Universität Karlsruhe (TH)
Forschungsuniversität • gegründet 1825

Fakultät für **Informatik**

Schnittstellenimplementierung zur Provisionierung von Identitätsdaten

Studienarbeit

am Institut für Telematik
Forschungsbereich Dezentrale Systeme und Netzdienste (DSN)
Prof. Dr. Hannes Hartenstein
in Kooperation
mit der Abteilung technische Infrastruktur (ATIS)
Fakultät für Informatik
Universität Karlsruhe (TH)

von

Youssef Chaouqi

Betreuer:

Dipl.-Math. Klaus Scheibenberger
Dipl.-Geophys. Olaf Hopp
Dipl.-Inform. Ingo Pansa
Dipl.-Inform. Frank Schell
Dipl.-Inform. Thorsten Hoellrigl

Bearbeitungszeit: 01 Mai 2009 – 31. Juli 2009

Universität Karlsruhe (TH) · Institut für Telematik · Geb. 20.20 · Zirkel 2 · 76131 Karlsruhe

Ehrenwörtliche Erklärung

Ich erkläre hiermit, die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet zu haben.

Karlsruhe, den 18.07.2009

Youssef Chaouqi

1 EINLEITUNG

Das erste Kapitel beschreibt zuerst in welchem Themenumfeld sich die Studienarbeit befindet – dem Identitätsmanagement – und in welchem Projektumfeld – dem Identitätsmanagement der Universität Karlsruhe, repräsentiert durch das Projekt „Karlsruher Integriertes Informationsmanagement“ (KIM) –. Danach werden die Aufgaben gezeigt, die die „Abteilung für Infrastruktur der Fakultät für Informatik“ (ATIS) bei ihrem Identitätsmanagement bewältigen muss. Anschließen werden die Möglichkeiten beschrieben, die das KIM der ATIS bieten kann, um ihr Identitätsmanagement zu optimieren. Schließlich werden die Aufgabenstellung und der Lösungsvorgang beschrieben: Den konkreten Lösungsansatz sowie das Vorgehen zur Entwicklung der Lösung.

1.1 Einführung in das Themengebiet

1.1.1 Motivation

Heutzutage hat fast jeder Mensch (vorwiegend in den Industrieländern) mit computerunterstützten Kommunikationsprozessen, digitalem Datenaustausch und digitaler Datenspeicherung zu tun, auch wenn sich einige Menschen überhaupt nicht bewusst sind, dass sie Informationstechnologien (IT) benutzen [LDB+06]. Dabei wächst die ausgetauschte und gespeicherte Datenmenge stetig. Gleichzeitig wächst die Gefahr, dass die Daten von Unbefugten abgefangen und missbraucht werden. Besonders sensibel sind private Personendaten. In der realen Welt –fernab von Computern– müssen Personen, Behörden und Unternehmen, den richtigen und sicheren Umgang mit diesen Daten gewährleisten. Ähnlich wie in der realen Welt stellt der Schutz, die Kontrolle und die Verwaltung der persönlichen Daten der Menschen in der digitalen Welt eine wichtige Aufgabe und gleichzeitig eine große Herausforderung dar.

Unternehmen, Behörden, Institutionen und andere Einrichtungen speichern und verwalten die personenbezogenen Daten von ihren Kunden, Mitarbeitern und Partnern. Diese Daten ermöglichen den Einrichtungen die Kontrolle des Zugangs zu IT-Systemen und des Zugriffs auf schützenswerte Ressourcen. Somit basiert die IT-Sicherheit auf Zugangs- und Zugriffskontrolle [DH+08]. Diese Kontrolle wird mithilfe eines Identitätsmanagementsystems realisiert, das im nächsten Abschnitt (1.1.2) beschrieben wird. Wenn zum Beispiel ein Unternehmen einen neuen Mitarbeiter einstellt, dann werden seine persönlichen Daten (Name, Vorname, Geburtsdatum, Steuernummer, Adresse usw.) in verschiedenen Systemen und verschiedenen Abteilungen erfasst. Die Buchhaltungsabteilung benötigt die Daten, um dem Mitarbeiter den Lohn abzurechnen und zu überweisen. Die Abteilung, in der er arbeiten wird muss ihm den Zugang zu den entsprechenden Räumen und Ressourcen freischalten usw. So müssen die Daten des Mitarbeiters in vielen Systemen verwaltet und gepflegt werden, was zu einem großen Aufwand und zur Inkonsistenz führen kann, besonders wenn diese Daten sich ändern (ein neuer Name nach der Heirat, eine neue Kontoverbindung, eine neue Adresse oder eine neue Abteilung). Darüber hinaus können Sicherheitslücken entstehen, wenn der Mitarbeiter das Unternehmen verlässt oder die Abteilung wechselt und er weiterhin unbefugten Zugang auf sensitiven Daten wie bisher hat.

Diese Mängel können mittels eines zentralen dienstorientierten Identitätsmanagementsystems behoben werden. In solch einem System werden personenbezogene Informationen aus einer Menge von Datenquellen aggregiert, in einem zentralen Verzeichnis gehalten und über eine Datenschnittstelle für unternehmensweite (oder auch institutionsweite) Applikationen zur Verfügung gestellt. Darüber hinaus werden Änderungen sowohl im zentralen Verzeichnis als auch in den angeschlossenen Datenquellen anhand vordefinierter Regeln abgeglichen, wodurch die Aktualität und Konsistenz der Daten erreicht wird [HS+06]. In dieser Arbeit geht es darum, den Dienst einer solchen Dienstschnittstelle in Anspruch zu nehmen. Die Universität Karlsruhe betreibt ein zentrales Identitätsmanagementsystem und bietet ihren Einrichtungen die Möglichkeit, seine Dienste zu nutzen, um ihre lokalen Identitätsmanagementsysteme zu unterstützen. Die Fakultät für Informatik – als eine diese Einrichtungen – will einen dieser Dienste verwenden, um die Daten ihrer Mitarbeiter und Studenten abzugleichen. Dafür wird in dieser Studienarbeit eine Anwendung entworfen und prototypisch implementiert.

1.1.2 Identitätsmanagement

Aus den personenbezogenen Eigenschaften eines Menschen lassen sich Identitäten ableiten bzw. bilden. Von daher hat jeder Mensch eine Identität. Eine Identität ist eine Menge von Attributen (Eigenschaften) einer Person, die sie innerhalb einer Personengruppe als Individuum einzigartig identifiziert [LDB+06]. Außerdem können nicht nur juristische und reale Personen eine Identität besitzen, sondern auch sämtliche Objekte wie Rechner oder Softwaresysteme, die durch Attribute eindeutig beschrieben und identifiziert werden können. Beispiele für Attribute sind bei Personen Name, Geburtsdatum, Adresse, Augenfarbe, Fingerabdruck, Größe oder Beruf; in unserem Szenario an einer Hochschule z. B. Studentenausweisnummer, Matrikelnummer, und bei Rechnern Marke, MAC-Adresse oder IP-Adresse [DH+08].

In der digitalen Welt ist eine Identität, eine sogenannte digitale Identität, die Repräsentation von Personen oder Gütern in digitaler Form. Eine digitale Identität wird durch eine Menge von personenbezogenen Attributen beschrieben, die eine Person oder ein Objekt innerhalb eines konkreten Systems eindeutig identifizierbar machen und ihre spezifische digitale Repräsentation in diesem System darstellen [DH+08]. Eine Person kann dabei durchaus mehrere Identitäten in unterschiedlichen Systemen erhalten, aber auch in unterschiedlichen Rollen, während eine Identität gewöhnlich nur einer Person zuzuordnen ist. „*Eine Rolle ist als erwartetes Verhalten (eine Reihe von Handlungen) in einem vorgegebenen individuellen sozialen Kontext definiert*“ [LDB+06]. Rollen sind also Zuordnungen von Personen zu Personengruppen wie Studenten, Mitarbeiter oder andere.

Beim Wort Identitätsmanagement bedeutet „Management“, dass Identitätsinformationen verwaltet und verarbeitet werden. Ein Identitätsmanagementsystem im allgemeinen Sinne umfasst sämtliche (Identitätsmanagement-) Applikationen, Infrastrukturen und spezielle Verfahren, die in Bezug auf die kontrollierte Gestaltung von Identitäten in Kommunikationen eine Rolle spielen [HKRG+03]. Das Identitätsmanagement (IdM) umfasst also die Speicherung, Verwaltung und Nutzung von Identitätsinformationen. Dies beginnt beim Speichern in zentralen Verzeichnissen und geht über Authentifizierung und Autorisierung bis hin zu automatisierten Prozessen für das Management von Identitäten. Ziel des Identitätsmanagement ist, den verlässlichen Zugang zu vielfältigen Ressourcen anhand von Personenstammdaten (Identitätsdaten), die aus verschiedenen Quellsystemen übernommen und zusammengeführt werden, zu regeln.

Die Menge der für die Nutzung der Dienste notwendigen Attribute werden in den Nutzerkonten (Accounts) zusammengefasst, die dann beispielsweise über einen Verzeichnisdienst den dienstrelevanten Systemen bereitgestellt werden. Jeder Account ist einer bestimmten Identität zugeordnet. Hingegen können einer Identität mehrere Accounts zugeordnet werden.

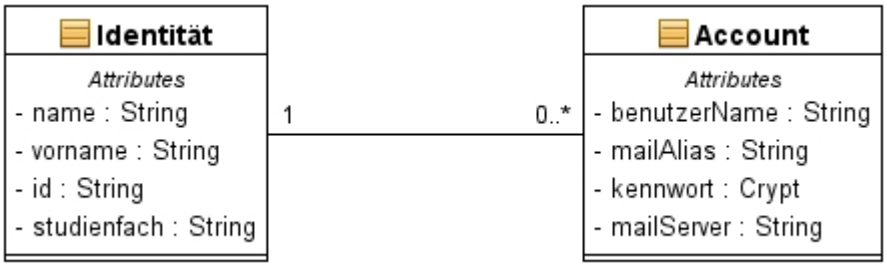


Abbildung 1 : "Identität und Account"-Klassendiagramm

Das Klassendiagramm (Abbildung 1) zeigt den Zusammenhang zwischen einer Identität und einem Account, der dieser Identität zugeordnet ist. Jeder Account muss genau einer Identität zugeordnet werden. Eine Identität hingegen kann mehrere Accounts „besitzen“. Dies bedeutet eine Person, die in einem Emailsystem durch eine einzige Identität repräsentiert wird, kann mehrere Email-Accounts besitzen. Gezeigt werden hier die Attribute des Accounts, die für die Authentifizierung u.a. am Mailsystem relevant sind (*benutzerName* und *passwort*) und die beispielsweise für die Auslieferung bzw. das Versenden von Mails (*mailAlias* und *mailServer*) relevant sind.

1.1.3 KIM-IdM

Im Rahmen des Projekts „Karlsruher Integriertes InformationsManagement“ (KIM) wurde das IdM-System der Universität Karlsruhe, das in dieser Arbeit als KIM-IdM bezeichnet wird, eingeführt. Die KIM-IdM-Architektur beruht auf dem Konzept, das die Universität als einen föderativen Verbund ihrer organisatorischen Einheiten bezeichnet. Diese Einheiten (Rechenzetrum, Bibliothek, Verwaltung, Fakultäten, Institute und andere) werden Satelliten genannt. Die Architektur (Abbildung 2) ermöglicht eine lose Kopplung von weiterhin autarken Organisationseinheiten, deren lokale IdM-Systeme auch durch die Integration in den föderativen Verbund weitgehend erhalten bleiben. Gleichzeitig können über Organisationseinheiten hinwegreichende Prozesse etabliert werden.

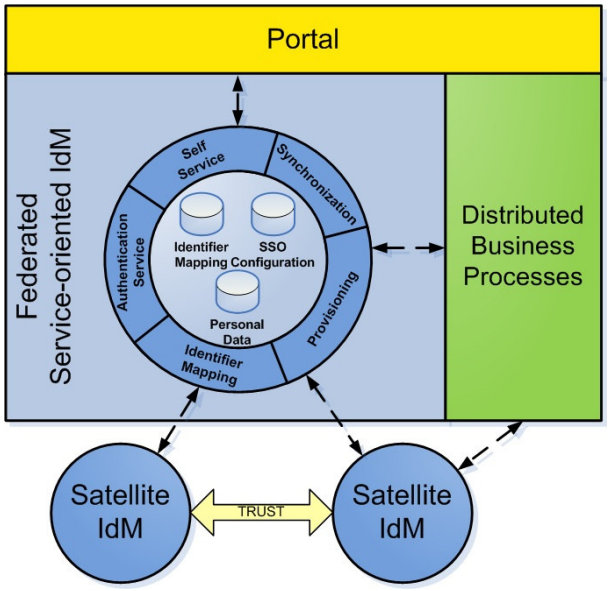


Abbildung 2 : Services und Satelliten in KIM-IdM

KIM-IdM bietet verschiedene Dienste für seine Satelliten, u.a. sogenannte „*Identity Shared Services for Infrastructure Services*“. Diese Identitätsinfrastrukturdienste können von Satelliten in Anspruch genommen werden. Einer von diesen Diensten ist die Provisionierung. Um in einem der Satelliten zu einer neuen Identität einen Account anzulegen, werden auf Anfrage die Attributwerte der vorab zwischen Satellit und KIM-IdM abgestimmten Identitätsattribute an den Satelliten übermittelt und anschließend werden Änderungen an den Identitätsattributen „synchronisiert“ [HS+07].

1.2 Szenario

Die Abteilung Technische Infrastruktur (ATIS) ist als Einrichtung der Fakultät für Informatik der Dienstleister zur technischen Unterstützung aller Forschungsgruppen. Die ATIS betreibt u.a. für die Fakultät einen studentischen Poolraum, der Studenten der Informatik für den Zweck der „Forschung und Lehre“ Arbeitsplätze mit Rechner anbietet, die mit entsprechender Software ausgestattet sind. Scanner, verschiedene Drucker und auch direkter Support werden angeboten. Zur Benutzung des Pools ist ein Pool-Account erforderlich. Zurzeit haben mehr als 2500 Studenten einen solchen Account. Mit dem Pool-Account kann sich ein Studierender an den Arbeitsplatzrechnern sowohl unter Windows als auch unter Linux anmelden (authentifizieren); dann wird ihm sein entsprechendes Heimatverzeichnis zugeordnet (Autorisierung). Damit kann er den Arbeitsplatzrechner und entsprechend auch die weiteren verfügbaren Dienste im Pool für seine Aufgaben nutzen.

Die Verwaltung all dieser Accounts und der zugehörigen Identitäten ist derzeit aufwendig und nicht optimal. Beim Anlegen neuer oder beim Verlängern der schon vorhandenen Accounts für die Studienanfänger jeder Studierende nachweisen, dass er ein immatrikulierter Informatik-Student ist. Bisher kann er das nur anhand seines Studentenausweises (Fricard) und seiner ausgedruckten Studienbescheinigung nachweisen.

Die beiden Dokumente –Fricard und Studienbescheinigung– haben sich als unpraktisch in der Handhabung erwiesen. Einerseits muss die Bescheinigung noch von einem Mitarbeiter online mit einem mehrstelligen alphanumerischen Code überprüft werden. Dieses Verfahren ist

somit sehr zeitaufwendig und problematisch. Das macht sich besonders am Anfang jedes Semesters bemerkbar. Während dieser Periode kommt täglich eine große Anzahl an Studenten in die Studentenauskunft, wo studentische Mitarbeiter den Studenten für den direkten Support zur Verfügung stehen, um Accounts anlegen oder verlängern zu lassen, was zu einem Massenproblem führt, das nur mit erhöhtem Zeit- und Personalaufwand zu bewältigen ist. Andererseits müssen die Daten der beiden Dokumente manuell erfasst werden, was das Massenproblem noch verschärft und die Fehleranfälligkeit erhöht.

Die Fricard und die Studienbescheinigung sind aber auch unzureichend im Bezug auf die Datenaktualität. Der Grund dafür ist, dass eine tägliche Aktualität der Daten mittels der Studienbescheinigung nicht gewährleistet werden kann, sondern nur eine sechsmonatige, da ein Student nur jedes Semester eine neue Studienbescheinigung und nur alle vier Jahre einen neuen Studentenausweis bekommt. Der sehr wichtige Punkt beim Management von Accounts –die Aktualität der Daten– lässt sich somit nur eingeschränkt erfüllen. Es sollte beispielsweise immer zeitnah festgestellt werden können, ob ein Student noch als Informatikstudent immatrikuliert ist, da nur wirklich immatrikulierte Informatik-Studierende das Recht auf einen aktiven Account haben. Wenn er exmatrikuliert wird, sollten seine Personendaten zeitnah aktualisiert werden, damit entsprechend reagiert werden kann, beispielsweise um seinen Account zu sperren, um Missbrauch zu verhindern. Eine erhöhte Datenaktualität erhöht somit die Sicherheit.

Automatisierte zeitnahe Abfrage und Synchronisation von Identitätsdaten aus einer verlässlichen Quelle würde die beiden oben erläuterten Schwachpunkte –schwierige Handhabung der Dokumente und mangelnde Aktualität der Daten– beseitigen.

1.3 Aufgabenstellung und Lösungsansatz

Das Ziel dieser Arbeit ist, eine Verbindung zwischen KIM-IdM und dem ATIS-IdM für den Studentenpool zu entwerfen und zu implementieren, um die oben genannten Schwachstellen durch die Nutzung des Provisionierungsdienstes von KIM-IDM zu beseitigen.

Die automatisierte Abfrage der Studentendaten aus KIM-IdM hätte Vorteile, die das Identitätsmanagement im Studentenpool der Fakultät optimieren und erweitern würden. Erstens sichert diese Anfrage täglich aktuelle und authentische Daten, weil sie aus dem Studienbüro stammen. Solche Daten ermöglichen eine hochqualitative Verwaltung. Zweitens übernimmt die Automatisierung sehr viel Routinearbeit. Das gilt auch für die Synchronisierung der alten Daten. Somit kann die ATIS den Poolraumbenutzern bessere Bedienung und Betreuung anbieten. Der weitere, ebenfalls in 1.2 angesprochene Schwachpunkt ist das fehleranfällige manuelle Erfassen der Personendaten. Könnten diese aus einer verlässlichen Quelle abgefragt und ohne manuellen Eingriff übernommen werden, würde dies den betrieblichen Ablauf, beispielsweise in der Zeit der Erstsemester-Anfangsphase, deutlich vereinfachen. Die in beiden Punkten angesprochene verlässliche Quelle ist im Falle des vorliegenden Szenarios KIM-IdM.

Beim Erstellen einer neuen Identität, der ein Account zugewiesen werden soll, sollen nur gewisse Identitätsdaten des Studenten (oder nur ein einziges Datum) als Anfrageparamenter an KIM-IDM übermittelt werden. Die restlichen erforderlichen Daten sollen dann vom KIM-IdM geliefert werden. Diese Identitätsdaten sollen anschließend mithilfe von KIM-IdM laufend aktualisiert werden, um feststellen zu können, ob der Benutzer immer noch berechtigt ist, einen aktiven Account zu besitzen.

1.4 Lösungsvorgehen

Das Lösungsvorgehensmodell in dieser Arbeit lehnt sich an bekannte Elemente aus der Softwaretechnik [Ba1-00]. Es besteht aus vier Phasen (Abbildung 3).

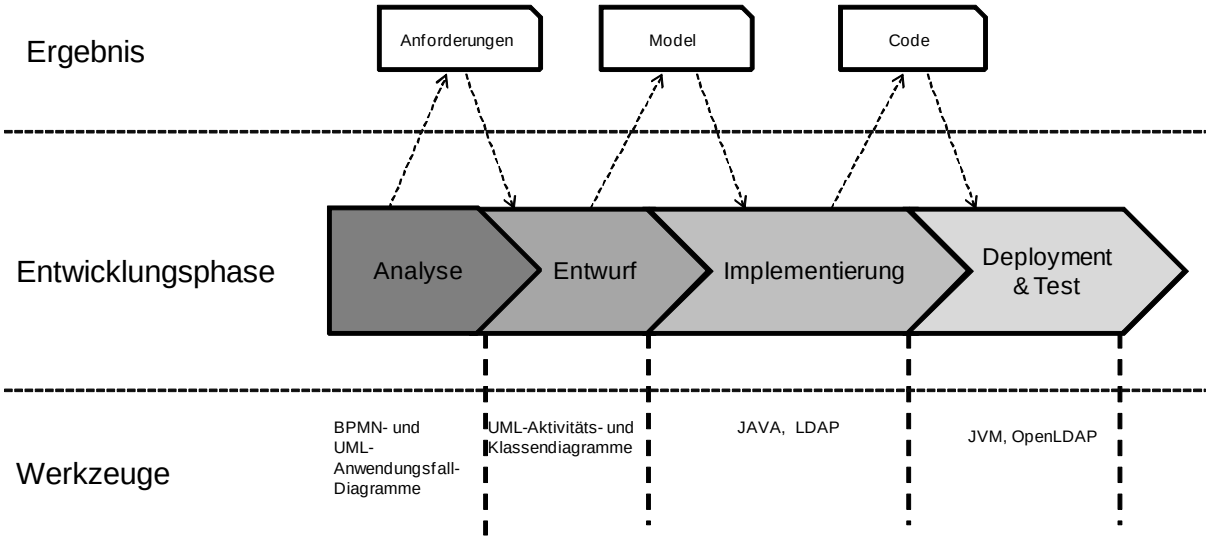


Abbildung 3 : Entwicklungsphasen

Die erste Phase ist die Analyse. Hier geht es darum, die Anforderungen zu erfassen und zu beschreiben, die die zu entwickelnde Softwarekomponente erfüllen soll. Ziel der Analyse ist, ein grobkörniges Modell der zu entwickelnden Anwendung zu erstellen, ohne implementierungsspezifische technische Details zu betrachten. Das Modell enthält ein statisches Teilmodell und ein dynamisches Teilmodell. Im statischen Teil werden Anwendungsfälle betrachtet, die die zu entwickelnde Verbindung unterstützen muss. Dafür werden UML-Anwendungsfall-Diagramme eingesetzt. Im dynamischen Teil werden mittels BPMN-Geschäftsprozessdiagrammen genau die vorher erwähnten Anwendungsfälle als Prozesse betrachtet. Die beiden Teilmodelle werden mittels UML-Aktivitäts- bzw. Klassendiagramme modelliert.

Die zweite Phase ist der Entwurf. Hier wird das in der Analyse erstellte Modell weiterentwickelt und darauf aufbauend eine Softwarearchitektur erstellt. Dabei wird das allgemeine Modell in eine konkrete Softwarearchitektur umgeformt, die Informationen über technische Umsetzungsdetails enthält und direkt als Vorlage für die Implementierung dient.

In der Implementierungsphase wird das entworfene Modell in teils Java-Code und teils in LDAP-Schema umgesetzt, sodass eine funktionierende Anwendung entsteht, die genau das entworfene Modell realisiert.

In der Schlussphase wird der Code ausgeführt, getestet und bewertet.

2 TECHNISCHE GRUNDLAGEN

Für ein besseres Verständnis der Arbeit werden die verschiedenen Technologien vorgestellt, die für die Entwicklung der Anwendung eingesetzt werden. Zum Modellieren der Anwendungsfälle, Ablaufprozesse und Softwarestrukturen wird die *Unified Modeling Language* (UML) benutzt. Zum Modellieren der Geschäftsprozesse ist der Einsatz der *Business Process Modelling Notation* (BPMN) vorgesehen. Die Datenhaltung der zu entwickelnden Anwendung basiert auf dem *Lightweight Directory Access Protocol* (LDAP). Für die Abfrage und Bereitstellung der Daten sind schließlich Webservices vorgesehen.

2.1 Business Process Modeling Notation

Die *Business Process Modeling Notation* (BPMN) ist eine grafische Spezifikationssprache, die Symbole zur Verfügung stellt, mit denen Geschäftsprozesse und Arbeitsabläufe (Workflows) modelliert werden können. Die Darstellung basiert grundlegend auf dem Konzept der Ablaufdiagramme. Dabei können ein oder mehrere Prozesse gekapselt werden. Die Diagramme sollen die Abbildung und die Entwicklung von Prozessen unter verschiedene Experten unterstützen [Ws04].

Die Grundkonstrukte von BPMN sind in 4 Kategorien unterteilt:

1. Ablaufobjekte (engl. Flow Objects) sind die Knoten in den Geschäftsprozessdiagrammen. Sie können ein Ereignis, eine Aktivität oder ein Entscheidungsoperator sein.

Ablaufobjekt	Beschreibung	Symbol
Ereignis	Ein Ereignis wird durch ein Kreis dargestellt und repräsentiert ein Geschehen im Geschäftsprozess. Es gibt drei Typen: Startereignis, Intermediate-Ereignis und Endereignis (siehe Symbole von rechts nach links).	
Aktivität	Eine Aktivität ist eine einzelne Tätigkeit oder ein Schritt im Prozess. Sie kann atomar oder zusammengesetzt sein und hat zwei Typen: Task oder Sub-Prozess. Der Sub-Prozess zeichnet sich durch ein kleines Plus-Zeichen in der unteren Mitte der Form.	
Entscheiduns-operator (Gateway)	Ein Entscheidungsoperator kontrolliert die Divergenz und Konvergenz des Sequenzflusses. Er definiert Entscheidung, Ablaufaufspaltung und Zusammenführung von Ablaufpfaden. Interne Markierungen zeigen das Kontrollverhalten	

Tabelle 1 : Ablaufobjekte in BPMN

2. Verbindungsobjekte (engl. Connecting Objects) sind die verbindenden Kanten in den Geschäftsprozessdiagrammen. Sie sind entweder ein Sequenzverbinder, ein Nachrichtenfluss oder eine Assoziation.

Verbindungsobjekt	Beschreibung	Symbol
Sequenzverbinder	Ein Sequenzverbinder stellt die Reihenfolge, in der Aktivitäten ausgeführt werden, dar.	
Nachrichtenfluss	Ein Nachrichtenfluss zeigt den Nachrichtenaustausch zwischen zwei Prozessteilnehmer (Geschäftsrollen).	
Assoziation	Eine Assoziation assoziiert Daten, Text und andere Artefakte mit Ablaufobjekte.	

Tabelle 2 : Vebindungsobjekte in BPMN

3. Schwimmbahnen (engl. Swimlanes) sind die Bereiche, mit denen Aktoren und Systeme dargestellt werden. Diese Konstrukte bestehen aus einem Pool oder einer Bahn.

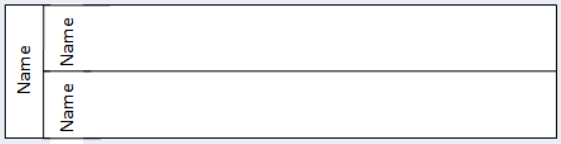
Schwimmbahn	Beschreibung	Symbol
Pool	Ein Pool repräsentiert einen Teilnehmer in einem Prozess.	
Bahn	Eine Bahn ist eine Unterpartition innerhalb eines Pools. Sie wird benutzt um Aktivitäten zu organisieren und kategorisieren.	

Tabelle 3 : Schwimmbahnen in BPMN

4. Artefakte (engl. Artifacts) sind weitere Elemente wie Datenobjekt, Gruppierung und Annotation.

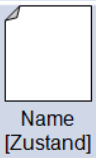
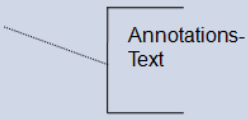
Artefakte	Beschreibung	Symbol
Ereignis	Ein Datenobjekt ist ein Mechanismus, der zeigt, welche Daten Aktivitäten austauschen. Sie werden durch Assoziationen mit Aktivitäten verbunden.	
Gruppierung	Eine Gruppierung dient nur zu Dokumentationszwecken und hat keinen Einfluss auf den Prozess.	
Annotation	Eine Annotation ermöglicht das Hinzufügen von zusätzlichen Informationen.	

Tabelle 4 : Artefakte in BPMN

Um nun ein einen konkreten Eindruck von BPMN-Modelle zu bekommen, wird ein Beispiel präsentiert, das mit dem Eclipse-BPMN-Modellierer erstellt wurde.

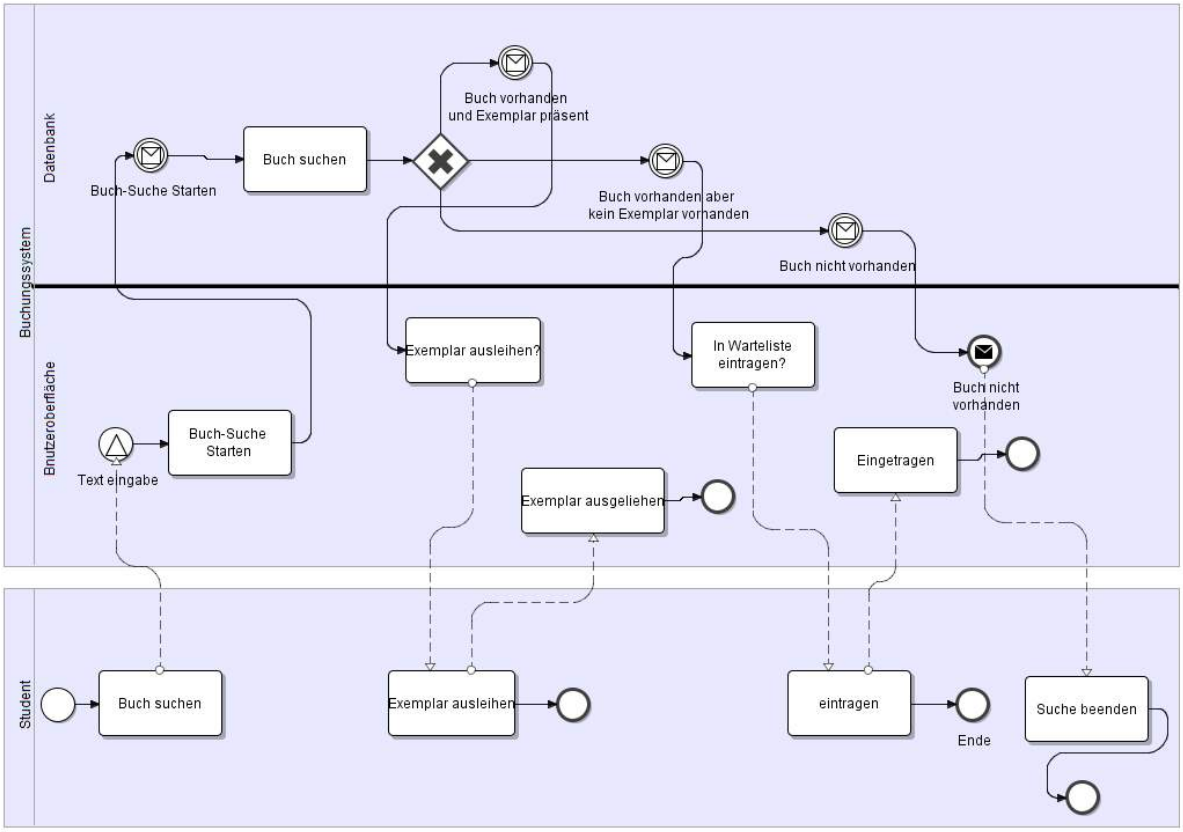


Abbildung 4 : BPMN-Beispiel

Das BPMN-Modell im Bild (Abbildung 4) zeigt einen Geschäftsprozess in einem Bibliotheksbuchungssystem. Es gibt zwei Hauptteilnehmer (oder Hauptrollen), *Student* und *Buchungssystem*, die durch die beiden Bahnen repräsentiert werden. Das Buchungsystem besteht wiederum aus zwei Teilnehmern, *Benutzeroberfläche* und *Datenbank*. Der Student fängt einen Teilprozess an, stoßt damit einen anderen Teilprozess im Buchungssystem an. Dies wird durch das Entsenden einer Nachricht vom Pool *Student* zum Pool *Buchungssystem* dargestellt. Das Startereignis *Texteingabe* ist ein Signalereignis. Signalereignisse werden mit dem Dreieck dargestellt. Die Message-Ereignisse wie „Buch-Suche starten“ sind anhand des Briefumschlagsymbols zu erkennen.

2.2 Unified Modeling Language

Die *Unified Modeling Language* (UML) ist eine standardisierte Sprache (Notation) für die Modellierung, Spezifikation, Visualisierung, Konstruktion und Dokumentation von Modellen für Softwaresysteme, Geschäftsmodelle und andere Systeme [UML]. Sie wird von Entwicklern bei dem Entwurf und der Entwicklung von Softwaremodellen eingesetzt, um auf einheitlicher Basis zu diskutieren und sich auszutauschen. Die *Object Management Group* [OMG] ist die Organisation, die sich mit der Entwicklung und Standardisierung von UML beschäftigt. In der OMG sind etwa 800 Unternehmen vereint, die gemeinsame Standards entwickeln, die die objektorientierte Softwareentwicklung unterstützen.

UML hat als Modellierungssprache geometrische Symbole, mit denen Modelle von Software erstellt werden können. Wenn diese geometrischen Symbole zusammengesetzt werden, ergeben sie Diagramme, die ganz bestimmte Strukturen einer Software oder Abläufe in einer Software darstellen. Je nach Diagrammtyp werden unterschiedliche Aspekte der Software hervorgehoben. Damit wird die Software aus verschiedenen Blickwinkeln beschrieben. Ein Diagramm stellt also ein Modell dar, das ganz bestimmte Aspekte der Software beschreibt [BR+06].

Um dieses Ziel zu erreichen differiert UML 13 Diagrammtypen, die sich in zwei Kategorien unterteilen:

Verhaltensdiagramme:

- Anwendungsfalldiagramm: stellt Beziehungen zwischen Akteuren und Anwendungsfällen dar
- Aktivitätsdiagramm: beschreibt einen Ablauf, der aus einzelnen Aktivitäten besteht
- Zustandsdiagramm: zeigt eine Folge von Zuständen eines Objekts
- Sequenzdiagramm: zeigt den zeitlichen Ablauf von Nachrichten zwischen Objekten
- Kommunikationsdiagramm: zeigt Beziehungen und Interaktionen zwischen Objekten
- Zeitverlaufsdiagramm: Interaktionsdiagramm mit Zeitverlaufskurven von Zuständen
- Interaktionsübersichtsdiagramm: Interaktionsdiagramm zur Übersicht über Abfolgen von Interaktionen

Strukturdiagramme:

- Klassendiagramm (wichtigstes Diagramm): zeigt Klassen und ihre Beziehungen untereinander
- Paketdiagramm: gliedert Softwaresysteme in Untereinheiten
- Objektdiagramm: beschreibt Objekte, Assoziationen und Attributwerte zu einem bestimmten Zeitpunkt während der Laufzeit
- Kompositionsstrukturdiagramm: zeigt Abbildung innerer Zusammenhänge einer komplexen Systemarchitektur
- Komponentendiagramm: beschreibt Komponenten und ihre Beziehungen und Schnittstellen
- Verteilungsdiagramm: Einsatzdiagramm, Knotendiagramm, Laufzeitumfeld

Anwendungsfalldiagramm

Ein Anwendungsfalldiagramm beschreibt das Verhalten eines Systems aus der Sicht eines äußeren Beobachters. Es besteht aus einer Menge von Anwendungsfällen und ihre Beziehungen zu den Akteuren. Ein Anwendungsfall ist eine Zusammenfassung von Aufgaben, die ein bestimmtes Ziel erfüllen. Akteure sind Rollen, die durch Menschen oder Gegenstände gespielt werden, die diese Aufgaben durchführen.

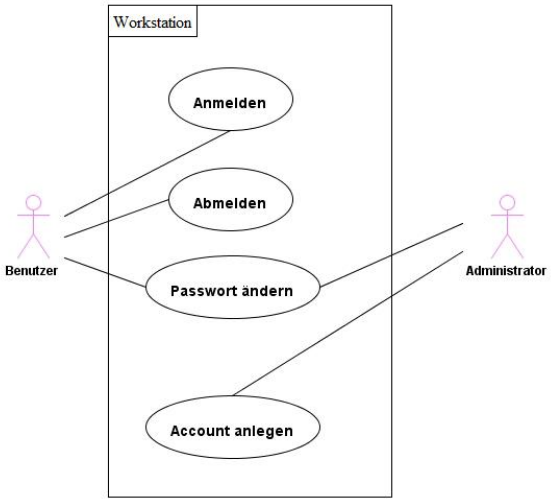


Abbildung 5 : Anwendungsfalldiagramm-Beispiel

Das Beispiel (Abbildung 5) zeigt übliche Anwendungsfälle an einer Workstation. Der Benutzer kann sich anmelden, abmelden und sein Passwort ändern. Der Administrator kann einen Account für einen Benutzer anlegen und jedes Passwort ändern. Die Anwendungsfälle sind durch Ellipsen und die Akteure sind durch Strichmännchen dargestellt. Die entsprechenden Anwendungsfälle und Akteure durch Linien miteinander verbunden. Die Systemgrenze wird durch einen Rahmen um die Anwendungsfälle symbolisiert.

Klassendiagramm

Eine Klasse ist eine Zusammenfassung gleichartiger Objekte, die dieselben Eigenschaften (Attribute) und Fähigkeiten (Operationen/Methoden) haben. Objekte sind die agierenden Grundelemente einer Anwendung. Also ist eine Klasse die Schablone, mit der ein Objekt erzeugt wird.

In UML ist das Klassendiagramm der wichtigste Diagrammtyp für die objektorientierte Software-Entwicklung. Es stellt sowohl Zusammenhänge zwischen Klassen als auch den Aufbau von einzelnen Klassen dar.

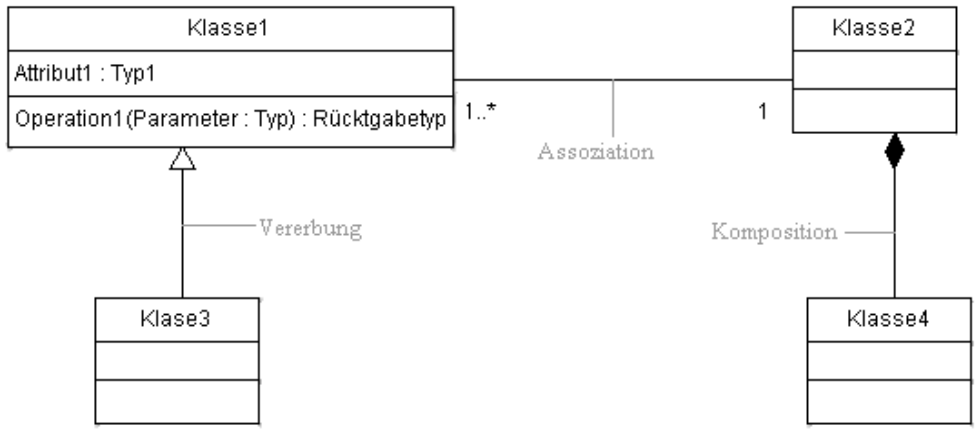


Abbildung 6 : Klassendiagramm-Beispiel

Wie im Beispiel (Abbildung 6) zu sehen ist, werden Klassen durch Rechtecke dargestellt, die den Namen der Klasse und eventuell die Attribute und Operationen der Klasse enthalten. Klassenname, Attribute und Operationen werden durch eine horizontale Linie getrennt. Der Klassenname steht im Singular und beginnt mit einem Großbuchstaben. Attribute können näher beschrieben werden, z.B. durch ihren Typ, einen Initialwert und Zusicherungen. Sie werden aber mindestens mit ihrem Namen aufgeführt. Operationen können ebenfalls durch Parameter, Initialwerte, Zusicherungen usw. beschrieben werden. Auch sie werden mindestens mit ihrem Namen aufgeführt. Die Vererbung wird mit einem geschlossenen und durchgezogenen Pfeil repräsentiert, der von der abgeleiteten Unterklasse zur Oberklasse zeigt. Die Oberklasse ist eine Generalisierung der Unterklasse und umgekehrt ist die Unterklasse eine Spezialisierung der Oberklasse. Assoziationen stellen Beziehungen zwischen Klassen dar und werden mit einem Strich dargestellt. Assoziationen können auch gerichtet sein (offener Pfeil), um die Richtung der Beziehung darzustellen. Sie können auch Multiplizitäten besitzen, die angeben, ob es sich zwischen zwei Klassen um eine 1:1-, 1:n- oder n:m-Beziehung handelt. Aggregationen und Kompositionen sind spezielle Assoziationen, die „Teile/Ganzes“-Beziehungen und „Hat-eine“-Beziehungen darstellen. Bei der Aggregation können die „Teile“ des „Ganzen“ auch einzeln existieren, bei der Komposition nur, wenn auch das „Ganze“ existiert. Die Verbindungslinie erhält auf der Seite des „Ganzen“ eine Raute, die bei der Aggregation ungefüllt und bei der Komposition gefüllt ist.

Komponentendiagramm

Das Komponentendiagramm wird in der Softwareentwicklung vor allem für die Modellierung von komponentenbasierten Softwaresystemen eingesetzt, damit bei späterer Implementierung der Softwarelösung die Abhängigkeiten klar sind. Dafür werden die Zusammenhänge der einzelnen Komponenten der späteren Softwarelösung in einem Komponentendiagramm dargestellt. Das Diagramm besteht aus Komponenten, deren Schnittstellen und Ports. Notationselemente, die ansonsten in Klassen- oder Kompositionsstrukturdiagrammen verwendet werden, können auch hier benutzt werden.

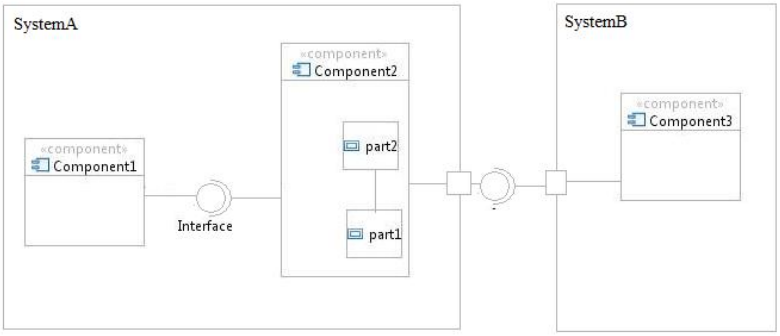


Abbildung 7 : Komponentendiagramm-Beispiel

Das Diagramm (Abbildung 7) zeigt das Beispiel eines Komponentendiagramms.

Aktivitätsdiagramm

Das Aktivitätsdiagramm beschreibt im Allgemeinen Abläufe. Es zeigt die Vernetzung von Aktivitäten, deren Verbindungen mit Kontroll- und Datenflüssen, und in welcher Reihenfolge diese Aktivitäten ausgeführt werden, dargestellt werden. Mit einem Aktivitätsdiagramm wird meist der Ablauf eines Anwendungsfalls beschrieben, es eignet sich aber zur Modellierung

aller Aktivitäten innerhalb eines Systems. Eine Aktivität ist ein einzelner Schritt bzw. eine elementare Aktion innerhalb eines Programmablaufes.

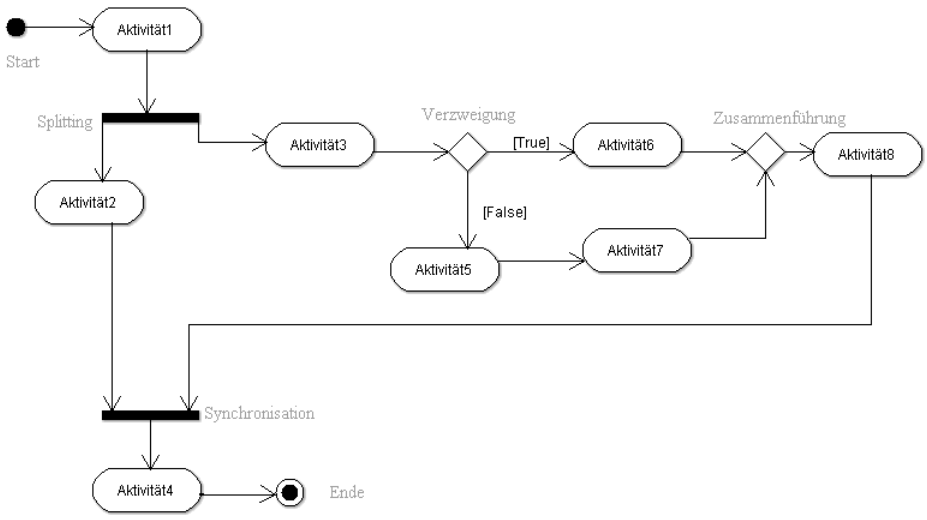


Abbildung 8 : Aktivitätsdiagramm-Beispiel

Wie im Beispiel (Abbildung 8) gezeigt wird, wird eine Aktivität durch ein Rechteck mit abgerundeten Seiten visualisiert. Sie enthält eine Beschreibung der internen Aktion. Von der Aktivität aus gehen die Transitionen, die den Abschluss der internen Aktion und den Übergang zur nächsten Aktivität darstellen. Verzweigungen, die aufgrund einer Bedingung stattfinden, werden mit einem Rautensymbol dargestellt. Die Verzweigungsbedingungen werden die in eckigen Klammern angegeben. Bei der Zusammenführung geht der Aktivitätsfluss weiter, sobald eine der eingehenden Transitionen eintrifft. Hingegen geht es bei der Synchronisation erst weiter, wenn beide eingehende Transitionen eintreffen. Das Splitting wird eingesetzt, wenn der Aktivitätsfluss auf mehrere Aktivitäten verteilt wird.

2.3 Lightweight Directory Access Protocol

Das *Lightweight Directory Access Protocol* (LDAP) ist ein standardisiertes Protokoll zum Zugriff auf Verzeichnisdienste. Der Zugriff kann zum Erstellen, Modifizieren oder Suchen von Einträgen in einem Verzeichnisdienst verwendet werden. LDAP funktioniert auf Basis des Netzwerktransportprotokolls TCP/IP und befindet sich in dem ISO/OSI-Modell auf der Anwendungsebene. Es beschreibt die Kommunikation, die auf Anfragen zwischen einem LDAP-Client und einem Verzeichnis-Server basiert.

LDAP ist eine vereinfachte Version des Directory Access Protocols (DAP), das ein Teil der X.500-Standard-Spezifikation ist. Der X.500-Standard ist sehr umfangreich und setzt auf einem vollständigen ISO/OSI-Stapel auf, was die Implementierung aufwendig und hardwareintensiv machte. Das behinderte die breitere Verteilung in vielen Unternehmen und Institutionen. Das Ziel von LDAP, das 1993 an der Universität von Michigan entwickelt wurde, war deswegen, Verzeichnisdienste einfacher und leichter handhabbar zu machen. LDAP setzt nur auf dem TCP/IP-Stapel auf und implementiert nur eine Auswahl der DAP-Funktionen und Datentypen. Dadurch ist die Implementierung von LDAP viel einfacher, was ihm eine große Popularität in der Anwendung ermöglichte.

Neben dem Protokoll besteht LDAP aus einem Verzeichnis. Ein Verzeichnis ist eine Sammlung von Objekten mit ähnlichen Eigenschaften, die in einer logischen und hierarchischen Art verteilt sind. Das bekannteste Beispiel von einem Verzeichnis ist das Telefonbuch, das aus einer Reihe von Namen besteht, die in alphabetischer Reihenfolge organisiert sind, wobei jeder Name mit einer Adresse und Telefonnummer angegeben ist. Das LDAP-Verzeichnis ist ein Baum von Verzeichniseinträgen. Ein Eintrag besteht aus einer Reihe von Attributen. Jedes Attribut hat einen Namen und einen oder mehreren Werte. Jeder Eintrag hat eine eindeutige Kennung: *Distinguished Name* (DN). Diese besteht aus seinem *Relative Distinguished Name* (RDN), der von einem Attribut des Eintrags gefolgt von dem DN des Vatereintrags gebaut wird. Man könnte sich den DN als einen vollständigen Dateinamen und den RDN als relativen Dateinamen in einem Ordner vorstellen. Das Beispiel im Bild (Abbildung 9) zeigt einen LDAP-Verzeichnis-Baum mit einigen DNs.

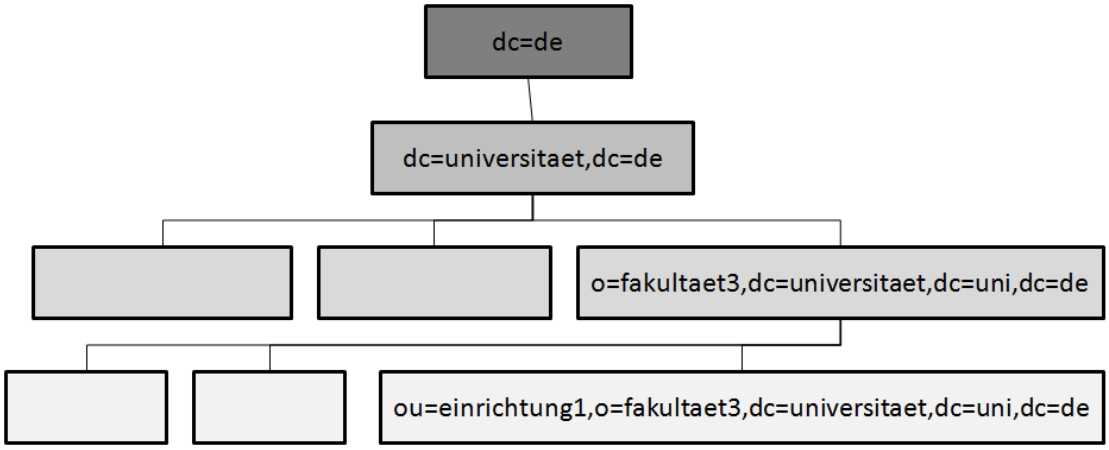


Abbildung 9 : Beispiel eines LDAP-Verzeichnisbaums

Jeder Eintrag in einem LDAP-Verzeichnis beschreibt ein Objekt, das eine konkrete Instanz einer oder mehrerer Objektklassen. Eine Objektklasse ist eine verallgemeinerte Beschreibung für dieses und gleichartige Objekte. Die Objektklassen können erben und vererben. Die Objektklasse verfügt über eine Liste der zwingend vorgeschriebenen (mandatory oder required) und der nicht verbindlich vorgeschriebenen (optionalen) Attribute. Diese werden durch ihren Namen und den Datentyp definiert. Zusätzlich werden auf ihnen Vergleichsoperationen und Ordnungen definieren. Attributnamen sind im Verzeichnis eindeutig und können grundsätzlich mehr als einen Wert speichern.

Die Definitionen der Attribute und Objektklassen werden in einem Schema eingetragen. Ein Schema stellt damit eine Sammlung von Strukturdefinitionen (Metadaten) dar:

- Definitionen der zu verwendenden Attributtypen
- Definitionen der zu verwendenden Objektklassen
- Filter/Matching-Regeln bei Vergleichsoperationen
- Rechte zum Anlegen und Modifizieren von Datensätzen

```
attributetype ( 2.16.840.1.113730.3.1.4
NAME 'employeeType'
DESC 'RFC2798: type of employment for a person'
EQUALITY caseIgnoreMatch
SUBSTR caseIgnoreSubstringsMatch
SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 )
Code 1 : Beispiel einer Attributdefinition

objectclass     ( 2.16.840.1.113730.3.2.2
NAME 'inetOrgPerson'
DESC 'RFC2798: Internet Organizational Person'
SUP organizationalPerson
STRUCTURAL
MAY (
          audio $ businessCategory $ carLicense $ departmentNumber $
          displayName $ employeeNumber $ employeeType $ givenName $ homePhone $
          homePostalAddress $ initials $ jpegPhoto $
          labeledURI $ mail $ manager $ mobile $  $ pager $ photo $ roomNumber $
          secretary $ uid $ userCertificate $
          x500uniqueIdentifier $ preferredLanguage $
           userSMIMECertificate $ userPKCS12 )
                          )
```

Codefragment 1 : Beispiel einer Attributdefinition und einer Objektklassendefinition

Es gibt verschiedene Schemata, die von den verschiedenen LDAP-Implementierungen mitgeliefert werden. Diese Schemata sind im Allgemeinen ausreichend für die üblichen Einsätze von LDAP. Sollten diese aber nicht ausreichen, kann man eigene definieren oder bestehende erweitern. Hier im Beispiel (Codefragment 1) ist ein Auszug aus dem mitgelieferten InetOrgPerson-Schema des OpenLDAPs, das in Kapitel 4 ausführlicher beschrieben wird.

2.4 Webservices

Ein Webservice ist definiert durch das *World Wide Web Consortium* [W3C] als *„ein Software-System zur Unterstützung interoperabler Maschine-zu-Maschine-Interaktion über ein Netzwerk*“. Somit ist ein Webservice eine im Netz auf einem Remote-System bereitgestellte Softwarekomponente, die eine Abstraktionsebene einer Anwendungslogik darstellt. Auf den Webservice kann über Internetstandardprotokolle zugegriffen werden. Für die Codierungen der Nachrichten wird die *Extended Markup Language* (XML) genutzt, um eine einfache Bereitstellung und hohe Verfügbarkeit von Webservices gewährleisten zu können.

Webservices sind ein Hauptbestandteil der serviceorientierten Architektur (SOA) und bieten Dienste an Dienstnehmer oder sog. Client-Programme. Ein Client sendet eine Anfrage an einen Webservice und dieser antwortet mit der entsprechenden Antwort.

2.4.1 Simple Object Access Protocol

Das *Simple Object Access Protocol* (SOAP) ist ein leichtgewichtiges Netzwerkprotokoll, das ein Nachrichtenformat spezifiziert. Aber zur Übermittlung der Nachricht wird noch ein Übertragungsprotokoll benötigt. Meistens wird SOAP in Verbindung mit dem *Hypertxt Transport Protocl* (HTTP) als Transportprotokoll für den Aufruf von Webservices benutzt. Ein SOAP-Dokument ist ein XML-Dokument, welches eine ähnliche Struktur wie die eines HTML-Dokuments hat. Es enthält einen *Head*, wo verschiedene Zusatzinformationen eingebettet werden können, sowie einen *Body*, in dem die eigentlichen Informationen transportiert werden [W3Cs00].

2.4.2 Web Service Description Language

Die *Web Service Description Language* (WSDL) ist eine XML-basierte Sprache zur Beschreibung von Webservices als eine Menge von Schnittstellen zum Austausch von Nachrichten. Die Webservice-Funktionen und die ausgetauschten Nachrichten werden abstrakt definiert und an einem konkreten Netzwerkprotokoll und Nachrichtenformat eingebunden, um eine Webservice-Schnittstelle zu definieren. Die Beschreibung eines Webservice mittels WSDL beinhaltet die Beschreibung von Funktionen mit ihren Parametern und Rückgabewerten, Daten, Datentypen und die benutzten Netzwerkprotokolle [W3Cw01]. Ein WSDL-Dokument besteht aus zwei Teilen, einem ersten Teil, wo abstrakte Definitionen vorgenommen werden, und einem zweiten Tei, wo konkrete Definitionen stattfinden.

Abstrakte Definitionen

Types: Definition der Datentypen, die zum Austausch der *messages* benutzt werden.
Message: Abstrakte Definitionen der übertragenen Daten, bestehend aus mehreren logischen Teilen, von denen jedes mit einer Definition innerhalb eines Datentypsystems verknüpft ist.
portType: Eine Menge von abstrakten Arbeitsschritten

Konkrete Beschreibungen

Binding: Bestimmt das konkrete Protokoll und das Datenformat für die Arbeitsschritte und Nachrichten, die durch einen bestimmten Port-Typ gegeben sind. Ports Spezifizieren eine Adresse für eine Bindung, also eine Kommunikationsschnittstelle, üblicherweise ein URI. In WSDL 2.0 wurde die Bezeichnung für den Endpoint geändert.
Services: Fassen eine Menge von verwandten Ports zusammen.

Ein WSDL Dokument ist eine Menge von Definitionen, die hierarchisch aufgebaut sind. Es gibt ein Definitionselement in der Wurzel und andere Definitionselemente innerhalb des baumartigen Dokuments. Die Grammatik ist anhand dieses Beispiels unten im Bild (Codefragment 2Codefragment 1) gezeigt. Das Beispiel handelt von einem Webservice, der Studentenindentitätsdaten auf Abruf bereitstellt. Ziel hier ist, ein allgemeines Bild zu präsentieren, wie eine WSDL-Datei strukturiert ist. Auf die einzelnen Details wird nicht eingegangen.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns:tns="http://services.kim.uni-karlsruhe.de/2008/10/ATISWebService"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="http://schemas.xmlsoap.org/wsdl/" name="ATISWebService">
<types>
<xsd:schema>
<xsd:import namespace="http://services.kim.uni-karlsruhe.de/2008/10/ATISWebService"
            schemaLocation="http://1[redacted]:6080/WebService/ATISWebService?xsd=1"/>
</xsd:schema>
</types>
<message name="ATISprovisionStudent">
<part name="parameters" element="tns:ATISprovision"/>
</message>
<message name="ATISprovisionStudentResponse">
<part name="parameters" element="tns:ATISprovisionStudentResponse"/>
</message>
<portType name="ATISSPMLWebService">
<operation name="ATISprovisionStudentWithMatrNr">
<input message="tns:provisionStudent"/>
<output message="tns:provisionStudentResponse"/>
</operation>
</portType>
<binding name="ATISWebServicePortBinding" type="tns:ATISWebService">
<soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
<operation name="ATISprovisionStudent">
<soap:operation soapAction=""/>
<input>
<soap:body use="literal"/>
</input>
<output>
<soap:body use="literal"/>
</output>
</operation>

</binding>
<service name="WebService">
<port name="ATISWebServicePort" binding="tns:ATISWebServicePortBinding">
<soap:address location="http://[redacted]:6080/WebService/ATISWebService"/>
</port>
</service>
</definitions>
```

Codefragment 2 : Beispiel einer WSDL-Datei

Die erste Zeile besagt, dass es von einem XML-Dokument handelt. Die nächste Zeile ist das *definitions*-Wurzelelement im WSDL-Dokument. Innerhalb dieses Elements werden Namespace-Attribute angegeben. Diese Attribute können entweder mit „*xmlns* für „*XML-Namespace*“ oder „*TargetNamespace*“ angegeben werden. Jedes Namespace-Attribut definiert ein Kürzel für jeden im Dokument verwendeten Namespace. Zum Beispiel ist „*xmlns:xsd*“ für den Namespace *http://www.w3.org/2001/XMLSchema*. Mithilfe dieses Kürzels kann im Dokument auf diesen Namespace verwiesen werden, indem vor dem Namen "*xsd*:" gesetzt wird. Die Namespaces dienen dazu, Namenskonflikte zu vermeiden. Wenn man verschiedene Webservices benutzt, können diese Elemente zwar die gleiche Bezeichnung, aber verschiedene Bedeutungen haben. Dies führt zu gravierenden Fehlern.

Das *types*-Element enthält den *Types*-Abschnitt der WSDL-Struktur. Hier werden die Datentypen definiert, die für den Nachrichtenaustausch relevant sind. WSDL-Datentypisierung basiert auf XML-Schema (XSD), das jetzt zur W3C-Empfehlung geworden ist. *ATISSPMLWebservice* verwendet ein eigenes Schema, das unter „*schemaLocation*“ angegeben wird. Das Schema enthält Elemente und Typen, auf die im nächsten Teil bescrieben werden.

Der *Message*-Abschnitt beschreibt die Eingabe und Rückgabeparameter der angebotenen Webservice-Operationen in Form von Nachrichten. Jedes untergeordnete *part*-Element im *message*-Element beschreibt einen Parameter. Der Name eines *message*-Elements für die Ausgabe endet per Vereinbarung auf „*Response*“. Das *part*-Element verfügt über ein *name*- und ein *type*-Attribut, so wie ein Funktionsparameter auch einen Namen und einen Typ besitzt.

Der *PortTypes*-Abschnitt definiert durch die *portType*-Elemente die Webservice-Operationen in den *operation*-Elementen. Der Webservice hat ein *operation*-Element: *ATIS_provisionStudent*. Dieser Name steht für die Funktionsnamen, die der Webservice anbietet. Das *operation*-Element verfügt über zwei untergeordnete Elemente: *input* und *output*. Das *message*-Attribut in jedem *input*- und *output*-Element verweist auf das entsprechende *message*-Element im *messages*-Abschnitt.

Das *binding*-Element hat den Namen „*ATISWebServicePortBinding*“, damit das *port*-Element im *services*-Abschnitt darauf verweisen kann. Es besitzt ein *type*-Attribut, das auf ein *portType*-Element verweist.

Das *soap-binding*-Element legt das verwendete Format „*document*" fest. Das *transport*-Attribut verweist auf den Namespace, der das verwendete HTTP-Protokoll beschreibt. Darüber hinaus verfügt es über ein *type*-Attribut, der den Namespace angibt, der die verwendeten *Types* beschreibt.

3 ANALYSE

In diesem Kapitel wird das bestehende System ATIS-IdM in Bezug auf seine Funktionalität analysiert. Dies soll das Verständnis für die Fragestellung und die Problematik verdeutlichen. Es werden zwei Anwendungsfälle mit UML-Anwendungsfalldiagrammen untersucht. Nach der Betrachtung der Anwendungsfälle werden die Details anhand von Geschäftsprozessen aufgezeigt und näher untersucht. Dafür werden die jeweils zugehörigen Geschäftsprozesse mittels BPMN modelliert. Die resultierenden Geschäftsprozessdiagramme beschreiben den Ablauf der Aktivitäten, die mit den Anwendungsfällen verbunden sind. Somit wird eine prozessbezogene Sicht auf die dynamischen Aspekte der modellierten Fälle gezeigt. Davon ausgehend wird das Optimierungspotenzial analysiert, das die Inanspruchnahme des KIM-IdM-Dienstes bietet. Dieses Potenzial wird als Optimierung der zuvor analysierten Anwendungsfälle vorgestellt und mittels UML-Anwendungsfall- und BPMN-Geschäftsprozessdiagrammen dargestellt.

3.1 Erster Anwendungsfall

Der erste Anwendungsfall, bei dem die Erweiterung des ATIS-IdM-Systems um den KIM-IdM-Provisionierungsdienst von großem Nutzen wäre, ist das Anlegen eines neuen Accounts für einen neuen Studenten durch einen ATIS-Mitarbeiter (siehe 1.2 Szenario). Das ist einer der zentralen Anwendungsfälle des Identitätsmanagements für den Betrieb des Studentenpools in der Fakultät, da erst nach dem Anlegen eines Accounts ein Student im Pool arbeiten kann.

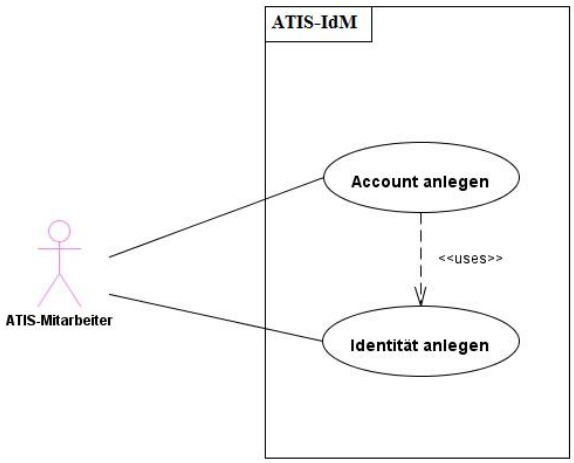


Abbildung 10 : Anwendungsfalldiargam "Account anlegen"

Jeder Account wird genau einer Identität zugewiesen. Deshalb muss zuerst eine Identität bzw. ein Identitätsobjekt des Studenten im System vorhanden sein, zu der dann ein Account angelegt wird. Wenn diese Identität nicht vorhanden ist, muss sie zunächst angelegt werden. Das zeigt, wie stark der Account von der Identität abhängt. Das Anwendungsfalldiagramm (**Fehler! Verweisquelle konnte nicht gefunden werden.**) beschreibt diesen Anwendungsfall.

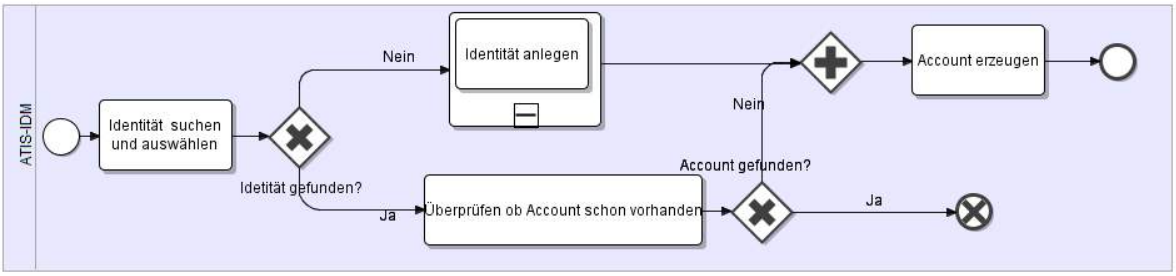


Abbildung 11 : Geschäftsprozessdiagramm "Account anlegen"

Das BPMN-Diagramm (Abbildung 11) liefert eine dynamische Sicht auf den Anwendungsfall. Nach dem Start des Prozesses ist die erste Aktivität „*Identität suchen und auswählen*“. Bei dieser ersten Aktivität muss eine Identität gesucht und ausgewählt werden, falls sie vorhanden ist, zu der dann der Account zugeordnet wird. Als Nächstes kommt das Gateway “*Identität gefunden?*“, das den Entscheidungspunkt für die letzte Aktivität darstellt. Dieses Subprozess “*Identität anlegen*“ wird angestoßen, wenn keine Identität des Studenten gefunden werden konnte. Andernfalls wird die Aktivität “*Überprüfen, ob Account schon vorhanden ist*“ ausgeführt. Hier wird überprüft, ob zu der gefunden Identität nicht schon ein zugehöriger Account existiert, um ihn nicht mehrfach anzulegen. Existiert noch keiner, wird ein Account erzeugt, wenn die notwendigen Schritte schon abgeschlossen sind. Wie in (Abbildung 11) gezeigt, könnten aber zu der Identität auch zusätzliche weitere Accounts hinzugefügt werden, beispielsweise für Tests. Diese Möglichkeit ist aber nicht der Standardfall, sondern die Ausnahme.

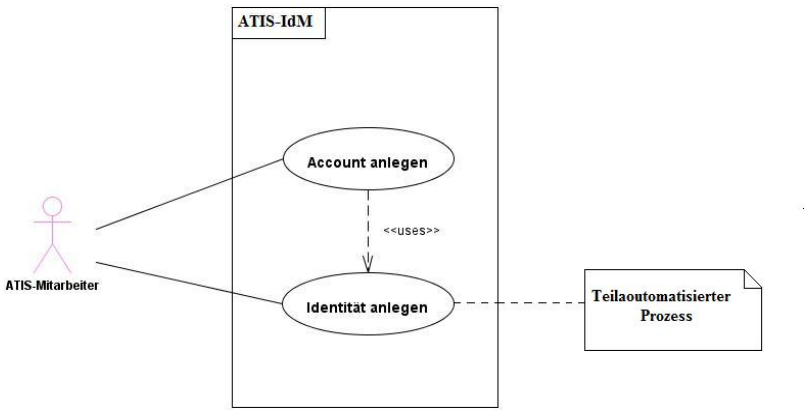


Abbildung 12 : Anwendungsfalldiagramm "Account anlegen (optimiert)"

Die Nutzung des Provisionierungsdienstes von KIM-IdM bietet dem ATIS-IdM in diesem Anwendungsfall (Abbildung 12) ein Optimierungspotenzial: Das Anlegen einer Identität könnte teilautomatisiert werden, wenn die benötigten Daten der Identitätsattribute nicht manuell eingetippt, sondern automatisch nach Abruf vom KIM bereitgestellt werden. Damit werden der manuelle Aufwand und die damit verbundenen Fehler verringert. Zusätzlich werden dadurch zuverlässige aktuelle Identitätsdaten verwendet, um einen neuen Account anzulegen.

Der Prozess „Account anlegen“ aus Abbildung 11 wird nach der Optimierung bzw. Erweiterung fast identisch bleiben. Nur der darin eingebundene Subprozess „Identität anlegen“ wird so geändert werden, dass die Daten bei KIM-IdM abgefragt und nach ihrer Lieferung vom ATIS-Mitarbeiter benutzt werden, um die Identität zu generieren.

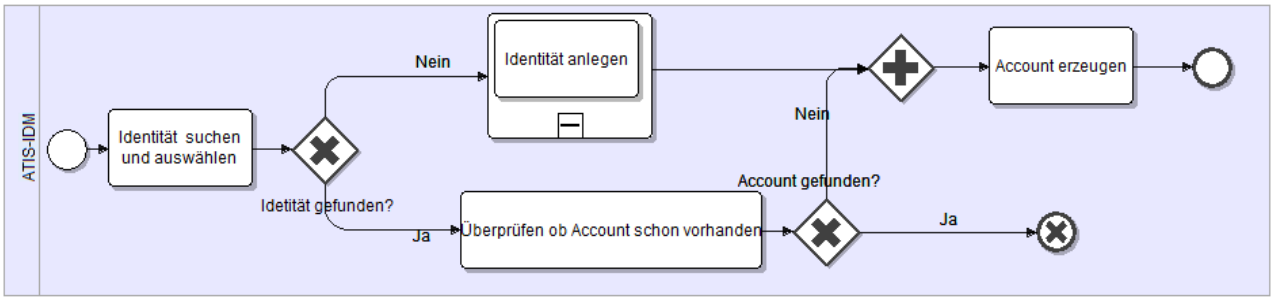


Abbildung 13 : Geschäftsprozessdiagramm "Identität anlegen"

Das Diagramm (Abbildung 13) beschreibt den optimierten Subprozess „Identität anlegen“. Das Start-Message-Event „*Identitätsdaten abrufen*“ zeigt, dass als erstes ein „*eindeutiges Attribut*“ des gesuchten Studenten von ATIS-IdM an KIM-IDM übermittelt werden soll. Diese Übermittlung geschieht in Form einer Anfrage. Das eindeutige Attribut soll jeden Benutzer eindeutig identifizierbar machen. Wenn KIM-IdM die Anfrage empfängt, provisioniert es mit dem vereinbarten Satz an Attributswerten. Mit den empfangenen Daten wird schließlich eine Identität generiert. Um welches eindeutige Attribut (Datum) es sich handelt, muss vorab natürlich zwischen ATIS-IDM und KIM-IDM festgelegt worden sein.

3.2 Zweiter Anwendungsfall

Der zweite Anwendungsfall ist „Daten aktualisieren“. Dieser Anwendungsfall tritt auf beim Verändern der Identitätsdaten eines Studenten, z.B. beim Beginn eines neuen Semesters, bei der Namensänderung, beim Fachwechsel oder bei der Exmatrikulation. Zuerst müssen diese Identitätsdaten aktualisiert werden. Danach können entsprechend die zugehörigen Account-Daten bzw. Accounts aktualisiert werden (Abbildung 14). Wenn ein neues Semester beginnt, werden alle Studenten überprüft, ob sie sich für das neue Semester rückgemeldet haben bzw. nicht, um ihre Accounts zu verlängern bzw. zu sperren. Wenn ein Benutzer seinen Namen ändert (z.B. wegen Heirat), dann muss sein Anmeldename (Login) und seine Email-Adresse geändert werden. Im Falle eines Fachwechsels oder einer Exmatrikulation erfüllt der Poolbenutzer nicht mehr die Bedingungen, die ihm das Recht auf die Poolraumbenutzung gewährt.

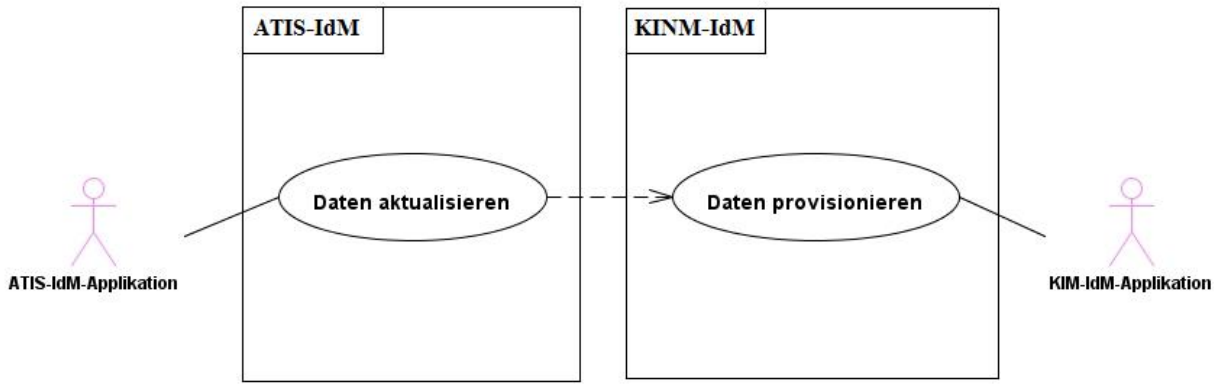


Abbildung 14 : Anwendungsfalldiagranmm "Daten aktualisieren"

Dieser Anwendungsfall ist ebenso von zentraler Bedeutung, da er öfters auftritt und eine enorme Rolle bei der Sicherheit des IdM-Systems spielt, wie schon in der Motivation erläutert wurde. In diesem Fall bietet die Kopplung von KIM-IdM und ATIS-IdM die Optimierungdmöglichkeit, Änderungen die in KIM-IdM auftreten zeitnah an ATIS-IdM weiterzuleiten und damit eine große Synchronisation zu erreichen. Wie im Diagramm (Abbildung 14) zu sehen ist, hängt die Datenaktualisierung (Identitätsdaten) in ATIS-IdM, die vom Aktor „*ATIS-IdM-Applikation*“ durchgeführt wird, von der Datenprovisionierung vom KIM-IdM ab, die vom „*KIM-IdM-Applikation*“ durchgeführt wird. Die Datenaktualisierung kann deshalb nur basierend auf den Daten der Provisionierung erfolgen. Der genauere Ablauf dieses Prozesses wird im Folgenden untersucht.

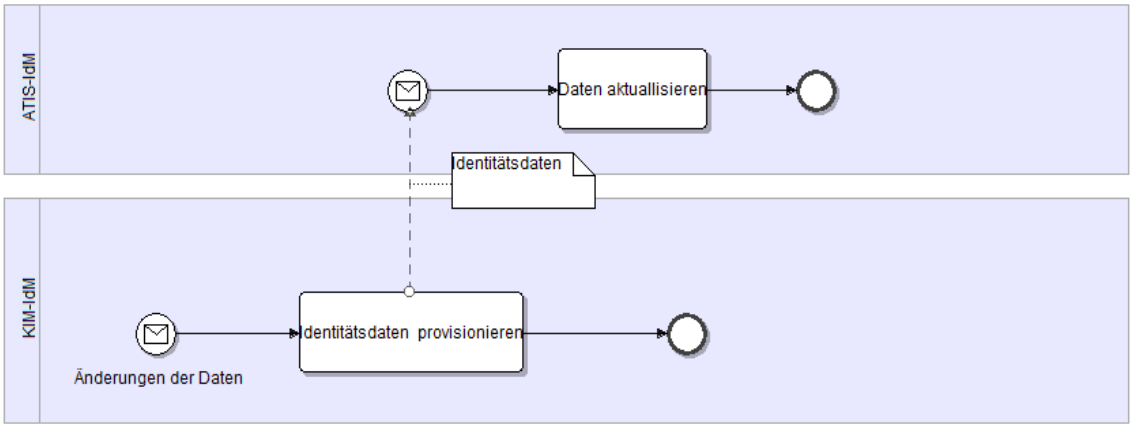


Abbildung 15 : Geschäftsprozessdiagramm "Daten aktualisieren"

Das Diagramm (Abbildung 15) zeigt, wie der Geschäftsprozess „Daten aktualisieren“ abläuft. Der Prozess wird gestartet, wenn eine Änderung von Personendaten einer Identität im KIM-IdM geschieht. Zeitnah werden entsprechend die Daten an ATIS-IDM provisioniert. Anhand dieser aktuellen Daten werden die Identitätsdaten aktualisiert.

3.3 Anforderungen

Die Anforderungsermittlung hat als Ziel, die Eigenschaften, die die Anwendung haben sollte, zu bestimmen. In der Softwaretechnik darf man sich nicht auf den intuitiven Eindruck verlassen, was gebaut werden sollte, sondern sollte die Anforderungen systematisch ermitteln.

Eine Anforderung ist eine Bedingung oder Leistungsfähigkeit, die von einem System oder einer Systemkomponente getroffen oder ausgeführt werden muss, um einem Vertrag, Standard, Spezifikation, oder einem anderen formal erhobenen Dokument zu genügen. Alle Anforderungen formen die Basis für nachfolgende Entwicklungen am System oder an der Systemkomponente. Die Anforderungen unterteilen sich in vier Typen: Funktionale Anforderungen, Nichtfunktionale Anforderungen, Sicherheitsanforderungen und Nebenbedingungen[Ba100]. In dieser Arbeit werden jedoch nur die funktionalen Anforderungen betrachtet.

Die Kopplung der beiden Systeme KIM-IdM und ATIS-IdM soll soll die beiden oben analysierten Anwendungsfällen eine große Unterstützung werden. Erstens soll die Kopplung, aufgrund einer entsprechenden Anfrage von ATIS-IdM an KIM-IdM, die Übermittlung von Identitätsdaten an ATIS-IDM ermöglichen. (Nutzung des „ersten Teils“ des Provisionierungsdienstes). Diese einmal übermittelten Daten (eine hergestellte „Provisionierungsbeziehung“) sollen anschließend im Rahmen des Provisionierungsdienstes auch fortlaufend durch KIM-IdM aktualisiert werden. Die Anwendung, die die Kopplung realisiert, soll die folgenden Anforderungrn genügen:

Anforderung 1: Abfragen von Identitätsdaten von ATIS-IdM aus KIM-IdM.

Anforderung 2: Für die Daten-Anfrage wird ein Attribut benötigt, das bei jeder Identität eindeutig ist. Dieses Attribut ist eine Kennung (Identifikator), das einen Studenten bzw. seine Identität eindeutig identifizierbar macht. Die Fricardnummer (Studentenausweisnummer) eines Studenten erfüllt genau diese Kriterien und wird deshalb verwendet.

Anforderung 3: Daten-Provisionierung von KIM-IdM aus in Richtung ATIS.

Anforderung 4: KIM-IdM provisioniert an ATIS-IdM einer Menge von Attributdaten, aus denen eine neue ATIS-Identität generiert werden kann. Aus Datenschutzgründen sollte diese Menge möglichst minimal sein und nur aus den notwendigsten Attributen bestehen. Diese Attribute sind:

- Name
- Vorname
- Immatrikulationsstatus
- Studienfach
- Information, ob Student gesperrt ist oder aktiv studiert

Name und Vorname werden gebraucht, um anhand der Fricard, die zum Zeitpunkt des Einrichtens ja vorliegt, zu verifizieren, ob die gelieferten Daten die richtigen sind. Außerdem kann somit ein Studierender auch persönlich, beispielsweise in einer Mail, angesprochen werden Der Immatrikulationsstatus und das Studienfach bestimmen das Recht auf die Benutzung des Pools bzw. auf einen Pool-Account, denn nur immatrikulierte Informatikstudenten haben das Recht auf einen Account in der ATIS. Es muss auch feststellbar sein, ob ein Student noch aktiv studiert oder beurlaubt ist. Das soll die Information, ob er (verwaltungstechnisch betrachtet) gesperrt ist, liefern, um Missbrauch seines Accounts während seiner Abwesenheit zu verhindern.

Anforderung 5: Die provisionierten Daten sollen fortlaufend und zeitnah synchronisiert werden.

4 ENTWURF

Wie schon in der Analyse erläutert wurde, haben sich aus dieser Phase Anforderungen herauskristallisiert, die die Erwartungen an der neuen Erweiterung des ATIS-IdMs beschreiben. In diesem Kapitel wird von diesen Anforderungen ausgehend ein Modell erstellt, das sie erfüllt. Später wird dann das Modell implementiert, welches im nächsten Kapitel behandelt wird.

Das Kapitel beginnt mit der statischen Sicht des Entwurfs. Dabei wird die vereinfachte Gesamtarchitektur der beiden gekoppelten Systeme beschrieben. Dazu werden die Bestandteile und die Struktur des Puffer-LDAP-Schemas definiert. Dieses Puffer-System dient zur indirekten Lieferung der ATIS-IdM mit Daten von KIM-IdM. Im zweiten Teil wird der Ablauf der Datenprovisionierung mittels UML-Aktivitätsdiagrammen entworfen.

4.1 Architektur

Die Darstellung der Kopplung-Architektur als UML-Komponentendiagramm (Abbildung 16) zeigt das ATIS-IdM und das KIM-IdM als zwei unabhängige, aber miteinander kommunizierende Systeme. Das ATIS-IdM besteht aus der Management-Applikation, dem IdM-LDAP als Datenhaltungssystem und dem Puffer-LDAP. Dieser Puffer wird für die Lieferung der Daten benötigt, damit SUN-IdM, die Management-Appliktion von KIM-IdM, nicht direkt in den sich im Produktivbetrieb befindenden LDAP schreibt. Ferner trägt er dazu bei, der Autarkie des ATIS-IdM als Satellit im gesamten Universitätverbund zu erhalten.

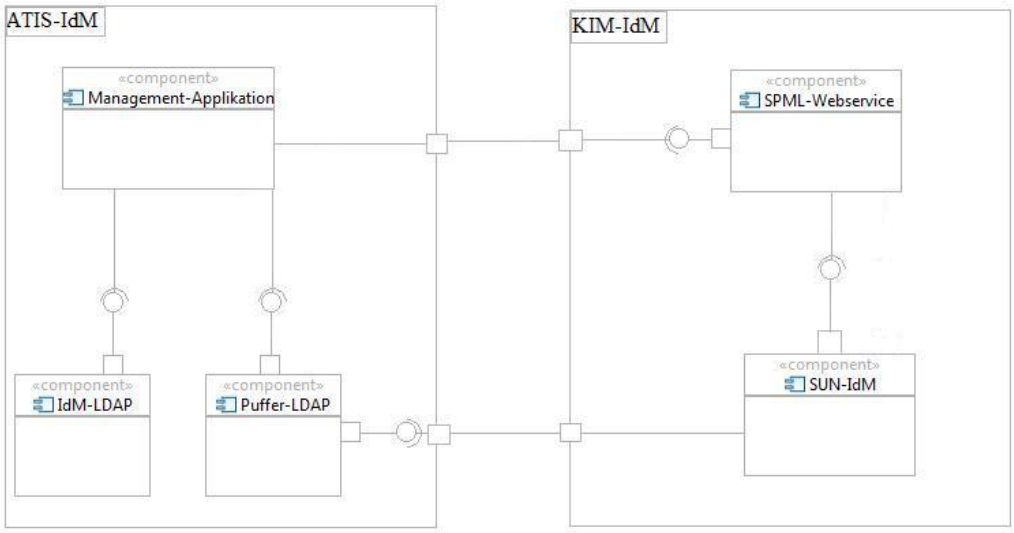


Abbildung 16 : Komponentendiagramm "Gesamtarchitektur"

Das ATIS-IdM spricht das KIM-IdM durch seine Webservice-Schnittstelle per Anfrage an. Anschließend provisioniert KIM-IdM in das Puffer-System. Bei der Realisierung des Puffersystems muss beachtet werden, dass die Datenhaltung des ATIS-IdM-Systems auf OpenLDAP basiert. Und eine eventuelle Verwendung einer anderen Technologie, z.B. eine relationale Datenbank, benötigt einen zusätzlichen Aufwand, um die Daten auf LDAP abzubilden.

Wie schon im Einführungskapitel beschrieben wurde, ist KIM-IdM ein recht komplexes System. Jedoch wird hier die Betrachtung und Beschreibung dieses Systems nur auf der Architektur im Bild (Abbildung 16) eingeschränkt, weil nur die abgebildeten Komponenten für die Realisierung der Schnittstelle relevant sind. Außerdem würde eine komplette Beschreibung des Systems den Rahmen dieser Arbeit sprengen.

Die ATIS braucht die Anbindung an den KIM-IdM-Verbund, um neue Identitätsdaten zu holen und bestehende Identitätsdaten zu aktualisieren. Deshalb wird die ATIS genau das Provisionierungsdienst von KIM-IdM in Anspruch nehmen, auf dessen vereinfachte Architektur hier eingegangen wird. Der Dienst besteht aus einem Webservice, der Anfragen entgegennimmt. Die Anfragen enthalten einen Identifikator zu dem eine bestimmte eindeutige Identität gesucht und ggfs. geliefert wird. Dafür leitet der Webservice die Anfrage an den SUN-IdM weiter. In SUN-IdM sind Daten von allen immatrikulierten Studenten der Universität gespeichert. SUN-IdM liefert dann die Identitätsdaten, wenn sie vorhanden sind.

ATIS-IdM ist ein im Aufbau befindliches System zur Verwaltung von Identitäten und Accounts in der Fakultät für Informatik. Das System besteht ursprünglich aus einer Management-Applikation, die als eine Webapplikation implementiert wird, und einem LDAP-Server, der als Datenhaltungssystem verwendet wird.

4.2 Puffersystem

Damit das IdM-System den Provisionierungsdienst in Anspruch nehmen kann, benötigt es ein Puffersystem, das als Schnittstelle zum SUN-IdM von KIM dient. In diesem Puffer werden die nachgefragten Daten unmittelbar von SUN-IdM geschrieben und im Anschluss von ATIS-IdM-Applikation gelesen und ergänzt. Die Daten, die hier geschrieben werden, werden benutzt, um ATIS-Identitäten zu bilden. Sie enthalten nur gewisse Attribute, die später genauer erläutert werden. Die Puffer-Identität müssen erst mit anderen Attributen ergänzt werden, um ATIS-Identitäten zu bilden. Deshalb sind sie aus Sicht des ATIS-IdMs nur Teilidentitäten.

Für das Puffersystem, das durch einen LDAP-Server umgesetzt wird, muss also ein LDAP-Schema definiert werden. Hier wird nur eine einzige Objektklasse benötigt, weil nur ein einziger Objekttyp gebraucht wird, und zwar die Puffer-Identität.

Das Schema muss die in der Analyse aufgeforderten Attribute enthalten, und zwar:

1. Name
2. Vorname
3. Immatrikulationsstatus
4. Studienfach
5. Information, ob Student gesperrt

Dazu kommen noch technisch bedingte Attribute. Die Fricadnummer (Studentenausweisnummer) wird als gemeinsamer Schlüssel (Identifikator) von ATIS und KIM benutzt, um die LDAP-Einträge zu identifizieren. Deshalb muss sie unbedingt vorhanden sein. Die KIM-GUID (*Globally Unique Identifier*) ist auch ein Identifikator, der von KIM-IdM verwendet wird, um die Puffer-Identitäten auf die eigenen Identitäten

abzubilden. Die ATIS-Id ist ebenso ein Identifikator, der von ATIS-IdM verwendet wird, um die Puffer-Identitäten auf die eigenen Identitäten abzubilden.

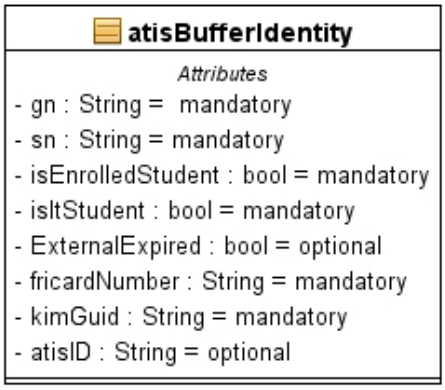


Abbildung 17 : Klassendiagramm "atisBufferIdentity"

Die Puffer-Identität-Objektklasse wird *„atisBufferIdentity„* genannt. Sie enthält die Attribute, die zuvor erwähnt wurden. Die Attribute werden wie folgt mit dem entsprechenden Variablentyp definiert (siehe Abbildung 17).

- Notwendiges Attribut: *String cn* (Name)
- Notwendiges Attribut: *String sn* (Vorname)
- Notwendiges Attribut: *String fricardNumber* (Fricardnummer)
- Notwendiges Attribut: *Boolean isEnrolledStudent* (Information, ob der Student immatrikuliert ist)
- Notwendiger Attribut: *Boolean isItStudent* (Information, ob der Student Informatik studiert)
- Notwendiges Attribut: *String kimGuid* (KIM-GUID)
- Optionales Attribut: *String atisId* (ATIS-Id). Dieses Attribut ist optional, weil KIM-IdM es selbst schreiben kann.
- Optionales Attribut: *Boolean ExternalExpired* (Information, ob Student gesperrt) . Das Attribut wird nur selten benutzt.

Die Objektklasse wird als ein *UML-Klassediagramm* dargestellt um sie mit ihren Attributen zu veranschaulichen.

4.3 Webservice-Anfrage

Um auf den angebotenen Provisionierungswebservice zugreifen zu können, wird in der ATIS-IdM-Applikation ein entsprechender Webservice-Client erstellt. Mit diesem Client kann auf die Webservice-Funktionen fern (*remote*) zugegriffen werden.

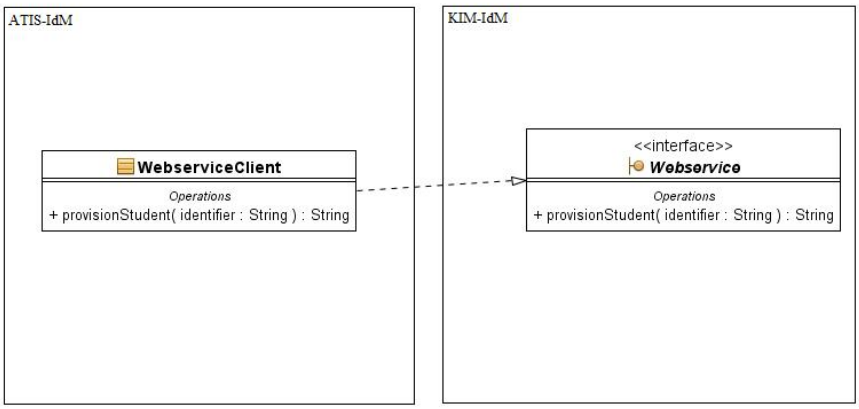


Abbildung 18 : Klassendiagramm "Webservice-Client"

Das Klassendiagramm (Abbildung 18) zeigt den Webservice-Client als eine implementierende Klasse des Webservice, der als eine Schnittstelle dargestellt wird. Die beiden Komponenten befinden sich in unterschiedlichen Systemen. Die Schnittstelle *Webservice* ist ein Teil des KIM-IdMs und die Klasse *Webservice-Client* ist ein Teil des ATIS-IdMs.

Sobald der Webservice-Client in der ATIS-Managemnet-Applikation integriert wird, kann die Anfrage erfolgen. Die Grafik (Abbildung 19) beschreibt den Ablauf einer Webservice-Anfrage mittels eines UML-Aktivitätsdiagramms.

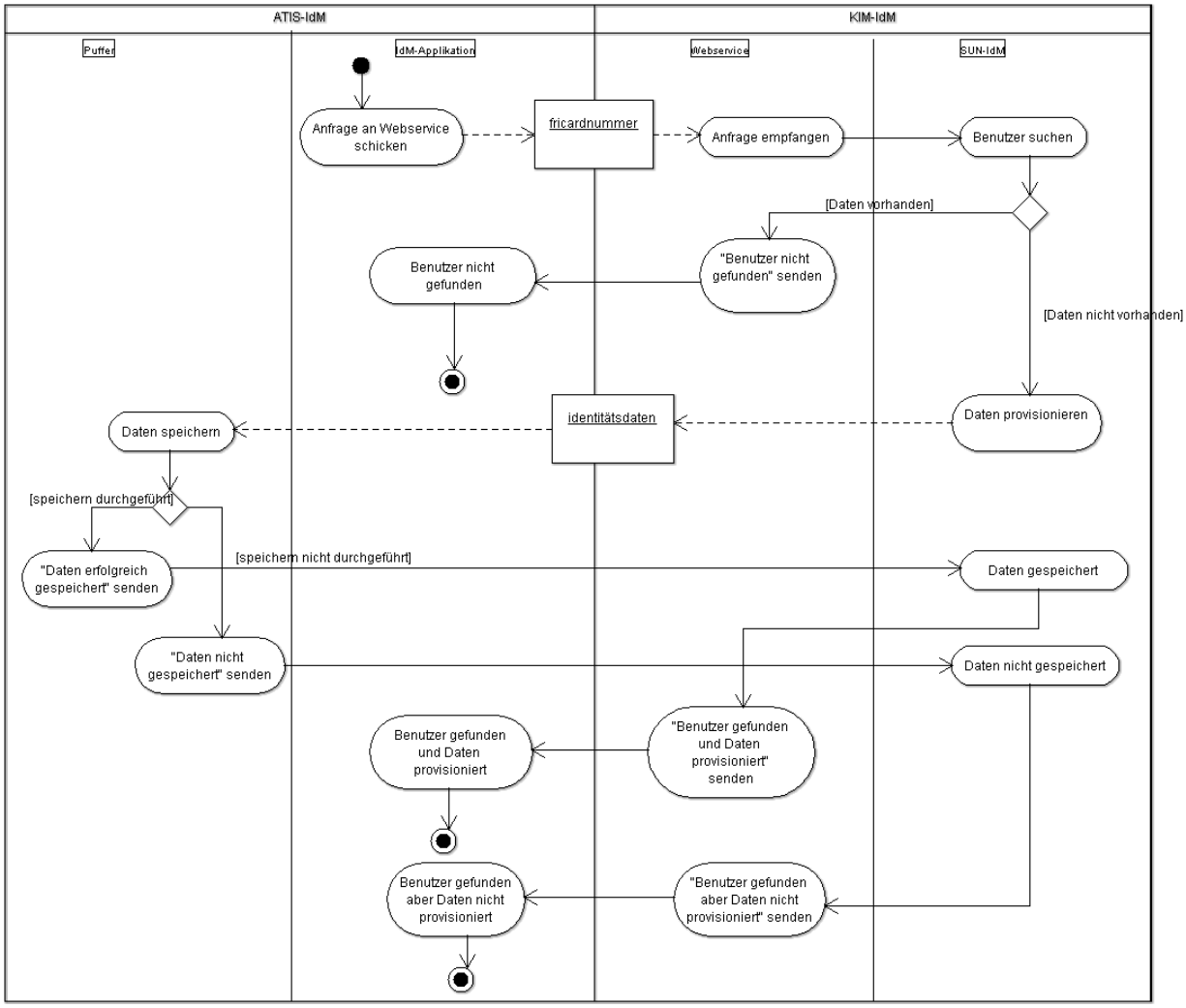


Abbildung 19 : Aktivitätsdiagramm "Webservice-Anfrage"

Um die Anfrage zu starten, wird die Fricardnummer des gesuchten Studenten eingegeben. Eine implementierte Funktion des Webservice wird dann mit dieser Nummer als Eingabewert aufgerufen. Diese Methode führt dann die eigentliche Anfrage, indem sie eine Nachricht, die die Fricardnummer beinhaltet, an den Webservice schickt. Wenn der Webservice die Anfrage erhält, stößt er den Suchvorgang von KIM-IdM an. Wenn der Benutzer nicht gefunden werden kann, liefert der Webservice eine entsprechende Nachricht an die ATIS-IdM-Applikation. Andernfalls wird es versucht, die entsprechenden Daten der Benutzeridentität in den Puffer zu provisionieren.

Wenn die Daten erfolgreich gespeichert worden sind, liefert der Webservice die Nachricht „*Benutzer gefunden und Daten erfolgreich gespeichert*“. Wenn aber die Provisionierung nicht durchgeführt werden kann, lautet die Rückgabe des Webservices „*Benutzer gefunden aber, Daten nicht gespeichert*“.

4.4 Prozess „Account anlegen“

Wenn die Daten einmal im Puffesystem gelandet sind, kann in der Management-Applikation mit dem Anlegen eines Accounts bzw. einer Identität angefangen werden.

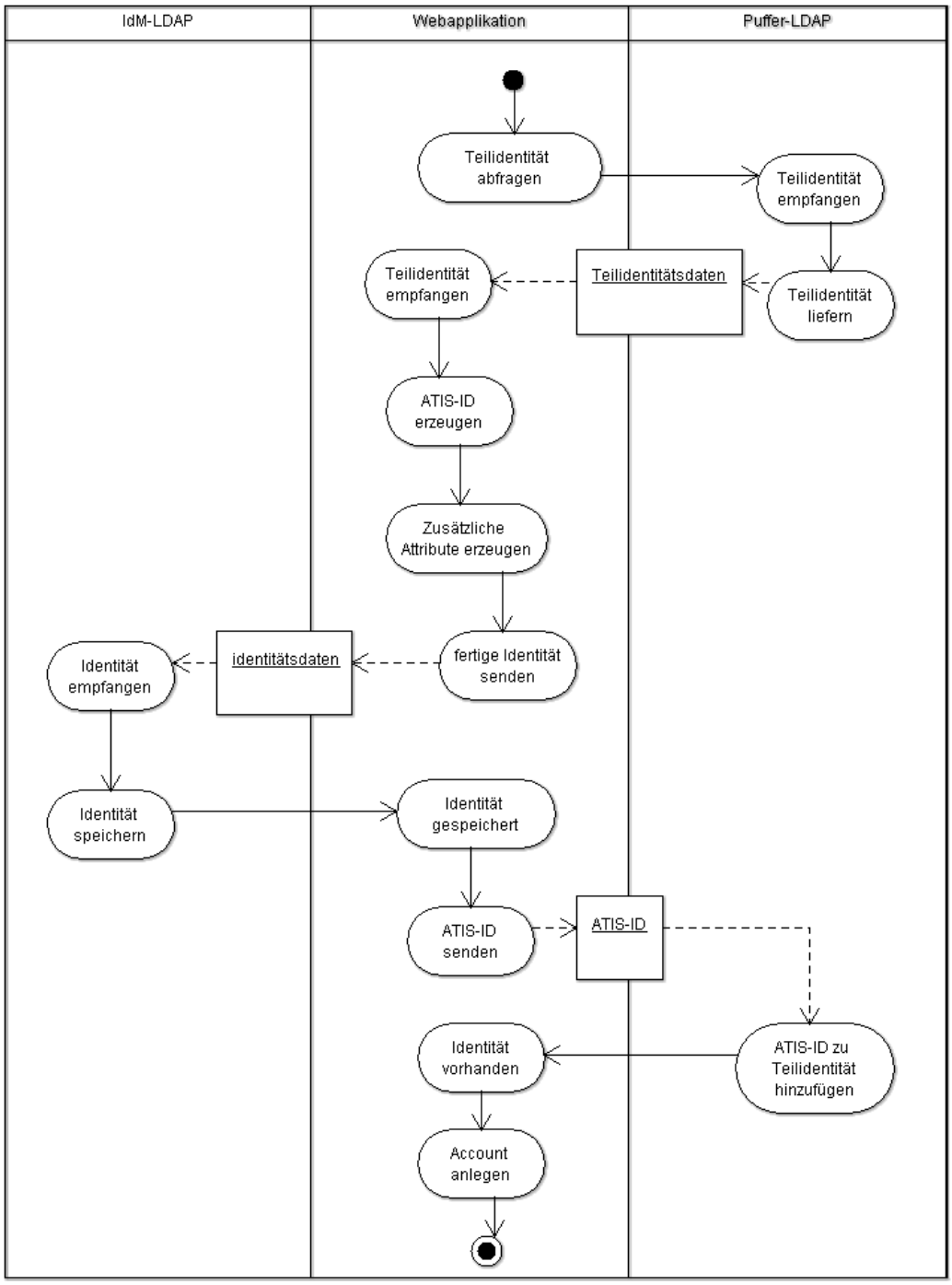


Abbildung 20 : Aktivitätsdiagramm "Account anlegen"

Als erstes wird die Puffer-Identität bzw. die Teilidentität vom Puffer geholt. Um daraus eine „ganze“ Identität bzw. eine ATIS-Identität zu bilden, werden andere technische Attribute erzeugt und hinzugefügt. Eines dieser Attribute ist die ATIS-ID. Dieser Identifikator wird auch der Puffer-Identität hinzugefügt. Die fertige ATIS-Identität wird schließlich im ATIS-

IdM-LDAP gespeichert. Wenn diese Schritte abgeschlossen sind, liegt eine Identität vor, zu der ein neuer Account erstellt werden kann, was auch direkt im Anschluss geschehen soll (siehe Abbildung 20).

5 IMPLEMENTIERUNG

Die Implementierung der entwickelten Anwendung wird in dieser Arbeit nur prototypisch und teilweise erfolgen, weil eine komplette Implementierung den Rahmen einer Studienarbeit sprengen würde. Der erste Abschnitt ist der Installation und Konfiguration des Puffer-LDAP gewidmet. Der zweite Abschnitt zeigt, wie die Provisionierungs-Webservice-Schnittstelle beschrieben wird, und danach, wie auf den Webservice anhand eines Webservice-Clients zugegriffen wird. Am Schluss wird ein Test durchgeführt, um die Anwendung zu testen und zu bewerten.

5.1 Puffer-LDAP

Das Puffersystem soll als LDAP-Server implementiert werden. Für diesen Zweck ist OpenLDAP genau richtig [OpenLDAP]. OpenLDAP ist die freie Open-Source Implementierung des LDAP, wie der Name schon andeutet. Das Open-Source-Projekt ging vor einigen Jahren aus einem Server-Projekt der Universität Michigan hervor.

5.1.1 Installation

OpenLDAP besteht aus einem skalierbaren Server mit passenden LDAP-Clients und unterstützt seit Version 2 auch die Protokolle der LDAP-Version 3 [RFC2251]. Das Produktpaket OpenLdap besteht aus mehreren Teilen:

1. *slapd* : *stand-alone LDAP daemon*
2. *slurpd* : *stand-alone LDAP update replication daemon*
3. *overlays* : ermöglichen *extended Operations*
4. *syncrep*l : Synchronisation und Replikation
5. Bibliotheken, die das LDAP-Protokoll bereitstellen
6. Werkzeuge, Hilfsmittel und Beispiele

Um OpenLDAP installieren zu können, braucht man zuerst einen Rechner mit einem laufenden Linux-Betriebsystem. Der Rechner wird durch eine VM-Maschine bereitgestellt, auf der dann eine CentOS-Linux installiert wird.

Die Entwickler der Linux-Distribution CentOS [CentOS] haben das Betriebsystem nun in einer Server-Variante veröffentlicht. Dabei handelt es sich um eine einzelne CD mit abgespecktem Paketumfang, der auf den Server-Einsatz zugeschnitten ist. CentOS basiert komplett auf den frei verfügbaren Quellen von Red Hat Enterprise Linux und ist kostenlos. Die Server-Edition liefert eine Auswahl an Paketen mit, die besonders für den Einsatz auf Servern sinnvoll sein sollen. Grundsätzlich enthält die Server-Variante jedoch dieselben Funktionen wie die komplette Version von CentOS 4.3. Die Installation ist einfach und schnell dank der freundlichen grafischen Benutzeroberfläche.

Um nun OpenLDAP auf dem CentOS-Linux-Server zu installieren, wird das Software-Management-Tool „yum“ eingesetzt.
Der folgende Befehl (siehe Abbildung 21) wird auf der Kommandozeile des Servers ausgeführt, der „idm-buffer“ genannt wird. Der Befehl liefert zurück, welche OpenLdap-Pakete zur Verfügung stehen.

```
[root@idm-buffer ~]# yum search openldap
Loading "fastestmirror" plugin
Loading mirror speeds from cached hostfile
 * base: ftp.plusline.de
 * updates: ftp.plusline.de
 * addons: ftp.plusline.de
 * extras: ftp.plusline.de
openldap-servers.i386 : OpenLDAP servers and related files.
openldap-devel.i386 : OpenLDAP development libraries and header files.
compat-openldap.i386 : OpenLDAP compatibility shared libraries.
openldap.i386 : The server SQL support module.
openldap-clients.i386 : Client programs for OpenLDAP.
openldap-servers-sql.i386 : OpenLDAP server SQL support module.
compat-openldap.i386 : OpenLDAP compatibility shared libraries.
openldap-clients.i386 : proconfiguration files, libraries, and documentation for OpenLDAP.
python-ldap.i386 : An object-oriented API to access LDAP directory servers.
openldap-devel.i386 : OpenLDAP development libraries and header files.
openldap-devel.i386 : OpenLDAP development libraries and header files.
openldap-servers-sql.i386 : OpenLDAP grams for OpenLDAP.
openldap.i386 : The configuration files, libraries, and documentation for OpenLDAP.
openldap-clients.i386 : Client programs for OpenLDAP.
openldap.i386 : The configuration files, libraries, and documentation for OpenLDAP.
openldap-servers.i386 : OpenLDAP servers and related files.
compat-openldap.i386 : OpenLDAP compatibility shared libraries.
openldap-servers.i386 : OpenLDAP servers and related files.

[root@idm-buffer ~]# yum install openldap-servers.i386 openldap-devel.i386 openldap.i386
```

Abbildung 21 : Installation von OpenLDAP mit "yum"

Die Pakete, die installiert werden müssen, sind *openldap-servers.i386, openldap-devel.i386* und *openldap.i386.* Dafür wird der Befehl (siehe letzte Zeile in Abbildung 21) ausgeführt.

5.1.2 Konfiguration

Bevor OpenLDAP genutzt werden kann, muss das System konfiguriert werden. Dazu dienen, wie unter Linux üblich, Textdateien, die sich im Ordner */etc/openldap* befinden. Für die Einstellungen des Servers ist die Datei *slapd.conf* verantwortlich. Am Anfang wird festgelegt, welche Schemata die nachfolgend definierten LDAP-Verzeichnisse kennen. Auf diese Weise werden die verwendbaren Datentypen vorgegeben.

```
include         /etc/openldap/schema/core.schema
include         /etc/openldap/schema/cosine.schema
include         /etc/openldap/schema/inetorgperson.schema
include         /etc/openldap/schema/nis.schema
include         /etc/openldap/schema/atis/atisBufferldap.schema
```

Codefragment 3 : "slapd.conf "-Auszug (Schemata einbinden)

Core.schema enthält Standard-Attribute, die grundsätzlich eingebunden werden. *cosine.schema* enthält wichtige Standard-Attribute der Version 3 vom LDAP. *nis.schema* definiert Attribute, die benutzt werden können, um LDAP als *Network Information Service* zu definieren.
Diese Unterkonfigurationsdateien werden mit „*include*“ eingebunden (siehe **Fehler! Verweisquelle konnte nicht gefunden werden.**). *inetorgperson.schema* hängt von *core.schema* und *cosine.schema* und beinhaltet Attribute, die für den Aufbau eines organisationsorientierten Dienstes verwendet werden können. *atisBufferldap.schema* ist ein Schema, das nicht vom OpenLDAP ist, sondern selbst definiert wurde. OpenLDAP bietet die

Möglichkeit, eigene Schemata zu definieren, die auf die eigenen Bedürfnisse zugeschnitten sind. Dieses Schema definiert alle Attribute und die Objektklasse, die gebraucht werden, um das Puffersystem zu betreiben. Im nächsten Abschnitt wird das Schema im Detail beschrieben.

Die folgenden Anweisungen (Codefragment 4) bestimmen, wo der LDAP-Server die entsprechenden Informationen über seine aktuelle Prozess-ID und die beim Aufruf verwendeten Parameter ablegt.

```
pidfile         /var/run/openldap/slapd.pid
argsfile        /var/run/openldap/slapd.args
```

Codefragment 4 : "slapd.conf "-Auszug (PID- und ARGS-File)

Über die *database*-Anweisung (Code 5) legen die von OpenLDAP zu verwendende Datenbank fest. Die Angabe von „*bdb*“ bewirkt die Nutzung der *Berkeley Database*, dem von den OpenLDAP-Entwicklern empfohlenen System.

```
database        bdb
```

Codefragment 5 : "slapd.conf "-Auszug (Datenbank-Definition)

Über das Schlüsselwort *suffix* (Codefragment 6) teilt man OpenLDAP mit, welche Wurzel es für Objekte verwenden soll, die nicht als vollständiger DN (*Distinguished Name*) angegeben wurden.

```
suffix          "dc=uka,dc=de„
```

Codefragment 6 : "slapd.conf "-Auszug (Datenbank-Suffix)

Der hinter *rootdn* angegebene Wert bezeichnet den Administrations-Account zur Verwaltung der LDAP-Verzeichnisse. Das Schlüsselwort *rootpw* legt das zum Administrations-Account gehörende Passwort fest (siehe Codefragment 7).

```
rootdn          "cn=Manager,dc=uka,dc=de„
rootpw          {crypt}ijFYNcSNctBYg
```

Codefragment 7 : "slapd.conf "-Auszug (Datenbank-Manager)

Im folgenden (Codefragment 8) wird das Verzeichnis angegeben, in dem OpenLDAP die diversen Datenbanken ablegt. Das Verzeichnis muss existieren, bevor der LDAP-Server gestartet wird, da dieser das Directory nicht selbst anlegt.

```
directory       /var/lib/ldap
```

Codefragment 8 : "slapd.conf "-Auszug (Datenbank-Verzeichnis)

5.1.3 LDAP-Schema

Wie man bereits in der Konfigurationsdatei slapd.conf sehen kann, werden mehrere Konfigurationsdateien eingebunden. Diese Schemadateien sollten nicht geändert werden. Das Schema enthält die Objektklassen und die Attribute für die Objektklassen. Ein Schemata-Element wird über einen OID, einen *Object Identifier*, eindeutig bestimmt. Das ist im

Wesentlichen eine Ziffernfolge, die man in etwa mit der Kapitel-Gliederung eines sehr großen Buches vergleichen kann: Es gibt 1.1 und 1.2, unterhalb von 1.1 dann 1.1.1 und 1.1.2. Es kann dann auch 1.1.2.1.19.241.243.4 geben und so weiter. Über OIDs kann man eigentlich allem Möglichen eine eindeutige Nummer geben. Das sind nicht nur Objektklassen, sondern auch Datentypen und Syntaxregeln. Die Syntax eines DNs ist zum Beispiel im OID 1.3.6.1.4.1.1466.115.121.1.12 definiert, ein "directoryString" (das ist eine Zeichenkette im UTF-8 Zeichensatz) ist definiert als 1.3.6.1.4.1.1466.115.121.1.15.

Die Universität Karlsruhe bzw. die Fakultät für Informatik und die ATIS besitzen eigene OIDs. „unikaOID:4“ z.B. bedeutet „1.3.6.1.4.1.87.4“, d.h. „unikaOID“ wird durch seine OID-Nummer ersetzt und dahinter eine 4 angehängt. Die Folgenden OID-Definitionen bilden den Anfang der Schemata-Datei (Code 9).

```
# OID prefix for University of Karlsruhe:
objectIdentifier unikaOID 1.3.6.1.4.1.87
# OID prefix for Abteilung technische Infrastruktur (ATIS):
objectIdentifier atisOID unikaOID:4
# ATIS LDAP definitions
objectIdentifier atisLdapOids atisOID:2
# ATIS LDAP attribute type definitions
objectIdentifier atisLdapAttrs atisLdapOids:1
# ATIS LDAP object class definitions
objectIdentifier atisLdapClasses atisLdapOids:2
```

Codefragment 9 : : "atisBufferIdentity.schema"-Auszug (OIDs)

Attribute

Damit Attribute korrekt definiert werden, wird die LDAP-Spezifikation verwendet. Die LDAP-Spezifikation liegt in Form von RFCs (*Request for Comment*) vor. Die Definition eines Attributtyps von LDAPv3 in [RFC2252] sieht folgendermaßen (Code 10) aus:

```
AttributeTypeDescription = "(" whsp      # whitespace (Leerzeichen)
numericoid whsp                          # AttributeType identifier OID
[ "NAME" qdescrs ]                       # menschenlesbare Name des Attributs
[ "DESC" qdstring ]                      # menschenlesbare Beschreibung
                                         #des Attributs
[ "OBSOLETE" whsp ]                      # Indikator, ob das Attribut obsolet ist
[ "SUP" woid ]                           # enthält OID von dem das
                                         #Attribut erben soll

[ "EQUALITY" woid                        # Art der Gleichheitstets
[ "ORDERING" woid                        # Anzuwendende Vergleichsregel
[ "SUBSTR" woid ]                        #Vergleichsregel für Teilstrings
[ "SYNTAX" whsp noidlen whsp ]           #OID der verwendeten Syntax
```

Codefargment 10 : Attribut-Definition im LDAPv3

Eine der wichtigsten Angaben bei der Definition eines Attributs ist die Syntax. Die folgende Tabelle (Tabelle 5) zeigt einen Auszug der für die Entwicklung dieser Anwendung relevanten Syntaxdefinitionen, die mittels ihrer OIDs angegeben werden.

Syntax	OID	Beschreibung
directoryString	1.3.6.1.4.1.1466.115.121.1.15	UTF-8 String (Unicode)
Boolean	1.3.6.1.4.1.1466.115.121.1.7	Boolscher Wert
Generalized Time	1.3.6.1.4.1.1466.115.121.1.24	Zeitangabe in LDAP
Integer	1.3.6.1.4.1.1466.115.121.1.27	Ganze Zahl

Tabelle 5 : Syntaxdefinitionen von Attributen

Die zweite Tabelle (Tabelle 6) listet die relevanten Vergleichsregeln. Der Typ *EQUALITY* wird angegeben, wenn ganze Strings miteinander verglichen werden sollen. Wenn aber nur die Teilstrings miteinander verglichen werden sollen, dann wird der Typ *SUBSTRING* verwendet. Diese Vergleiche werden bei der Suche nach Einträgen im LDAP-Verzeichnis zwischen die gespeicherten Daten und angegebenen Daten benuzt. *ORDERING* definiert, welche Vergleichsregel angewendet werden soll, wenn die Strings angeordnet werden sollen.

Name der Regel	Typ	Art des Vergleichs
caseIgnoreMatch	EQUALITY	case sensitive, space sensitive
integerMatch	EQUALITY	Integer
booleanMatch	EQUALITY	boolean
generalizedTimeMatch	EQUALITY	Generalized Time
caseIgnoreSubstringsMatch	SUBSTRING	case sensitive, space sensitive
caseIgnoreOrderingMatch	ORDERING	case sensitive, space sensitive
integerOrderingMatch	ORDERING	case sensitive, space sensitive
generalizedTimeOrderingMatch	ORDERING	Generalized Time

Tabelle 6 : Vergleichsregeln von Attributen

Anhand der Informationen der LDAP-Spezifikation können nun die Puffersystem-Attribute definiert werden.

Die aufgeführten Attribute (Code 11) sind die, die im Entwurf entstanden sind. Dazu kommen zwei zusätzliche technische Attribute „atisModifyTimestamp“ und „atisCreationTimestamp“. Die beiden geben an wann ein Eintrag im Verzeichnis erstellt und wann wurde es zum letzten Mal modifiziert wurde.

```
attributetype ( atisLdapAttrs:14 NAME ( 'atisId' 'atisIdentifier' )
        EQUALITY caseIgnoreMatch
        SUBSTR caseIgnoreSubstringsMatch
        ORDERING caseIgnoreOrderingMatch
        DESC 'Unique key identifying a person within the Fakultät für Informatik
              - University of Karlsruhe.'
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.15
        SINGLE-VALUE )

attributetype ( atisLdapAttrs:50 NAME ( 'kimGuid' 'KIM-GloballyUniqueIdentifier' )
        EQUALITY caseIgnoreMatch
        SUBSTR caseIgnoreSubstringsMatch
        ORDERING caseIgnoreOrderingMatch
        DESC 'Unique key identifying a person within the KIM-IDM  - University of Karlsruhe.'
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.15
        SINGLE-VALUE )

attributetype ( atisLdapAttrs:51 NAME 'atisFricardNumber'
        DESC 'Student card number assigned by the University of Karlsruhe'
        EQUALITY integerMatch
        ORDERING integerOrderingMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.27
        SINGLE-VALUE )

attributetype ( atisLdapAttrs:27 NAME 'atisExternalExpired'
        DESC 'TRUE if this identity is no longer valid in the external identity management.'
        EQUALITY booleanMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.7
        SINGLE-VALUE )

attributetype ( atisLdapAttrs:28 NAME 'atisEnrolledStudent'
        DESC 'TRUE if this student is enrolled.'
        EQUALITY booleanMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.7
        SINGLE-VALUE )

attributetype ( atisLdapAttrs:29 NAME 'atisItStudent'
        DESC 'TRUE if this student is enrolled at the faculty of information technology.'
        EQUALITY booleanMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.7
        SINGLE-VALUE )

attributetype ( atisLdapAttrs:17 NAME 'atisModifyTimestamp'
        DESC 'Timestamp of last modification by a person'
        EQUALITY generalizedTimeMatch
        ORDERING generalizedTimeOrderingMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.24
        SINGLE-VALUE )

attributetype ( atisLdapAttrs:19 NAME 'atisCreationTimestamp'
        DESC 'Timestamp of creation by a person'
        EQUALITY generalizedTimeMatch
        ORDERING generalizedTimeOrderingMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.24
        SINGLE-VALUE )
```

Codefragment 11 : Attribut-Definition in "atisBufferIdentity.schema"

Die *objectclasse*-Direktive dient dem Definieren einer neuen Objektklasse. Genauso wie bei den Attributen ist die Spezifikation einer Objektklasse im [RFC4512] definiert (siehe Code 12).

```
ObjectClassDescription = "(" whsp                 # whitespace (Leerzeichen)
                     numericoid whsp              # object class identifier OID
[ "NAME" qdescrs ]                                # Name der Objektklasse
[ "DESC" qdstring ]                               # Beschreibung der Objektklasse
          #Attributs
[ "OBSOLETE" whsp ]                               # Indikator, ob die Objektklasse obsolet ist
[ "SUP" woid ]                                    # enthält OID von dem die
                                                  #Klasse erben soll

[ ( „ABSTRACT“ / „STRUCTURAL“ / „AUXILIARY“) whsp]   # Objektklasentyp :
                                                   #(default strucural)

[ „MAY“ oids ]                                     # erforderliche Attribute
[„MUST“ oids]                                      # optionale Attribute

          whsp ")"
```

Codefragment 12 : Objektklassedefinition in LDAPv3

und entsprechend ist die Puffersystem-Objektklasse aufgebaut (Code 13).

```
objectclass (  atisLdapClasses:30 NAME 'atisBufferIdentity'
        SUP inetOrgPerson
        STRUCTURAL
        DESC 'Identity of a person.'
        MUST (  cn $ sn   $ kimGuid $ atisFricardNumber $ atisEnrolledStudent $ atisItStudent )
        MAY ($ atisExternalExpired $ atisId) )
```

Codefragment 13 : "atisBufferIdentity"-Objektklasse

Die Objektklasse atisBufferIdentity (Code 13) erbt von der Klasse *inetOrgPerson*. Sie ist *STRUCTURAL*, d.h. sie definiert elementare Attribute eines Objektes. *AUXILIARY* und *ABSTRCT* Objektklassen können selbst keine Objekte definieren, sondern nur durch eine *STRUCTURAL* Objektklasse. *atisBufferIdentity* erbt von der Klasse *inetOrgPerson* und zwar die Attribute *cn* (*common name*) für den Nachnamen und *sn* (*surname*) für den Vornamen. Die Objektklasse hat die notwendigen Attribute: *cn*, *sn*, *kimGuid*, *atisFricardNumber*, *atisEnrolledStudent* und *atisItStudent*. Die optionalen Attribute der Objektklasse sind: *atisExternalExpired* und *atisId*. *atisExternalExpired* ist optional, weil er nur selten benutzt wird. Und *atisId* ist optional, weil die KIM-IdM über ihn nicht verfügt, was zu einem Fehler führt, wenn KIM-IdM Daten in den Puffer-LDAP bei der Provisionierung zu schreiben versucht und das Attribut notwendig wäre.

5.2 Webservice-Anfrage

5.2.1 Webservice-Client

Die Grafik (Abbildung 22) zeigt in Form von einem UML-Klassendigramm wie die Datei „*ATISSPMLWebservice.wsdl*“ aufgebaut ist. Diese Datei beschreibt den Webservice „*ATISSPMLWebservice*“, der den Provisionierungsdienst bereitstellt. Die Verwendung von einem UML-Klassendiagramm soll hier die Struktur und die Zusammenhänge der verschiedenen WSDL-Elemente besser veranschaulichen.

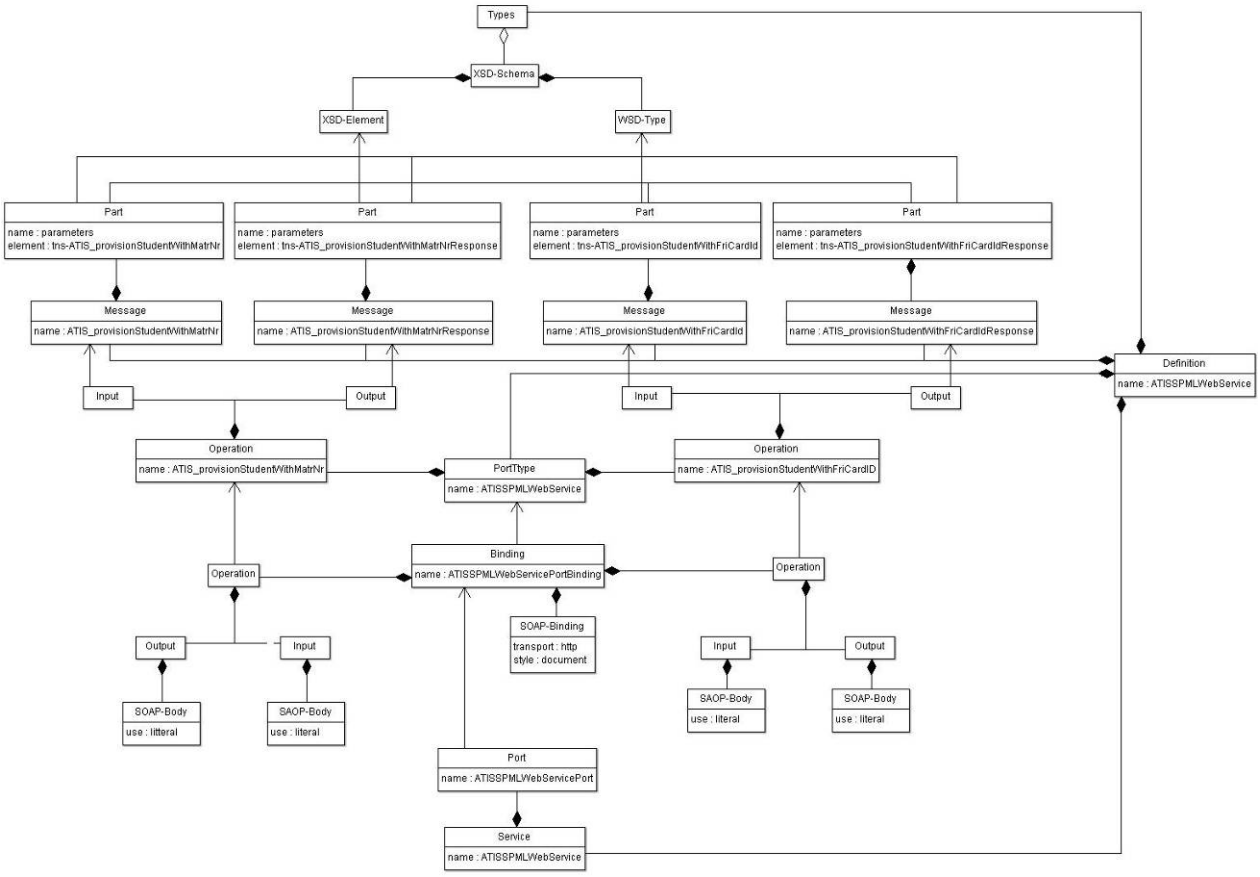


Abbildung 22 : "ATISSPMLWebservice.wsdl" als Klassendiagramm

Wie im Diagramm (Abbildung 22) abgelesen werden kann, verfügt der Webservice über zwei Operationen „*ATISSPMLprovisionStudentrWithMatrNr*“ und „*ATISSPMLprovision StudenWithFriCardId*“. Wie die Namen schon deuten, dienen die Operationen zur Provisionierung von Identitätsdaten von Studierenden entweder durch ihre Matrikelnummer oder ihre Fricardnummer (Studentenausweisnummer).

Die Operationen werden im abstrakten Teil der WSDL-Datei im *PortType*-Abschnitt definiert. Sie verfügen beide über jeweils einen Input und einen Output, die auf die *Messages* refenrenzieren. Die Operation „*ATISSPMLprovisionStudentWithMatrNr*“ hat als Input die Message „*ATISSPMLprovisionStudentrWithMatrNr*“ und als Output „*ATISSPMLprovision StudentrWithMatrNrResponse*“. Die Operation „*ATISSPMLprovisionStudentrWithFriCard*“ hat als Input die *Message* „*ATISSPMLprovisionStudentrWithFricardId*“ und als Output „*ATISSPMLprovisionStudentrWithFricardIdResponse*“.

Die *Messages* bestehen aus Nachrichtenteilen (*Parts*). Diese *Parts* referenzieren Elemente aus der Schemadefinition durch Angabe eines Präfixes für den Namensraums. Die Schemadefinition wird in *Types* mittels einer URL angegeben. Die URL ist gibt die Adresse an, wo sich das verwendete XML-Schema befindet.

Die Operationen bekommen als Eingabeparameter eine Nachricht. Diese Nachricht enthält eine Kennung, die eine bestimmte Identität identifiziert (Matrikelnummer oder Fricardnummer) in Form einer Zeichenkette (*String*), wie es im Schema angegeben ist. Und

als Ausgabe versenden die Operationen eine Nachricht in Form einer Zeichenkette (String), die eine von den drei folgenden Informationen enthält:

1. „*ERROR: User found but Provisioning was not successful*.”
2. “*ERROR: User not found*.”
3. “*SUCCESS* “, wenn die Provisionierung erfolgreich durchgeführt wurde.

Im konkreten Teil der WSDL-Datei werden die Operation vom *PortType* an konkrete Technologien gebunden. Das Binding hat ein *Soap-Binding*-Element, das HTTP als Übertragungsprotokoll und SOAP als Nachrichtenformat angibt. Die zwei Operationen im *Binding* referenzieren auf die vom *PortType*. Die Operationen haben ebenso jeweils einen Input und einen Output. Die Output und Input-Elemente haben einen *SAOP-Body*, wo „*literal*“ als Parameter benutzt wird. Der Port „*ATISSPMLWebServicePort*“ ist die Außenschnittstelle des Webservice und hat den konkreten Endpunkt, unter dem der Service aufgerufen werden kann. Der Endpunkt wird als URL angegeben.

Die Entwicklung der Webservice-Anfrage wird in der Netbeans-Entwicklungsumgebung erfolgen. Mithilfe der Entwicklungsumgebung ist es recht einfach, einen Webservice-Client zu erstellen.

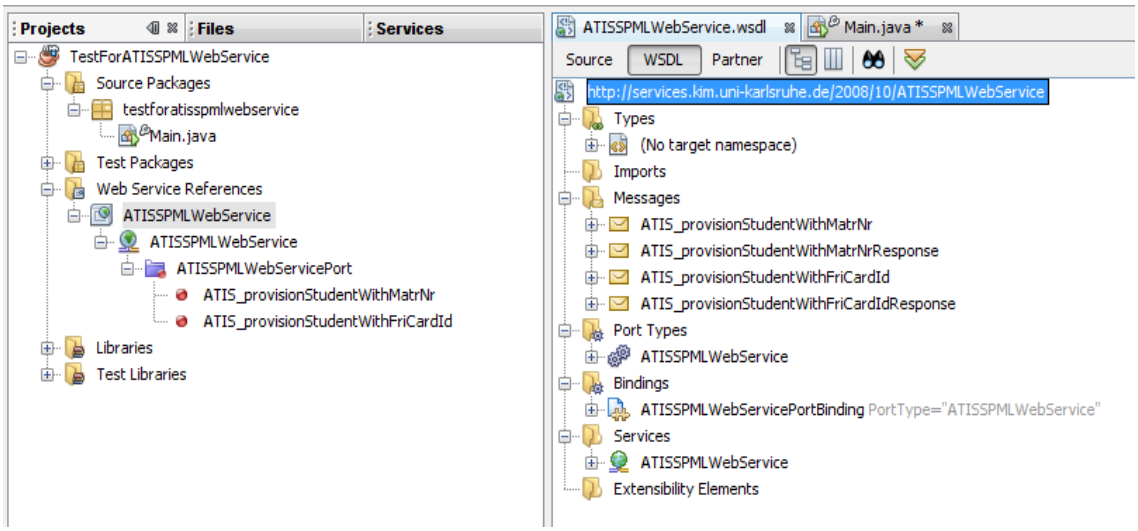


Abbildung 23 : Webservice-Client in Netbeans

Dies erfolgt, indem man die URL des Webservice, das durch eine WSDL-Datei beschrieben wird, angibt. Wenn man nun zum Knoten "*Web Service References*" im „*Projects*“-Fenster wechselt (siehe Abbildung 23), stellt man fest, dass der Webservice sowie seine Operation im Objektbaum verfügbar sind. Per Rechtsklick auf die Operation kann man diese mit einem übergebenen Wert testen.

5.2.2 Webservice-Anfrage

Der Zugriff auf den Webservice beginnt mit der Instanziierung eines Service, der am Ende der WSDL-Datei angegeben wird. Dieser Service, in der die WSDL-Datei beschrieben wird, besitzt einen Port, der das wichtigste Element einer WSDL ist. Er definiert in einem Webservice die Operationen, die aufgerufen werden können, und die Nachrichten, die dazu

dienen. Damit definiert der Port die Verbindung zu einem Webservice. Deswegen kann der Zugriff auf die Operationen eines Webservice nur durch den Port erfolgen.

```
 * @author chaouqi
 */
public class Main {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        ATISSPMLWebService_Service myService = new ATISSPMLWebService_Service();
        de.uni_karlsruhe.kim.services._2008._10.atisspmlwebservice.ATISSPMLWebService myPort = myService.getATISSPMLWebServicePort();
        System.out.println("Bitte Fricardnummer eingeben:");
        String fricardnumber= "";

         try {
// Lesen des Keyboardbuffers in myInput
      BufferedReader myInput = new BufferedReader(new InputStreamReader(System.in));
// Kopieren der ersten Zeile (Ende durch 'RETURN-Taste') in String s
      fricardnumber = myInput.readLine();
   } //end try
     catch (Exception err) {
           err.printStackTrace();
       }



        String result = myPort.atisProvisionStudentWithFriCardId(fricardnumber);
        System.out.println(result);
    }
}
```

Abbildung 24 : Auszug aus der Webserviceanfrage-Java-Klasse

Dafür instanziiert man zunächst einen Port (Abbildung 24), der eine Schnittstelle des instanziierten Service ist. Der Port „*myPort*“ ermöglicht nun den Zugriff auf die Operation „*atisProvisionStudentWithFricardId()*“, die die Identitätsdaten eines Studenten liefert anhand seiner Fricardnummer, wenn diese Daten vorhanden sind. Diese Anfrage wird als eine Java-Applikation implementiert. Die Fricardnummer wird von der Kommandozeile eingelesen.

6 TEST UND BEWERTUNG

6.1 Test

In der Softwareentwicklung werden Testverfahren durchgeführt, um Fehler aufzudecken. Die Fehler unterteilen sich in drei verschiedene Klassen. Die Anforderungsfehler entstehen durch inkorrekte Angaben, unvollständige Angaben über funktionale Anforderungen, Inkonsistenz oder Undurchführbarkeit der Anforderungen. Die Entwurfsfehler entstehen durch unvollständige oder fehlerhafte Umsetzung der Anforderung, Inkonsistenz des Entwurfs oder Inkonsistenz zwischen Anforderung, Spezifikation und Entwurf. Die dritte Klasse von Fehlern wird durch fehlerhafte Umsetzung des Entwurfs hervorgerufen [Ba200].

In dieser Arbeit wird die Anwendung nur auf die Implementierungsfehler untersucht. Dafür muss ein Softwaretest durchgeführt werden. Ein Softwaretest führt eine einzelne Softwarekomponente oder eine Konfiguration von Softwarekomponenten unter bekannten Bedingungen (Eingaben und Ausführungsumgebung) aus und überprüft ihr Verhalten (Ausgaben und Reaktionen). Die zu überprüfende Softwarekomponente oder Konfiguration wird Testobjekt genannt. Ein Testfall besteht aus einem Satz von Daten für die Ausführung eines Teils oder des ganzen Testobjekts [Ba200].

Im diesem Fall:

- Testobjekt: Die Anwendung, bestehend aus dem Puffer-LDAP-Server, Webservice und Webservice-Client

- Testfall: 3 verschiedene Eingaben:

    1. Student ist in Datenbank vorhanden
    2. Student ist in Datenbank nicht vorhanden
    3. Student ist in Daten vorhanden aber die Provisionierung ist nicht möglich (dafür wird der LDAP-Server heruntergefahren)

Erster Fall

In diesem Fall wird die Fricardnummer eines Studenten, der in der Datenbank von KIM-IdM vorhanden ist eingegeben, D.h. ein immatrikulierter Informatikstudent.

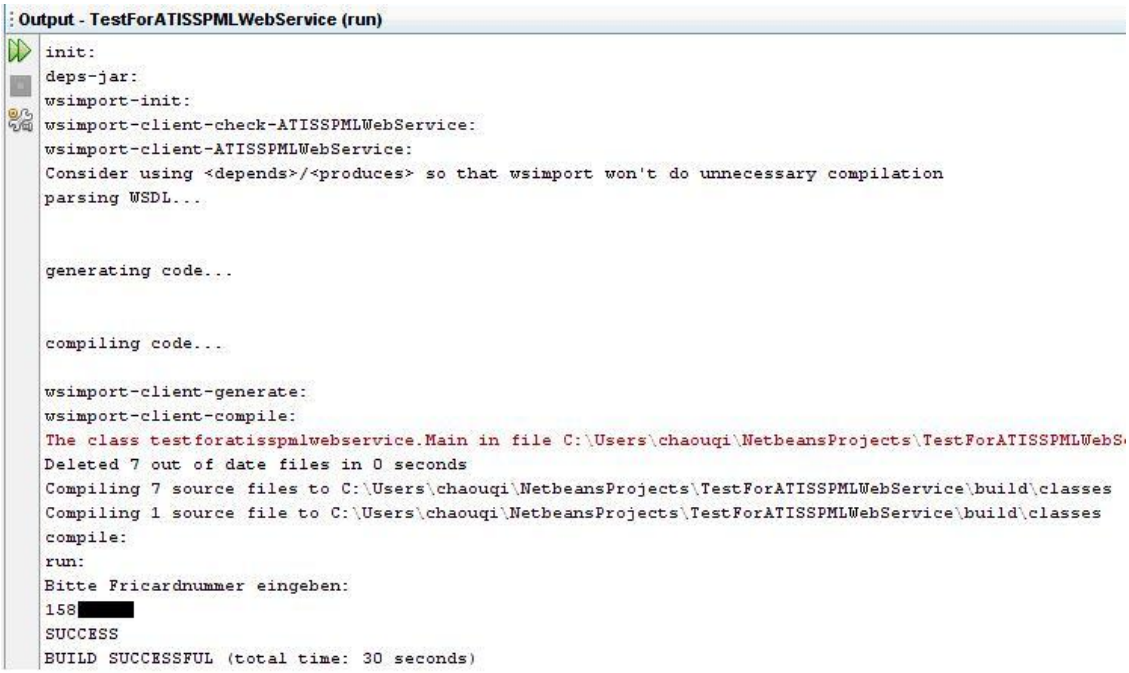


Abbildung 25 : Ausführen des Codes (ester Fall)

Nach dem erfolgreichen Kompilieren und Ausführen des Codes, erscheint auf der Konsole die Auforderung, eine Fricarnummer einzugeben. Diese Fricardummer gehört einem immatrikulierten Informatikstudenten. Das Programm liefert die gewünschte Rückgabe „Success“ (Abbildung 25).

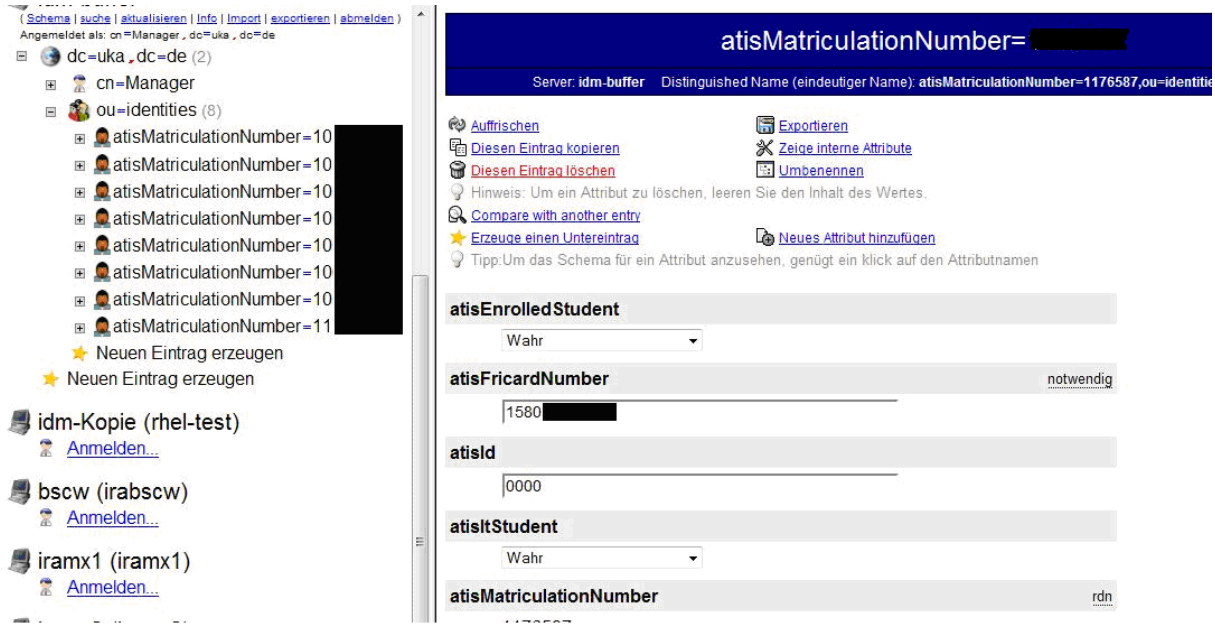


Abbildung 26 : ATISBufferLDAP im PHP-LDAP-Browser

Tatsächlich befindet sich im Puffer-LDAP das angefragte Objekt wie im Bild das LDAP-Browser zeigt (Abbildung 26).

Zweiter Fall

Jetzt wird für die Anfrage die Fricardnummer eines Studenten verwendet, der in der Datenbank von KIM-IdM nicht vorhanden ist.

```
run:
Bitte Fricardnummer eingeben:
158[redacted]
ERROR: User not found.
BUILD SUCCESSFUL (total time: 30 seconds)
```

Abbildung 27 : Ausführung des Codes (zweiter Fall)

Die Meldung „ERROR: User not found“ (Abbildung 27) zeigt, dass der Student nicht gefunden werden konnte, was das erwartete Ergebnis ist. Die Provisionierung ist ebenfalls nicht durchgeführt worden, weil der LDAP keine neuen Einträge hat.

Dritter Fall

In diesem letzten Fall wird dieselbe Fricardnummer wie im ersten Fall verwendet, also die von einem Benutzer, der in der Datenbank vorhanden ist. Aber in diesem Fall wird der BufferLDAP-Server heruntergefahren, um die Provisionierung zu verhindern.

```
compile:
run:
Bitte Fricardnummer eingeben:
158[redacted]
ERROR: User found but Provisioning was not successful.
BUILD SUCCESSFUL (total time: 15 seconds)
```

Abbildung 28 : Ausführung des Codes (dritter Fall)

Die Fehlermeldung (Abbildung 28) bestätigt die Erwartung, dass der Benutzer vorhanden aber die Provsionierung nicht durchführbar ist.

In allen drei Fällen hat sich gezeigt, dass die entwickelte Anwendung funktioniert.

6.2 Bewertung

In diesem Abschnitt wird untersucht, inwieweit die entwickelte Anwendung die gestellten Anforderungen erfüllt. Das Ziel ist, eine Art Bewertung der Anwendung zu machen.

Anforderungen	Zuständige Softwarekomponenten
Anforderungen 1 und 2: Die Abfragen von Identitätsdaten von ATIS aus KIM soll ermöglicht werden. Für die Daten-Anfrage wird ein Attribut benötigt, das bei jeder Identität eindeutig ist	**Webservice-Client:** Diese Komponente ermöglicht die Anfragen an den Webservice von KIM-IdM, das die Identitätsdaten liefert. Für die Anfrage wird die Fricardnummer, die jede Identität eindeutig unterscheidet, verwendet.
Anforderung 3: Die Daten-Provisionierung von KIM aus in Richtung ATIS soll ermöglicht werden.	**Puffer-LDAP**: Mit diesem LDAP-Server kann die Identitätsdatenprovisionierung erfolgen. Das KIM-IdM kann die Daten in diesen LDAP-Sever provisionieren
Anforderung 4: KIM-IDM provisioniert ATIS-IDM mit einer Menge aus Attributen aus denen eine neue ATIS-Identität generiert werden kann. Aus Datenschutz Gründen sollte diese Menge möglichst minimal sein und nur aus notwendigen Attributen bestehend.	**LDAP-Schema**: In dem Schema werden alle mit KIM-IdM Attribute definiert. In der Analyse-Phase wurde die Auswahl jedes einzelne Attribut genau begründet. Somit erfüllen sie diese Anforderung.
Anforderung 5: Die provisionierten Daten sollen fortlaufend und zeitnah synchronisiert werden.	**KIM-IdM-Applikation und Puffer-LDAP**: Die Daten, die in den Puffer-LDAP schon provisioniert wurden, werden automatisch von der KIM-IdM-Applikation aktualisiert.

Tabelle 7 : Anforderungen und ihre erfüllenden Softwarekomponenten

Natürlich wurden viele Einschränkungen bei der Entwicklung der Anwendung vorgenommen, um den Rahmen einer Studienarbeit nicht zu sprengen. Von daher ist die Implementierung nur prototypisch. Trotzdem hat die Anwendung gezeigt, dass es möglich ist, die Identitätsdaten zwischen den beiden Systemen abzugleichen. Genau deswegen hat diese Anwendung die gestellten funktionalen Kernanforderungen erfüllt und ist bestimmt in der Lage, weiter entwickelt zu werden, um andere Anforderungen zu erfüllen.

7 ZUSAMMENFASSUNG UND AUSBLICK

In dieser Arbeit wurde gezeigt, wie eine Verbindung zu KIM-IdM auf der Seite der ATIS entwickelt wird, um das neue im Aufbau befindliche Identitätsmanagementsystem bezüglich der automatischen Erfassung und Verifikation von Identitätsdaten zu unterstützen. Zuerst wurde das Themenumfeld Identitätsmanagement vorgestellt. Danach wurden die Herausforderungen, die die „Abteilung für Infrastruktur der Fakultät für Informatik“ (ATIS) bei ihrem Identitätsmanagement bewältigen muss, gezeigt, und verdeutlicht welche Möglichkeiten KIM der ATIS bieten kann, um ihr Identitätsmanagement zu optimieren. Die eingesetzten Technologien zur Bewältigung der Aufgabe werden ebenso erläutert. Die Entwicklung wurde in vier Schritte gegliedert. Im ersten Schritt wurde das bestehende System ATIS-IdM in Bezug auf seine Funktionalität analysiert, was das Verständnis für die Fragestellung und die Problematik verdeutlicht hat. Zur Analyse wurden zwei Anwendungsfälle untersucht und Geschäftsprozesse aufgezeigt, um eine prozessbezogene Sicht auf die dynamischen Aspekte der modellierten Fälle zu zeigen. Das Optimierungspotenzial, das die Inanspruchnahme des KIM-IdM-Diensts vom ATIS-IdM bietet, wurde analysiert und als Optimierung der zuvor analysierten Anwendungsfälle vorgestellt. Im zweiten Schritt wurde von den Anforderungen ausgehend ein Modell erstellt, das sie erfüllt. Es besteht aus einer Sicht der Gesamtarchitektur der beiden gekoppelten Systeme, dem Entwurf der Bestandteile und Struktur des Puffer-LDAP-Schemas und die Beschreibung des Ablaufs der Datenprovisionierung. Der Puffer dient zur indirekten Lieferung der ATIS-IdM mit Daten von KIM-IdM. Der dritte Schritt ist die Implementierung. Sie wurde nur prototypisch und teilweise realisiert. Zuerst wurden die Installation und die Konfiguration des Puffer-LDAP beschrieben. Dann wurde gezeigt, wie die Provisionierungswebservice-Schnittstelle beschrieben wird. Schließlich wurde verdeutlicht, wie auf den Webservice anhand eines Webservice-Clients zugegriffen wird. Im letzten Schritt wurde ein Test durchgeführt um die Anwendung zu testen und, basierend auf den Anforderungen, die Anwendung bewertet.

Die Anwendung wurde nur soweit entwickelt, dass das ATIS-IdM den Provisionierungsdienst von KIM-IdM in Anspruch nimmt. Dies bedeutet, die Anwendung ermöglicht es, aktuelle Identitätsdaten abzufragen und ständig zu aktualisieren. Aber diese aktuellen Daten sind vorerst nur im Puffersystem. Wie sie verwendet werden, wird vom ATIS-IdM-Team entworfen, implementiert und in das IdM-System der Fakultät integriert.

Die Automatisierung der Identitätsmanagementprozesse im ATIS-IdM können noch erweitert werden. Die Fricards der Studenten verfügen über einem Chip. Im Chip ist ein Identifikator (ID) gespeichert und mittels eines Lesegeräts kann dieser Identifikator ausgelesen werden. Somit kann die Eingabe der Fricardnummer in der Managementapplikation automatisiert werden. In Zukunft wird ein Student nur seine Fricard auf ein Lesegerät legen, um einen neuen Account anlegen zu lassen.

8 ANHÄNGE

8.1 Literatur

[Ba100] Lehrbuch der Software-Technik 1. Band (2.Auflage) - Helmut Balzert, Heidelberg; Berlin, 2000 (1.Auflage 1995); Spektrum Akademischer Verlag

[Ba200] Lehrbuch der Software-Technik 2. Band (2.Auflage) - Helmut Balzert, Heidelberg; Berlin, 2000 (1.Auflage 1995); Spektrum Akademischer Verlag

[BR+06] Das UML Benutzerhandbuch – Grady Booch, James Rumbaugh, Ivar Jacobsen; Addison-Wesley 2006

[CentOS] CentOS Homepage; http://www.centos.org/

[DH+08] Netzwerk- und IT-Sicherheitsmanagement**,** Eine Einführung - Dinger Jochen, Hartenstein Hannes; Universitätsverlag Karlsruhe 2008

[HKRG+03] Datenschutzaspekte von Identitätsmanagementsystemen - Marit Hansen, Henry Krasemann, Martin Rost, Riccardo Genghini; Datenschutz und Datensicherheit (27) 2003; https://www.datenschutzzentrum.de/projekte/idmanage/DUD-27-9.pdf

[HS+06] Dienstorientiertes Identitätsmanagement für eine Pervasive University – Torsten Höllrigl, A. Maurer, Frank Schell, Horst Wenske, Hannes Hartenstein; Karlsruhe, 2006

[HS+07] Föderatives und dienstorientiertes Identitätsmanagement im universitären Kontext – Torsten Höllrigl, Frank Schell, Horst Wenske, Hannes Hartenstein; Karlsruhe, 2007

[KL+08] LDAP verstehen. OpenLDAP einsetzen – Dieter Klünter, Jochen Laser; dpunkt.verlag 2008

[LDB+06] PRIME General Public Tutorial V1 - Katja Liesebach, ilko Donker, Katrin Borcea-Pfitzmann; Privacy and Identity Management for Europe 2006; http://www.prime-project.eu/

[LU+06] OpenLDAP - Oliver Liebell, John Martin Ungar; Galilieo Press 2006

[OL] OpenLDAP; www.openldap.org

[OMG] Object Management Group; http://www.omg.org/

[RFC2251] LDAPv3 – Mark Wahl / Tim Howes / Steve Kille; 1997 http://www.ietf.org/rfc/rfc2251.txt

[RFC2252] Lightweight Directory Access Protocol (v3): Attribute Syntax Definitions - M. Wahl/ A. Coulbeck/ T. Howes/ S. Kille; 1997
http://www.ietf.org/rfc/rfc2252.txt

[RFC4512] Lightweight Directory Access Protocol (LDAP): Directory Information Models - K. Zeilenga; 2006

[Tc04] Web Services Description Language (WSDL) im Überblick - Carlos Tpang; Microsoft Developer Network (MSDN) 2004

[UML] Unified Modeling Language; www.uml.org

[W3C] The World Wide Web Consortium; www.w3.org

[W3Cs00] Simple Object Access Protocol (SOAP) 1.1; W3C 2000

[W3Cw01] Web Services Description Language (WSDL) 1.1; W3C 2001

[Ws04] Introduction to BPMN - Stephen A. White; IBM 2004

8.2 Abbildungsverzeichnis

8.3 Tabellenverzeichnis

8.4 Codefragmentverzeichnis

8.5 LDAP-Konfigurationsdateien

slapd.conf

```
#
# See slapd.conf(5) for details on configuration options.
# This file should NOT be world readable.
#
include         /etc/openldap/schema/core.schema
include         /etc/openldap/schema/cosine.schema
include         /etc/openldap/schema/inetorgperson.schema
include         /etc/openldap/schema/nis.schema
include         /etc/openldap/schema/atis/atisldap.schema

# Allow LDAPv2 client connections.  This is NOT the default.
#allow bind_v2

# Do not enable referrals until AFTER you have a #working
directory
# service AND an understanding of referrals.
#referral       ldap://root.openldap.org

pidfile         /var/run/openldap/slapd.pid
```

```
argsfile        /var/run/openldap/slapd.args

schemacheck on

# Load dynamic backend modules:
# modulepath    /usr/lib/openldap
# moduleload    back_bdb.la
# moduleload    back_ldap.la
# moduleload    back_ldbm.la
# moduleload    back_passwd.la
# moduleload    back_shell.la

# The next three lines allow use of TLS for encrypting
#connections using a
# dummy test certificate which you can generate by #changing to
# /etc/pki/tls/certs, running "make slapd.pem", and fixing
permissions on
# slapd.pem so that the ldap user or group can read it.  Your
client software
# may balk at self-signed certificates, however.
# TLSCACertificateFile /etc/pki/tls/certs/ca-bundle.crt
# TLSCertificateFile /etc/pki/tls/certs/slapd.pem
# TLSCertificateKeyFile /etc/pki/tls/certs/slapd.pem

# Sample security restrictions
#       Require integrity protection (prevent hijacking)
#       Require 112-bit (3DES or better) encryption for updates
#       Require 63-bit encryption for simple bind
# security ssf=1 update_ssf=112 simple_bind=64

# Sample access control policy:
#       Root DSE: allow anyone to read it
#       Subschema (sub)entry DSE: allow anyone to read #it
#       Other DSEs:
#               Allow self write access
#                 Allow authenticated users read access
#                 Allow anonymous users to authenticate
#       Directives needed to implement policy:
# access to dn.base="" by * read
# access to dn.base="cn=Subschema" by * read
# access to *
# access to *
#       by self write
#       by users read
#       by anonymous auth
#
# if no access controls are present, the default policy
# allows anyone and everyone to read anything but
#restricts
# updates to rootdn.  (e.g., "access to * by * #read")
#
# rootdn can always read and write EVERYTHING!

#######################################################################
#############
# ldbm and/or bdb database definitions
#######################################################################
#############
```

```
database        bdb
suffix          "dc=uka,dc=de"
rootdn          "cn=Manager,dc=uka,dc=de"
# Cleartext passwords, especially for the rootdn, should
# be avoided.  See slappasswd(8) and slapd.conf(5) for
details.
# Use of strong authentication encouraged.
 rootpw                {crypt}ijFYNcSNctBYg

# The database directory MUST exist prior to running slapd
AND
# should only be accessible by the slapd and slap tools.
# Mode 700 recommended.
directory       /var/lib/ldap

# Indices to maintain for this database
#index objectClass                      eq,pres
#index ou,cn,mail,surname,givenname     eq,pres,sub
#index uidNumber,gidNumber,loginShell   eq,pres
#index uid,memberUid                    eq,pres,sub
#index nisMapName,nisMapEntry           eq,pres,sub

# identities
index atisId,cn,gn,sn,title,initials,displayName
pres,eq,sub
index departmentNumber,employeeNumber,employeeType
pres,eq,sub
index
telephoneNumber,internationaliSDNNumber,homePhone,mobile,pa
ger pres,eq,sub
index fax,telexNumber pres
index
l,postalCode,st,street,roomNumber,preferredLanguage,secreta
ry pres,eq
index postalAddress,registeredAddress pres
index userCertificate,userSMIMECertificate,jpegPhoto,photo
pres
index atisExternalExpired pres,eq
# Replicas of this database
#replogfile /var/lib/ldap/openldap-master-replog
#replica host=ldap-1.example.com:389 starttls=critical
#     bindmethod=sasl saslmech=GSSAPI
#     authcId=host/ldap-master.example.com@EXAMPLE.COM
access to *  by * read
```

atisBufferSchema

```
 # OID prefix for University of Karlsruhe:
 objectIdentifier unikaOID 1.3.6.1.4.1.87
 # OID prefix for Abteilung technische Infrastruktur (ATIS):
 objectIdentifier atisOID unikaOID:4
 # ATIS LDAP definitions
 objectIdentifier atisLdapOids atisOID:2
 # ATIS LDAP attribute type definitions
 objectIdentifier atisLdapAttrs atisLdapOids:1
 # ATIS LDAP object class definitions
```

```
objectIdentifier atisLdapClasses atisLdapOids:2

attributetype    (    atisLdapAttrs:14    NAME    (    'atisId'
'atisIdentifier' )
        EQUALITY caseIgnoreMatch
        SUBSTR caseIgnoreSubstringsMatch
        ORDERING caseIgnoreOrderingMatch
        DESC  'Unique  key  identifying  a  person  within  the
Fakultät für Informatik - University of Karlsruhe.'
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.15
        SINGLE-VALUE )

attributetype  (  atisLdapAttrs:50  NAME  (  'kimGuid'  'KIM-
GloballyUniqueIdentifier' )
        EQUALITY caseIgnoreMatch
        SUBSTR caseIgnoreSubstringsMatch
        ORDERING caseIgnoreOrderingMatch
        DESC 'Unique key identifying a person within the KIM-IDM
- University of Karlsruhe.'
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.15
        SINGLE-VALUE )

attributetype ( atisLdapAttrs:34 NAME 'atisMatriculationNumber'
        DESC 'Matriculation number assigned by the University of
Karlsruhe'
        EQUALITY integerMatch
        ORDERING integerOrderingMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.27
        SINGLE-VALUE )

attributetype ( atisLdapAttrs:51 NAME 'atisFricardNumber'
        DESC 'Student card number assigned by the University of
Karlsruhe'
        EQUALITY integerMatch
        ORDERING integerOrderingMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.27

        SINGLE-VALUE )
attributetype ( atisLdapAttrs:17 NAME 'atisModifyTimestamp'
        DESC 'Timestamp of last modification by a person'
        EQUALITY generalizedTimeMatch
        ORDERING generalizedTimeOrderingMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.24
        SINGLE-VALUE )

attributetype ( atisLdapAttrs:19 NAME 'atisCreationTimestamp'
        DESC 'Timestamp of creation by a person'
        EQUALITY generalizedTimeMatch
        ORDERING generalizedTimeOrderingMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.24
        SINGLE-VALUE )

attributetype ( atisLdapAttrs:27 NAME 'atisExternalExpired'
        DESC 'TRUE if this identity is no longer valid in the
external identity management.'
        EQUALITY booleanMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.7
        SINGLE-VALUE )

attributetype ( atisLdapAttrs:28 NAME 'atisEnrolledStudent'
        DESC 'TRUE if this student is enrolled.'
```

```
        EQUALITY booleanMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.7
        SINGLE-VALUE )

attributetype ( atisLdapAttrs:29 NAME 'atisItStudent'
        DESC 'TRUE if this student is enrolled at the faculty of
information technoligy.'
        EQUALITY booleanMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.7
        SINGLE-VALUE )

attributetype ( atisLdapAttrs:48 NAME ( 'atisDisabled' )
        EQUALITY booleanMatch
        DESC 'tada'
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.7
        SINGLE-VALUE )

attributetype ( atisLdapAttrs:52 NAME ( 'atisToBeDeleted' )
        EQUALITY booleanMatch
        DESC 'tada'
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.7
        SINGLE-VALUE )

# ATIS LDAP object classes
# OID subtree 1.3.6.1.4.1.87.4.2.2
# objectidentitier is atisLdapClasses

objectclass (  atisLdapClasses:30 NAME 'atisBufferIdentity'
        SUP inetOrgPerson
        STRUCTURAL
        DESC 'Identity of a person.'
        MUST (  cn $ sn   $ kimGuid $ atisFricardNumber   )
        MAY  (  gn  $  atisExternalExpired  $  atisDisabled  $
atisMatriculationNumber $ atisId  ) )
```