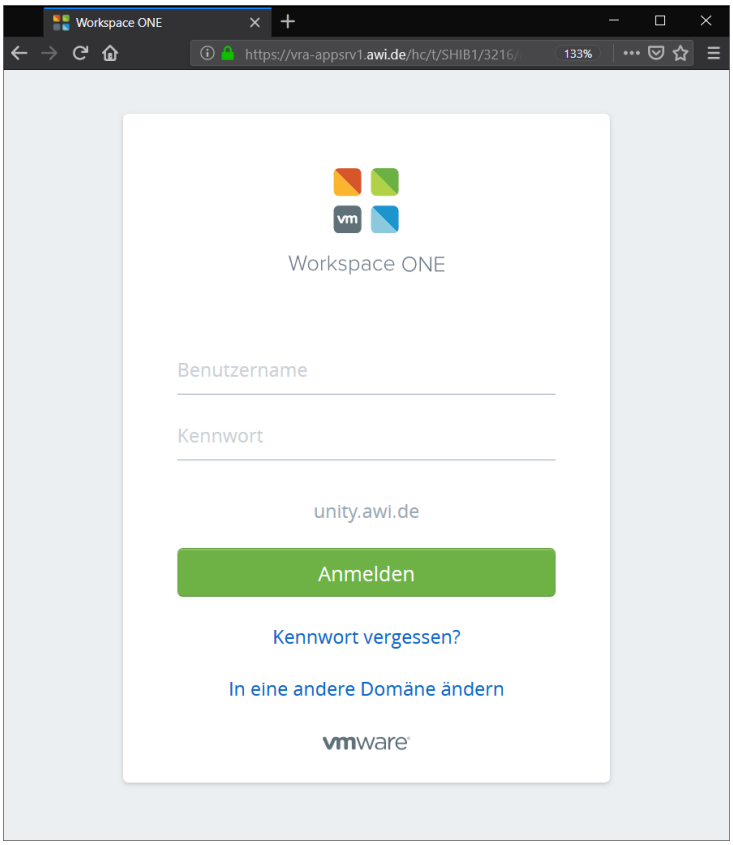

Abbildung A.10: Anmeldemaske der *Marketplace*-Software

In der Abbildung A.11 und Abbildung A.12 ist die Konfigurationsoberfläche in vRA zu sehen, in der die Integration des OpenLDAP-Verzeichnisses erfolgte [VT18a]. Für einen erfolgreichen Import der Benutzerkonten musste ein weiteres Attribut (Zweckentfremdung von „*description*“) jeder Entität im LDAP-Verzeichnisdienst hinzugefügt werden. vRA benötigt nämlich einen zusätzlichen Eintrag, der den jeweiligen DN enthält; das Attribut „*dn*“ reicht hier nicht aus.

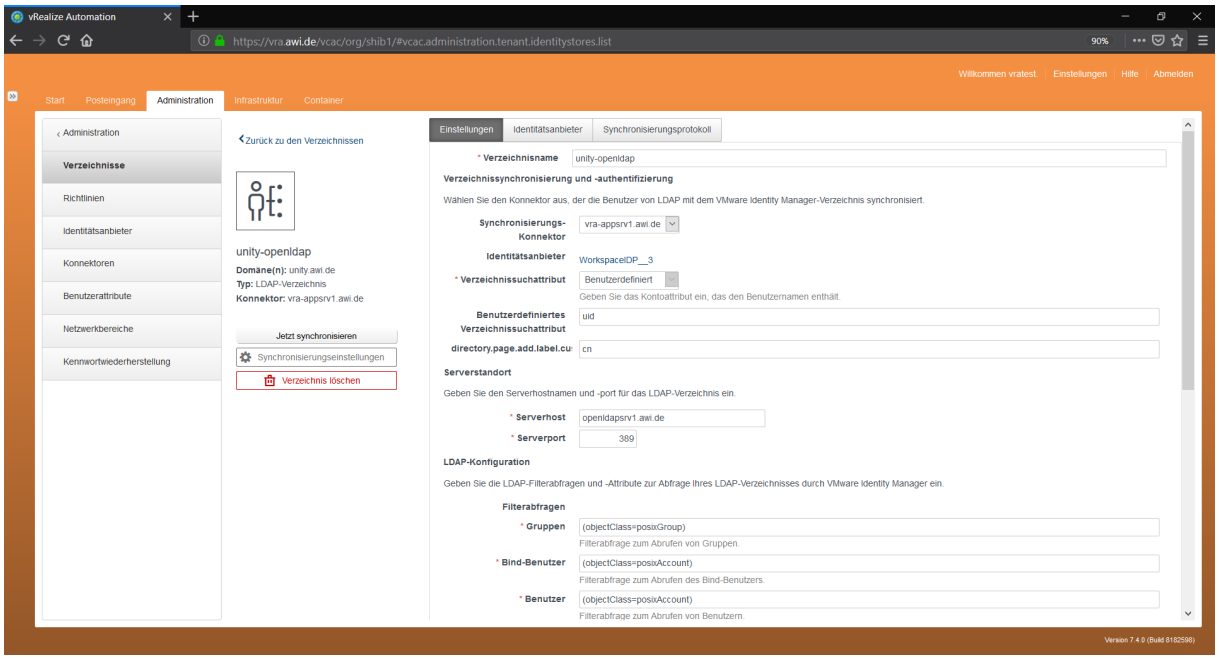

Abbildung A.11: Integration des LDAP-Verzeichnisses in vRA 1

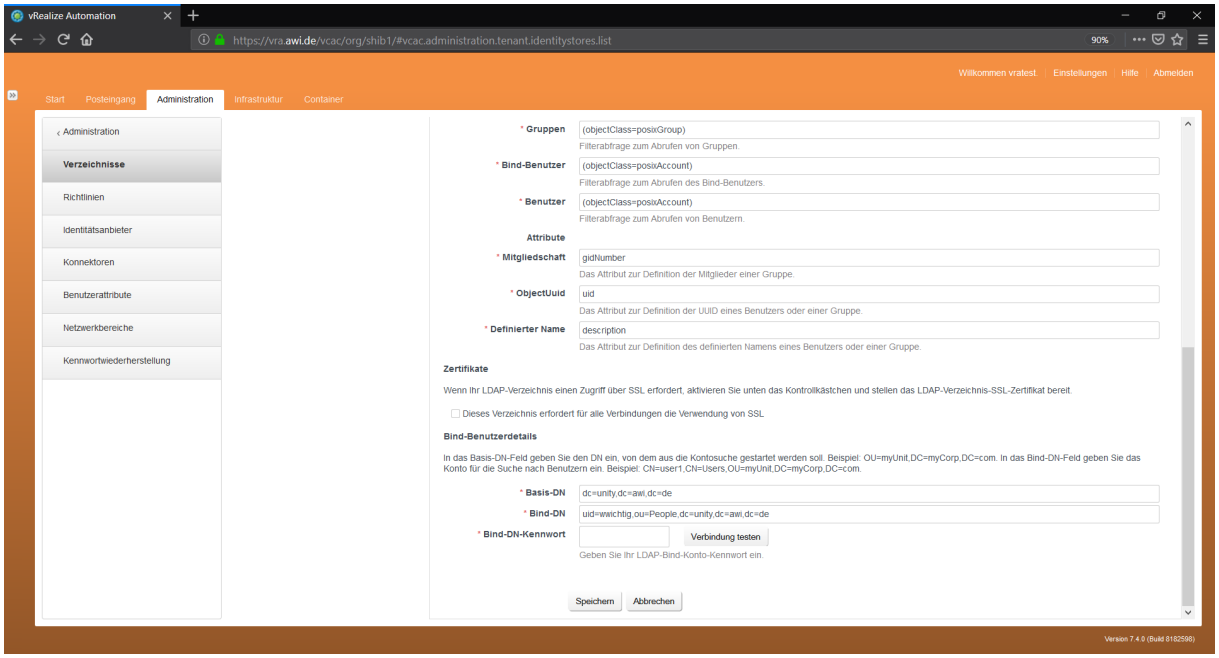

Abbildung A.12: Integration des LDAP-Verzeichnisses in vRA 2

Eidesstattliche Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Die Stellen der Arbeit, die anderen Werken dem Wortlaut oder dem Sinn nach entnommen wurden, sind durch Angaben der Herkunft kenntlich gemacht.

Diese Erklärung erstreckt sich auch auf in der Arbeit enthaltene Grafiken, Skizzen, bildliche Darstellungen sowie auf Quellen aus dem Internet. Die Arbeit habe ich in gleicher oder ähnlicher Form auch auszugsweise noch nicht als Bestandteil einer Prüfungs- oder Studienleistung vorgelegt.

Ich versichere, dass die eingereichte elektronische Version der Arbeit vollständig mit der Druckversion übereinstimmt.

Bremerhaven, den 03.12.2018

Fabian Mangels <fabian@mangels.it>
Matrikel-Nr.: 34295