Verteilte Autorisierung innerhalb von Single Sign-On-Umgebungen

Analyse, Architektur und Implementation eines Frameworks für verteilte Autorisierung in einer ADFS-Umgebung

Diplomarbeit

an der Universität Potsdam

Peter Kirchner

1. Gutachter: Prof. Dr.-Ing. Klaus Rebensburg
2. Gutachter: Prof. Dr.-Ing. Tiziana Margaria-Steffen

Potsdam, März 2007

Online veröffentlicht auf dem
Publikationsserver der Universität Potsdam:
http://opus.kobv.de/ubp/volltexte/2008/2228/
urn:nbn:de:kobv:517-opus-22289
[http://nbn-resolving.de/urn:nbn:de:kobv:517-opus-22289]

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides Statt, dass ich die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht.

Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungskommission vorgelegt und auch nicht veröffentlicht.

Ort, Datum Unterschrift (Vor- und Nachname)

Inhalt

Abstract

Deutsch

Aktuelle Softwaresysteme erlauben die verteilte Authentifizierung von Benutzern über Verzeichnisdienste, die sowohl im Intranet als auch im Extranet liegen und die über Domänengrenzen hinweg die Kooperation mit Partnern ermöglichen. Der nächste Schritt ist es nun, die Autorisierung ebenfalls aus der lokalen Anwendung auszulagern und diese extern durchzuführen – vorzugsweise unter dem Einfluss der Authentifizierungspartner.

Basierend auf der Analyse des State-of-the-Art wird in dieser Arbeit ein Framework vorgestellt, das die verteilte Autorisierung von ADFS (Active Directory Federation Services) authentifizierten Benutzern auf Basis ihrer Gruppen oder ihrer persönlichen Identität ermöglicht. Es wird eine prototypische Implementation mit Diensten entwickelt, die für authentifizierte Benutzer Autorisierungsanfragen extern delegieren, sowie ein Dienst, der diese Autorisierungsanfragen verarbeitet. Zusätzlich zeigt die Arbeit eine Integration dieses Autorisierungs-Frameworks in das .NET Framework, um die praxistaugliche Verwendbarkeit in einer aktuellen Entwicklungsumgebung zu demonstrieren.

Abschließend wird ein Ausblick auf weitere Fragestellungen und Folgearbeiten gegeben.

Englisch

Current software systems allow distributed authentication of users using directory services, which are located both in the intranet and in the extranet, to establish cooperation with partners over domain boundaries. The next step is to outsource the authorization out of the local applications and to delegate the authorization decisions to external parties. In particular the authorization request is back delegated to the authentication partner.

Based on an analysis of the state of the art this paper presents a framework which allows the distributed authorisation of ADFS authenticated users. The authorization decisions are based on the user's identity and groups. In this work there will be developed a prototypical implementation of services which are capable of delegating authorization requests. Additionally, this work points out the integration of these services into the .NET framework to demonstrate the usability in a modern development environment.

Finally there will be a prospect of further questions and work.

1 Einleitung und Grundlagen

Einleitung

Das Thema der verteilten Autorisierung umfasst mehrere Sachgebiete der Informatik. Es beinhaltet Themen der Sicherheit, wie Identifizierung, Authentifizierung, Autorisierung und ADFS als Vertreter für ein Single Sign-On-System. In dieser Arbeit werden aktuelle Standards, bisherige Arbeiten und aktuelle Produkte untersucht und bewertet, die für die oben genannten Themen von Bedeutung sind.

Neben der Aufarbeitung dieser Themen hat diese Arbeit die Erstellung eines Frameworks als Ziel, das eine prototypische Implementation der Autorisierungsdienste sowie ein SDK für .NET enthält, womit auf diese Dienste über eine API zugegriffen werden kann.

In früher Zeit waren die Computersysteme lokal und isoliert voneinander. Ressourcen mussten lediglich vor den lokalen Nutzern und Prozessen geschützt werden. Mit der Vernetzung begann auch der Wunsch, die Ressourcen im Netzwerk gemeinsam nutzen zu können. Neben der Nutzung gemeinsamer Ressourcen wurden auch bald zentrale Anwendungen im Netzwerk angeboten, später wurden diese sogar extern zur Verfügung gestellt, damit diese von Partnern oder eigenen Mitarbeitern außerhalb des Firmennetzes genutzt werden konnten. Es wurde nun zwischen Intranet und Extranet unterschieden. Das Intranet steht für alle Dienste im eigenen Firmennetz. Das Extranet dagegen umfasst die Dienste, die von einer Firma extern angeboten werden, das heißt der Zugriff ist auch außerhalb des Firmennetzes möglich. Verteilte Autorisierung ist somit ein konsequenter Schritt der Entwicklung, die die IT bisher genommen hat.

Mit dem wachsenden Angebot der Dienste im Intra- und Extranet wuchsen auch die Bedürfnisse, die Authentifizierung gegenüber diesen Diensten zu vereinfachen. Diesem Zweck dienen Single Sign-On-Systeme (SSO-Systeme). Sie ermöglichen es, auf eine Vielzahl von Diensten nach einmaliger Anmeldung zuzugreifen. Vertreter für SSO-Systeme sind zum Beispiel das Active Directory und die Active Directory Federation Services, die auch in dieser Arbeit vorgestellt und verwendet werden.

Bei der Nutzung von Anwendungen im Extranet können drei Klassen der Authentifizierung und Autorisierung abgeleitet werden:

1. Lokale Authentifizierung und lokale Autorisierung
2. Verteilte Authentifizierung und lokale Autorisierung
3. Verteilte Authentifizierung und verteilte Autorisierung

Die erste Variante stellt vom Aufwand für den Aufbau der notwendigen Infrastruktur die einfachste Variante dar, da sowohl die Benutzerverwaltung als auch die Konfiguration der Privilegien auf einem einzigen System erfolgen.

In der zweiten Variante wird die Benutzerverwaltung aus diesem System ausgegliedert und in einen separaten Dienst verlagert. Diese Variante ist aufwändiger als die erste, da nun eine Vertrauensstellung zwischen dem System, das die Anwendung anbietet, und dem System, das die Benutzerverwaltung betreibt, aufgebaut werden muss. Das System, das die Anwendung anbietet, wird als Resource Partner und das System, das die Benutzerverwaltung betreibt, als Account Partner bezeichnet.

In der dritten Variante wird schließlich auch die Autorisierung in einen separaten Dienst ausgelagert. Das System, das diesen Dienst betreibt, bezeichne ich als Access Control Partner. Der Aufbau einer Infrastruktur nach dieser Variante ist nochmals aufwändiger als Variante 2, da zusätzlich die Vertrauensstellungen zwischen dem Resource Partner und dem Access Control Partner sowie zwischen Account Partner und Access Control Partner eingerichtet werden

müssen. Variante 3 stellt allerdings die flexibelste und mächtigste Konfiguration dar. Warum dies so ist und wie diese Infrastruktur verwirklicht werden kann, ist Thema dieser Arbeit.

1.1 Aufbau der Arbeit

Anschließend an diese Einleitung wird in diesem *ersten Kapitel* in die Grundlagen eingeführt, die zum Verständnis der Arbeit hilfreich sind. Neben der Begriffsbildung werden verwandte Themen wie Authentifizierung, Autorisierung und Standards beschrieben, die bei der Kommunikation zwischen den Systemen verwendet werden. Abschließend wird ein Beispielszenario eingeführt, das im weiteren Verlauf dieser Arbeit für Erklärungen verwendet wird.

Im *Kapitel 2*, die Problemanalyse, wird die Notwendigkeit der verteilten Autorisierung herausgearbeitet. Dazu wird gegenübergestellt, welche Vor- und Nachteile die verteilte Autorisierung gegenüber der lokalen Autorisierung aufweist. Außerdem werden ausgewählte, aktuelle Arbeiten zu diesem Thema und aktuelle Software, die Autorisierungsdienste beinhalten, vorgestellt und bewertet.

Die gewonnenen Erkenntnisse der Problemanalyse führen zu einem Entwurf, der im *Kapitel 3* entwickelt wird. Der Entwurf beinhaltet die Analysen, welche Anwendungsfälle eine Lösung zur verteilten Autorisierung enthalten sollte und wie eine geeignete Architektur dafür aussehen kann.

Dieser Entwurf ist die Grundlage der Realisierung im *Kapitel 4*. Die Realisierung beschreibt, wie der Entwurf konkret umgesetzt wurde. Zudem werden hier Höhepunkte der Realisierung aufgezeigt, also wichtige Entscheidungen oder Hindernisse bei der Implementierung.

Im *Kapitel 5* wird schließlich die Realisierung in ihrer Arbeitsweise und anderen Faktoren bewertet. Es wird der Aufbau der Testumgebung mit einer funktionsfähigen ADFS-Infrastruktur beschrieben, in der die Testanwendung betrieben wird, die ebenfalls in dieser Arbeit erstellt wird. Ausgehend vom Beispielszenario werden Workflows für die Testläufe ausgewählt und auf korrekte Funktion hin überprüft.

In den *Kapiteln 6 und 7* wird die gesamte Arbeit reflektiert. In einer Zusammenfassung und einem Ausblick wird aufgezeigt, was erreicht wurde und welche Anknüpfungspunkte diese Arbeit für eine weitergehende Arbeit in diesem Bereich bietet.

Das Glossar schließt sich im *Kapitel 8* an. Es werden häufig verwendete Abkürzungen und Begriffe kurz erläutert. Wurden die Begriffe in dieser Arbeit eingeführt oder verwendet, werden zusätzlich Verweise auf die entsprechenden Abschnitte gegeben.

Die Verzeichnisse der Literatur, Abbildungen, Tabellen und Quellcode-Auszüge sind im *Kapitel 9* zu finden. *Kapitel 10* enthält schließlich den Anhang, der Auskunft über die beiliegenden Datenträger gibt und weiterführendes Material enthält.

1.2 Lesart

In dieser Arbeit werden Schreibweisen, Formatierungen und Diagramme verwendet, die in ihrer Verwendung Regeln folgen, die in diesem Abschnitt erläutert werden.

1.2.1 Schreibweisen und Formatierungen

Fachtermini, insbesondere die, die der englischen Sprache entstammen, werden in ihrer Schreibung beibehalten, so wie sie in ihrem jeweiligen Kontext verwendet werden. Fachtermini werden in dieser Formatierung kenntlich gemacht. Zum Beispiel gibt es in den Active Directory Federation Services (siehe *Abschnitt 1.4.2* in diesem Kapitel) die Begriffe Account

Partner und Resource Partner. Diese Begriffe werden in ihrer englischen Schreibweise beibehalten.

Die Modellierung des Softwaresystems, das in dieser Arbeit entworfen und entwickelt wird, ist in Deutsch gehalten. Bezeichner von Namensräumen, Komponenten, Klassen, Methoden, Parametern, Variablen und ähnlichem werden dagegen, wie es in der Softwareentwicklung üblich ist, in Englisch benannt.

Innerhalb dieser Arbeit werden viele Themen besprochen, die oft miteinander in Beziehung stehen. Zur Erleichterung der Einordnung ins Thema werden daher oft Querverweise zu den entsprechenden Abschnitten, Abbildungen, Tabellen und Quellcode-Auszügen gegeben. Querverweise werden immer eingeleitet (zum Beispiel: siehe Abschnitt 1.1) und in *dieser Formatierung* kenntlich gemacht. Verweise auf Nachweise im Literaturverzeichnis werden in eckigen Klammern geschrieben (zum Beispiel: [Priebe 2005]).

Bezeichner von Methoden, Variablen, Parametern, sowie Pfade und Adressen werden zur besseren Unterscheidbarkeit ebenfalls vom restlichen Text hervorgehoben. Bezeichner dieser Art werden in *dieser Formatierung* kenntlich gemacht.

1.2.2 Diagramme

Dieser Abschnitt erläutert die verschiedenen Diagrammtypen, die in dieser Arbeit verwendet werden. Darunter sind Fluss-, Verteilungs-, Anwendungsfall-, Klassen- und Sequenzdiagramme zu finden. Die letzten vier genannten Diagramme verwenden den UML-Standard.

Flussdiagramme bringen Aktoren, Prozesse und Dokumente bzw. Daten miteinander in Beziehung. Anhand von *Abbildung 1* wird die Symbolik von Flussdiagrammen erläutert. Aktoren können Prozesse ausführen, sowie Dokumente übermitteln und empfangen. Die Aktoren sind in vertikale Bereiche gegliedert. Prozesse werden durch Rechtecke verdeutlicht. Prozessen können weitere Prozesse folgen oder aber das Ergebnis eines Prozesses ist ein Dokument. Dokumente werden durch Rechtecke dargestellt, deren Unterseite wellenförmig ist. In *Abbildung 1* ist das Ergebnis von *Prozess 1*, der vom *Aktor 2* ausgeführt wird, ein Dokument, das zum *Aktor 1* übermittelt wird.

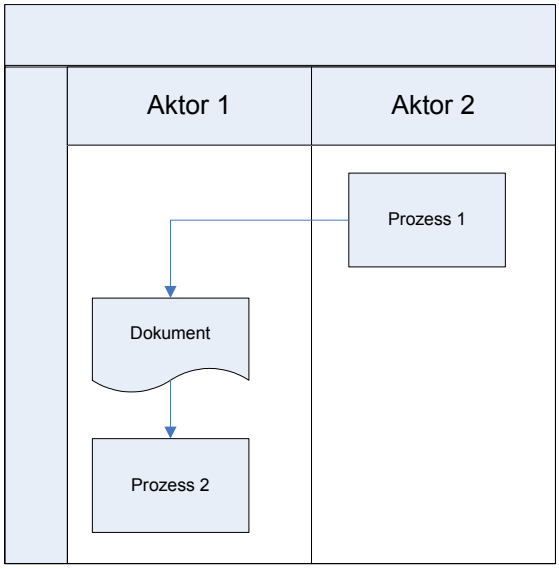

Abbildung 1 Flussdiagramm

Verteilungsdiagramme enthalten Knoten und Artefakte. Knoten repräsentieren dabei eigenständige Computersysteme, meist innerhalb eines Netzwerks. Artefakte sind Programme oder Dienste, die auf diesen Knoten ausgeführt werden. Zwischen den Programmen und Diensten der unterschiedlichen Knoten kann Kommunikation stattfinden. In *Abbildung 2* ist als Bei-

spiel gegeben, dass sich in dem Knoten *Node1* ein Artefakt *Artefact1* befindet, das mit dem Artefakt *Artefact2*, das sich in dem Knoten *Node2* befindet, kommuniziert.

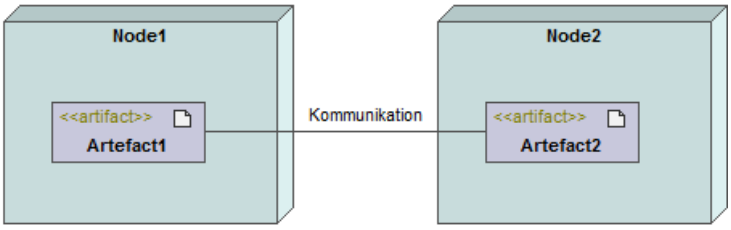

Abbildung 2 Verteilungsdiagramm

Anwendungsfälle beschreiben die Tätigkeiten, die ein Aktor mit einem Softwaresystem ausführen kann. Ein Aktor kann dabei ein Mensch oder ein anderes Softwaresystem sein. Anwendungsfälle werden aus Sicht des Aktors formuliert, also so wie der Aktor die Tätigkeit wahrnimmt. Er hat dabei keine Kenntnis über die genauen Abläufe innerhalb der Software. Die Anwendungsfälle, die ein Aktor direkt ausführen kann, werden mit einer Linie zum Aktor verbunden. Anwendungsfälle können wiederum andere Anwendungsfälle beinhalten. Diese Anwendungsfälle werden mit einer gerichteten, gestrichelten Linie verbunden. Zusätzlich wird die Bezeichnung <<include>> angebracht. Anwendungsfälle können auch voneinander erben. Dabei besitzt der erbende Anwendungsfall all die Eigenschaften der beerbten Anwendungsfälle. Die Symbolik ist ähnlich dem Include mit dem Unterschied, dass die Bezeichnung <<extend>> verwendet wird.

In *Abbildung 3* ist ein Beispiel mit einem Aktor *Aktor1* gegeben. Dieser Aktor besitzt einen direkten Anwendungsfall *Anwendungsfall1*. *Anwendungsfall1* beinhaltet zwei weitere Anwendungsfälle *Anwendungsfall2* und *Anwendungsfall3*. *Anwendungsfall2* erbt vom Anwendungsfall *Anwendungsfall4*.

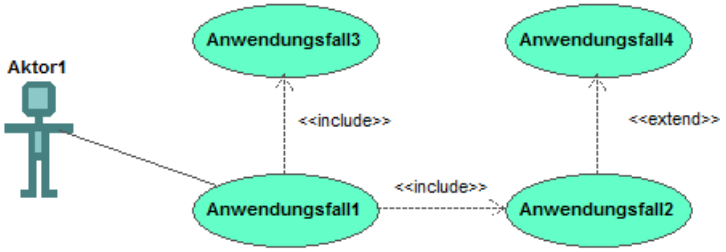

Abbildung 3 Anwendungsfalldiagramm

Klassendiagramme dienen der Modellierung von Klassen und Schnittstellen und bringen diese miteinander in Beziehung. Klassen und Schnittstellen werden durch Rechtecke dargestellt, die in drei Bereiche gegliedert sind. Der erste Bereich enthält den Namen der Klasse bzw. der Schnittstelle. Der zweite Bereich enthält eine Auflistung der Variablen und der dritte Bereich eine Auflistung der Methoden. Schnittstellen tragen zur Unterscheidung von den Klassen im ersten Bereich den Stereotyp <<interface>>. Schnittstellen können keine Variablen besitzen.

Klassen und Schnittstellen werden in Paketen gruppiert. Pakete werden durch ein gelbes Rechteck mit einem Karteireiter symbolisiert. Im Beispiel in *Abbildung 4* sind alle Klassen Teil des Pakets *AuthorizationLib*. In der Regel sind Pakete Bibliotheken oder ausführbare Programme. In .NET werden Bibliotheken auch als Assemblies bezeichnet.

Variablen und Methoden können in UML vier verschiedene Zugriffsmerkmale aufweisen: öffentlich, privat, geschützt und intern. Öffentliche Methoden oder Variablen können von allen anderen Klassen benutzt werden. Private Methoden oder Variablen können nur von Klassen des gleichen Typs verwendet werden. Geschützte Methoden oder Variablen können nur von Klassen benutzt werden, die von der Klasse erben, die die geschützten Methoden oder Variablen enthält. Interne Methoden und Variablen können nur von Klassen benutzt werden,

die sich im gleichen Paket befinden. In dem Beispiel in *Abbildung 4* ist die Variable *Property1* und die Methode *Operation1* öffentlich, *Property2* und *Operation2* privat, *Property3* und *Operation3* geschützt und *Property4* und *Operation4* intern.

UML sieht nicht alle Aspekte moderner Programmiersprachen vor. Die Realisierung des verteilten Autorisierungssystems in dieser Arbeit basiert auf .NET als Plattform und C# als Programmiersprache. C# enthält ein Sprachelement namens Eigenschaft (engl. property). Eigenschaften sind in ihrem Verhalten eine Mischung aus öffentlichen Variablen und Methoden. Beim Zuweisen und beim Abrufen von Eigenschaften sind sie nicht von öffentlichen Variablen zu unterscheiden. Eigenschaften können jedoch beim Lesen oder Setzen Anweisungen ausführen und sehen dadurch Methoden sehr ähnlich.

Eigenschaften werden in dieser Arbeit als öffentliche Variablen modelliert. Variablen werden generell nicht modelliert, ob privat oder öffentlich. Sollte von dieser Regel abgewichen werden, wird darauf explizit hingewiesen. Eigenschaften können schreibgeschützt sein. In diesen Fällen steht hinter den Eigenschaften im Klassendiagramm der Hinweis {readOnly}.

Klassen und Schnittstellen können auf vielfältige Weise in Beziehung stehen. Die einfachste Beziehung ist die Benutzen-Beziehung. Sie bedeutet, dass eine Klasse auf eine nicht weiter definierte Weise eine andere Klasse zur Ausführung benötigt. Im Beispiel in *Abbildung 4* benutzt die Klasse *Class1* die Klasse *Class5*.

Verwendet eine Klasse eine andere Klasse als Eigenschaft, wird dies durch eine gerichtete, durchgezogene Linie mit einem ausgefüllten Punkt auf der nutzenden Seite symbolisiert. An die Linie wird der Name der Eigenschaft inklusive dem Sichtbarkeitskürzel geschrieben. Plus (+) steht für öffentlich, Minus (-) für privat, Raute (#) für geschützt und Tilde (~) für intern. Im Beispiel besitzt die Klasse *Class2* eine öffentliche Eigenschaft *Property5* mit dem Typ *Class1*.

Spezialisierte Formen der Nutzung von Klassen sind die Aggregation und Komposition. Sie werden verwendet, wenn Klassen Teilmengen anderer Klassen repräsentieren. Zum Beispiel kann es eine Klasse *Straße* und eine Klasse *Stadt* geben. Dann ist die Klasse *Straße* eine Teilmenge von der Klasse *Stadt*. Wird die Klasse, die eine Teilmenge darstellt, von der anderen Klasse verwaltet – also erzeugt und zerstört, so spricht man von Komposition ansonsten von Aggregation. Aggregation wird durch eine nicht ausgefüllte und Komposition durch eine ausgefüllte Raute auf der Seite der Klasse dargestellt, die die Teilmengen-Klasse beinhaltet. Im Beispiel stellt die Klasse *Class3* eine Teilmenge von der Klasse *Class1* dar, wobei *Class1* nicht für die Erzeugung und die Zerstörung von *Class3* verantwortlich ist – eine Aggregation liegt vor. Klasse *Class4* wird dagegen von der Klasse *Class1* erzeugt und zerstört. Hier liegt eine Komposition vor.

Implementiert eine Klasse ein Interface wird dies durch eine gerichtete, gestrichelte Linie mit einer leeren Spitze zum Interface dargestellt. Im Beispiel implementiert die Klasse *Class1* das Interface *Interface1*. Auch die Vererbung kann in UML symbolisiert werden. In diesem Fall wird eine gerichtete, durchgezogene Linie mit einer leeren Spitze zur Basisklasse verwendet. Im Beispiel erbt die Klasse *Class3* von der Klasse *Class4*.

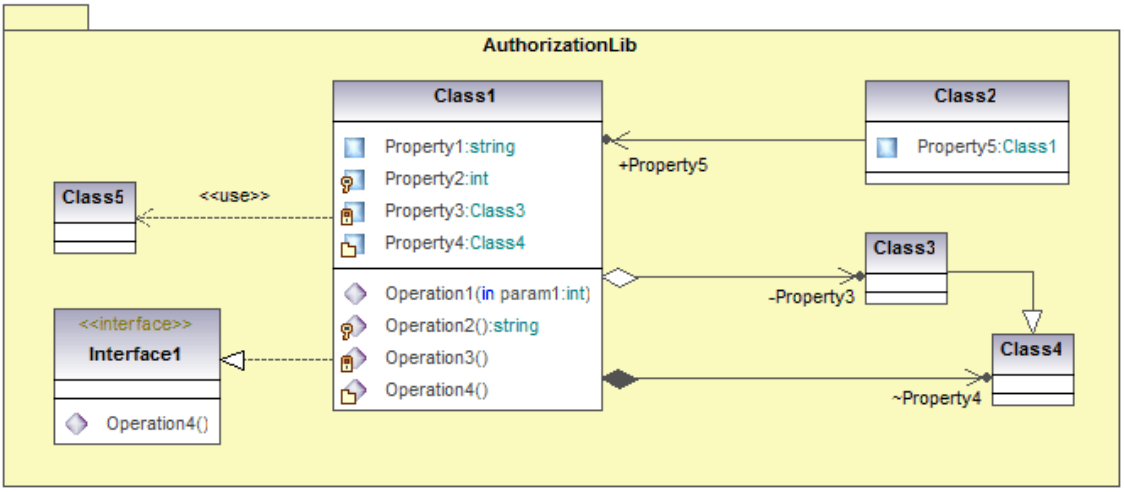

Abbildung 4 Klassendiagramm

Sequenzdiagramme verdeutlichen Abläufe von Operation von Klassen. Jede Klasse wird durch ein oder mehrere Instanzen vertreten, die in Sequenzdiagrammen als Lebenslinien bezeichnet werden. Lebenslinien werden durch ein Rechteck oben, das den Namen der Klasse und den Bezeichner der Instanz enthält, und durch eine vertikale Linie nach unten dargestellt. Weiße und graue Rechtecke auf dieser Lebenslinie symbolisieren Aktivität während eines Ablaufs. Die grauen Rechtecke haben die Spezialbedeutung einer verzweigten Aktivität innerhalb derselben Instanz. Der Ablauf wird oft durch eine anonyme Quelle gestartet, die als Gate bezeichnet wird und durch ein kleines Quadrat repräsentiert wird. Aufrufe von Methoden werden durch gerichtete, durchgezogene Linien dargestellt. Der Aufruf wird mit einer Sequenznummer und dem Namen der aufgerufenen Methode beschriftet. Die Erzeugung von Instanzen wird durch eine gerichtete, gestrichelte Linie dargestellt. Die Beendigung einer Methode wird ebenfalls durch eine gerichtete, gestrichelte Linie dargestellt, die zurück zur aufrufenden Instanz zeigt. Verzweigungen und Schleifen werden in Rechtecken gruppiert, die in der oberen, linken Ecke die Art der Verzweigung oder Schleife trägt.

In dem Beispiel in *Abbildung 5* ruft eine anonyme Quelle die Methode *Operation1* der Instanz *a* vom Typ *Class1* auf. *a* existiert zu diesem Zeitpunkt bereits, über die Erzeugung von *a* sagt das Sequenzdiagramm hier nichts aus. Die Methode *Operation1* in *Class1* erzeugt eine neue Instanz *b* vom Typ *Class2*. *Class1* ruft dann ihrerseits die Methode *Operation2* in *b* auf. *Class2* ruft während der Ausführung von *Operation2* eine Methode *Operation3* innerhalb der eigenen Instanz auf. *Operation2* und *Operation1* terminieren schließlich und geben die Ausführung an die anonyme Quelle zurück.

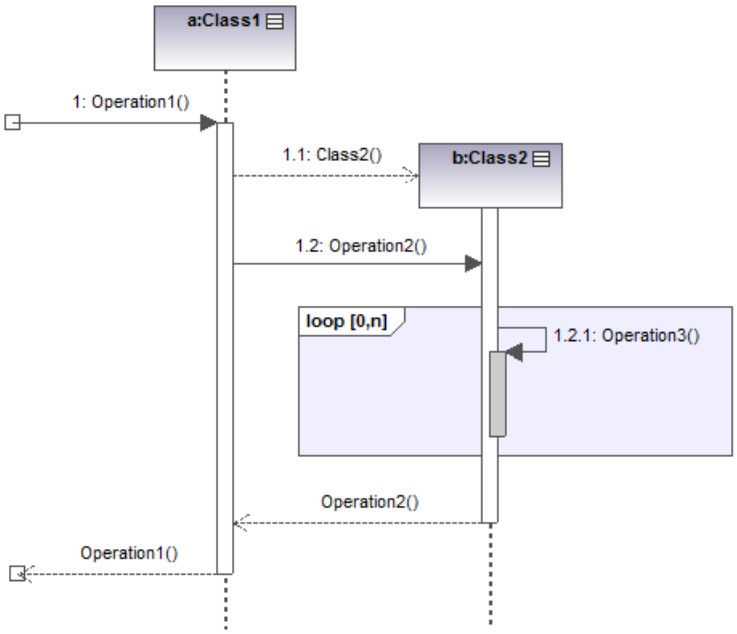

Abbildung 5 Sequenzdiagramm

1.3 Begriffsbildung

Dieser Abschnitt führt Begriffe ein, die eine spezielle Bedeutung in dieser Arbeit besitzen. Allgemeine Begriffe wie Authentifizierung und Autorisierung folgen im nächsten Abschnitt der *Einleitung und Grundlagen*. Etliche Begriffe, die in dieser Arbeit nicht neu eingeführt und in einer bekannten Bedeutung verwendet werden, sind im Glossar im *Kapitel 8* zu finden.

Der Prototyp, der in der dieser Arbeit implementiert wurde, wird als verteiltes Autorisierungssystem bezeichnet. Dieses umfasst die Serverdienste inklusive dem Framework, das vom Resource Partner für die Übermittlung von Autorisierungsanfragen und dem Empfang von Autorisierungsantworten verwendet wird.

Diese Arbeit unterscheidet zwischen lokaler und verteilter Autorisierung. Bei der lokalen Autorisierung findet die Zugriffskontrolle innerhalb des Prozesses statt, der die Zugriffskontrolle auf eine Ressource veranlasst hat. Bei der verteilten Autorisierung hingegen findet die Zugriffskontrolle außerhalb des Prozesses statt. Dabei ist irrelevant, ob die Autorisierung an einen Prozess delegiert wird, der auf dem gleichen System oder auf einem anderen System im Netz ausgeführt wird. Anders umschrieben, kann gesagt werden, dass die lokale Autorisierung von einer Anwendung eigenverantwortlich durchgeführt wird, während bei der verteilten Autorisierung die Zugriffskontrolle an eine fremde Instanz delegiert wird.

1.4 Authentifizierung

Die folgenden Abschnitte beschäftigen sich mit grundlegenden Themen der Authentifizierung, die im Rahmen dieser Arbeit relevant sind. Verzeichnisdienste bilden die Basis für die Speicherung und Anfragen von Sicherheitsobjekten wie Benutzern und Computern. Über eine kurze Einführung allgemeiner Verzeichnisdienste wird direkt Kurs auf den Verzeichnisdienst Active Directory genommen, der die Grundlage für Windows Netzwerke darstellt sowie für die Active Directory Federation Services, die als Single Sign-On-System in der Testumgebung für die prototypische Implementation der verteilten Autorisierungsdienste, die in dieser Arbeit modelliert und implementiert wurden, verwendet werden.

"Die Überprüfung der Authentizität der Objekte und Subjekte ist die Voraussetzung für die Realisierung weiterer Sicherheitsanforderungen wie Integrität und Vertraulichkeit." [Eckert

2002, S. 366] Die Art und Weise, wie die Identität von Subjekten (Benutzer oder vom Benutzer beauftragte Prozesse) und Objekten festgestellt wird, ist nicht Thema dieser Arbeit. Gängige Verfahren sind Benutzername und Passwort oder die Verwendung von Smartcards. Für das verteilte Autorisierungssystem ist die Authentifizierung Grundlage für die Autorisierung.

Active Directory verwendet, ebenso wie ADFS, standardmäßig Kerberos für die Authentifikation. Daher wird auch in dieses Authentifikations-Protokoll vorgestellt und die Grundlagen vermittelt.

1.4.1 Verzeichnisdienste

Im Allgemeinen ist ein Verzeichnis eine Datenbank, die eine Menge von Objekten speichert, denen jeweils Attribute zugeordnet sind. Diese Attribute werden mit Werten belegt, den eigentlichen Daten der Datenbank. Die Benutzer eines Verzeichnisses sind immer Prozesse, hinter denen Benutzer oder Applikationen stehen.

Einer der ersten Verzeichnisdienste war der WHOIS-Dienst [Daigle 2004], der anfangs dafür gedacht war, zu IP-Adressen bzw. -Bereichen die verantwortlichen Administratoren herauszufinden. WHOIS konnte allerdings nur als der erste Versuch eines Verzeichnisdienstes angesehen werden. Die gravierendsten Probleme dieses Dienstes waren seine zentrale Datenbank, nicht performante Antwortzeiten und die manuelle Replikation [Klement 2003].

1.4.1.1 X.500

Eine Antwort auf die Nachfrage nach einem besseren Verzeichnisdienst war der X.500-Standard, hinter dem sich eine ganze Reihe weiterer Standards versteckt, die alle einen sehr umfassenden, aber auch komplexen Verzeichnisdienst beschreiben.

Die Funktionsweise von X.500 zu erklären, würde den Rahmen dieser Arbeit sprengen und auch nicht wesentlich zum Verständnis dieser Arbeit beisteuern. Es werden nur die wichtigsten Eigenschaften von X.500 und ein Kursabriss dargestellt, um ein Grundverständnis zu vermitteln. Hier einige Schlagworte, die X.500 beschreiben.

- Dezentralisierte Verwaltung
- Umfassende Abfragemöglichkeiten
- Globaler Namenskontext
- Strukturierte Informationsbasis

Ein X.500-Verzeichnis ist ein kooperierendes offenes System. Das bedeutet, dass das Verzeichnis über mehrere Systeme verteilt werden kann. Damit die Daten des Verzeichnisses auf den unterschiedlichen Systemen aktuell vorliegen, ist eine Replikation der Daten notwendig.

Beim X.500-Standard wird die Masterreplikation verwendet, die in diesem Standard auch als Shadowing bezeichnet wird. Man unterscheidet bei der Replikation zwischen Master- und Multimasterreplikation. Die Masterreplikation ist wesentlich einfacher umzusetzen als die Multimasterreplikation. Bei der Masterreplikation gibt es einen Master-Server und viele Slave-Server. Änderungen des Verzeichnisses sind nur auf dem Master-Server erlaubt. Von dort aus werden alle Daten und Veränderungen zu den Slave-Servern übertragen, die lesenden Zugriff auf das Verzeichnis ermöglichen. Bei der Multimasterreplikation gibt es mehrere Master, auf denen allen Änderungen am Verzeichnis erlaubt sind. Über Synchronisierungsmechanismen werden die Änderungen am Verzeichnis in beide Richtungen übertragen. Der Entwurf dieser Synchronisierungsmechanismen und das Behandeln von Konfliktfällen (wenn dieselben Daten auf mehreren Systemen gleichzeitig geändert wurden) ist gleichzeitig der Vor- und Nachteil der Multimasterreplikation, bei der zwar der Implementierungsaufwand höher ist, das Verzeichnis aber flexibler und skalierbarer als die einfache Masterreplikation

ist. Active Directory, das im *Abschnitt 1.4.1.3* vorgestellt wird, verwendet dagegen die Multimasterreplikation.

Jedes Objekt im Verzeichnis muss einen eindeutigen Namen besitzen. Der Name eines einzelnen Objekts wird als Relative Distinguished Name (RDN) bezeichnet. Der vollständige Name, um auf ein beliebiges Objekt zuzugreifen, wird Distinguished Name (DN) genannt und bildet sich durch die Verkettung der RDNs aller direkten Vorgänger plus den RDN des aktuellen Objekts. Der DN ist im gesamten Verzeichnis eindeutig. Der RDN dagegen muss nur innerhalb der eigenen Hierarchie gegenüber den direkten Nachbarn eindeutig sein. Das Benennungsschema wird in *Abbildung 6* verdeutlicht.

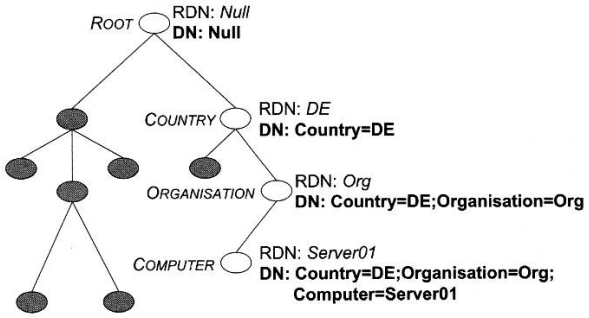

Abbildung 6 Namenskontext in X.500 [Klement 2003, S. 29]

Um dem Problem einer inkonsistenten Datenhaltung zu begegnen, werden Objekte, Datentypen, Strukturen und Regeln definiert. Die Gesamtheit dieser Definitionen wird als Schema bezeichnet, das dem Verzeichnis zugrunde liegt. Ein Schema umfasst Informationen über die Namenskonventionen, Strukturregeln der Datenbankhierarchie (Directory Information Tree, DIT), Objektklassen- und Attributtypregeln.

Die Namenskonventionen geben an, wie die DNs gebildet werden. Die DIT-Strukturregeln enthalten die Beziehungen, die zwischen Objekten bestehen können. Beispielsweise wird hier die Information festgehalten, dass ein Benutzer einer Abteilung untergeordnet werden kann, aber keinem Land oder gar einem Computer. Wichtig ist hierbei, dass die DIT-Strukturregeln nicht zwingend durchgesetzt werden, sondern Empfehlungen für die Struktur eines Verzeichnisses sind, um Fehler beim Anlegen von Objekten zu vermeiden.

Objektklassen enthalten Regeln, welche Attribute zu einem Objekt gehören und geben an, welche dieser Attribute obligatorisch und welche fakultativ sind. Objektklassen unterstützen das Konzept der Vererbung. Erbende Objektklassen enthalten die Attribute und Regeln ihrer Elternklassen. Tatsächlich nutzen in X.500 alle Objektklassen die Vererbung. Es gibt eine generelle Objektklasse *top*, von denen alle weiteren Objektklassen erben. *Abbildung 7* zeigt die Zuordnung von Attributen zu Objektklassen. Attributtypen regeln die Verwendung und den Dateninhalt von Attributen, den Grundelementen von Objekten. Ein Attribut kann je nach Zweck ein oder mehrere Werte aufnehmen, deren Syntax durch eine spezielle Sprache definiert wird.

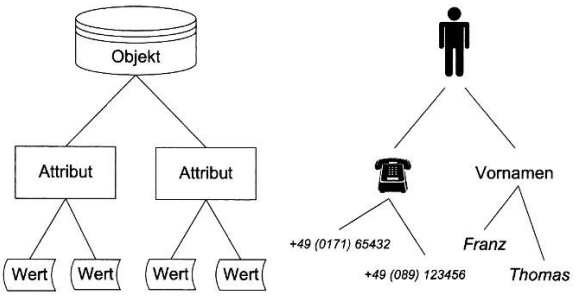

Abbildung 7 Objekte, Attribute und Werte in X.500 [Klement 2003, S. 26]

1.4.1.2 Lightweight Directory Access Protocol

Lightweight Directory Access Protocol (LDAP) ist eine vereinfachte Variante des Directory Access Protocol, das den Zugriff auf den X.500-Verzeichnisdienst definiert. DAP basiert direkt auf dem OSI-Protokoll-Stack und ist daher aufwändig zu implementieren. LDAP verwendet als Kommunikationsbasis den bewährten TCP/IP-Protokoll-Stack und ist als Protokoll in Schicht 7 zu finden.

Anfangs diente LDAP für den vereinfachten Zugriff auf den X.500-Verzeichnisdienst. Mittlerweile hat die Verbreitung von LDAP stark zugenommen und es werden vermehrt direkt eigenständige LDAP-Server eingesetzt, die ähnlich zu X.500 hierarchisch aufgebaut sind, aber durch das Fehlen von verteilten Systemoperationen wesentlich einfacher zu implementieren, aufzubauen und zu warten sind.

Ähnlich X.500 ist die Datenbank von LDAP hierarchisch aufgebaut. Die Namensgebung entspricht ebenfalls der von X.500 und Regeln für die Syntax von Objekten und Attributen lassen sich ebenfalls in einem Schema definieren.

Es sei hier nur festgehalten, dass LDAP ein Protokoll zur Kommunikation mit LDAP-Servern definiert, das bei weitem weniger komplex als das von X.500 ist. Durch diese Vereinfachung hat sich LDAP schnell verbreitet und wird in vielen Systemen eingesetzt. Weitere Informationen können [Sermersheim 2006] entnommen werden.

1.4.1.3 Active Directory

Das Active Directory (AD) ist der Verzeichnisdienst, der innerhalb von Microsoft Netzwerken verwendet wird. Jeder Domänencontroller (Domain Controller) führt ein AD. Das AD ist technologisch eine Kombination aus dem Domain Name System (DNS), X.500 und dem Lightweight Directory Access Protocol. AD ist kein reines X.500-Verzeichnis. Es verwendet allerdings sein Datenmodell, seine Namenskonvention und einige Konzepte.

Das Verzeichnis kann sich über mehrere Server erstrecken, um die Redundanz, Verfügbarkeit und Zugriffsgeschwindigkeit zu erhöhen. Für die Synchronisierung des Verzeichnisses über die verschiedenen Datenbanken hinweg ist der Replikationsdienst verantwortlich. Innerhalb einer Domäne sind alle Domänencontroller gleichberechtigt und führen demnach auch das gleiche Verzeichnis. Für die Synchronisierung dieser Verzeichnisse wird die Multimasterreplikation verwendet. Informationen zur Multimasterreplikation kann der Literatur [Klement 2003] entnommen werden.

Komponenten

Objekte im AD werden in einer relationalen Datenbank gespeichert. Diese Datenbank ist das eigentliche Verzeichnis eines AD. Objekte sind gemäß dem X.500-Standard aufgebaut und bestehen aus einer definierten Menge an Attributen. Diese Menge an Attributen wird auch Klasse genannt.

Die Definitionen, welche Attribute existieren, und Regeln, welche die Syntax und Gültigkeit von Klassen und Attributen im Verzeichnis beschreiben, werden in einem so genannten Schema gespeichert. Objekte müssen entsprechend dem Schema angelegt werden. Dadurch wird eine konsistente Speicherung innerhalb des Verzeichnisses sichergestellt.

Gewisse Attribute im Verzeichnis müssen besonders schnell abgefragt werden können. Dazu wird ein spezieller Index aufgebaut, der die Abfrage dieser Attribute beschleunigt. Dieser Index wird als Global Catalog bezeichnet. Der Global Catalog ist kein gesonderter Dienst, sondern eine zusätzliche Rolle, die Domänencontrollern zugewiesen werden kann.

Dieser Index enthält die Daten ausgewählter Attribute des gesamten Verzeichnisses, nicht nur der Domäne, der der Domänencontroller angehört. Der logische Aufbau eines Verzeichnisses wird in den nächsten Abschnitten näher gebracht. Der Global Catalog wird standardmäßig auf dem ersten Domänencontroller eines AD eingerichtet. Der Global Catalog ist wichtig für die Performanz des Verzeichnisses, wenn es über mehrere Standorte verteilt ist.

Das AD verfolgt ein Sicherheitskonzept, das auf Richtlinien (Policies) und Zugriffssteuerungslisten (Access Control Lists) basiert. Damit ist es möglich, für jedes Objekt im Verzeichnis und sogar auf einzelne Attribute Zugriffsrechte zu vergeben. Wer also zum Beispiel Objekte oder Attribute sehen, lesen oder ändern darf. Dieses Sicherheitskonzept ermöglicht die Delegierung von Administrationsfunktionen und die Vererbung von Rechten. Beide Eigenschaften können auf Container-Objekte im Verzeichnis angewandt werden. So ist die Weitergabe von administrativen Funktionen für bestimmte Bereiche des Verzeichnisses möglich, als auch die einfache Konfiguration von Zugriffsrechten für eine Vielzahl von Objekten innerhalb einer Hierarchie.

Domänen

Die Basiseinheit der Verwaltung im AD ist die Domäne. Eine Domäne wird automatisch bei der Einrichtung des ersten Domänencontrollers angelegt. Eine Domäne muss als Bezeichner einen gültigen DNS-Namen besitzen, der nicht zwangsläufig von der ICANN [ICANN] registriert sein muss. Domänen enthalten Objekte wie zum Beispiel Benutzer, Computer, Drucker und beliebige weitere Ressourcen, die in einem gemeinsamen Namenskontext zusammengefasst werden. Zu einer Domäne können mehrere Domänencontroller gehören, die alle das gesamte Verzeichnis dieser Domäne halten.

Alle Domänencontroller sind gleich. Manche allerdings haben Zusatzrollen, die so genannte Flexible Single Master Operations (FSMOs) übernehmen, deren Ausführung für das Verzeichnis sehr wichtig sind. Es gibt insgesamt fünf verschiedene FSMO-Arten, welche jeweils in einer Domäne nur von einem einzigen Domänencontroller gleichzeitig ausgeführt werden dürfen. Andersherum kann ein Domänencontroller allerdings mehrere FSMOs übernehmen.

Die fünf FSMOs sind Schema Master, RID Pool Master, Domain Names Master, PDC Emulator und Infrastructure Master. Zum Beispiel darf in einer Domäne nur ein einziger Domänencontroller Änderungen am Schema durchführen. Das Schema ist existentiell für das Verzeichnis und Konflikte im Schema müssen unbedingt vermieden werden.

Organisationseinheiten

Eine Organisationseinheit (Organizational Unit, OU) ist ein Container-Objekt im AD. Eine OU ist einer Domäne fest zugeordnet und kann daher nur Objekte dieser Domäne aufnehmen. OUs erlauben es, unabhängig von Domänen eine Hierarchie innerhalb einer Domäne abzubilden. So können beispielsweise innerhalb einer Domäne, die einen Unternehmensbereich darstellt, OUs für die Abbildung von Abteilungen verwendet werden.

Sinnvoll ist dies, da auf Basis von OUs, wie für alle Container-Objekte, die Administration delegiert werden kann. Somit ist es nicht nötig, einem Administrator für eine kleinere Abteilung Verwaltungsbefugnis über die gesamte Domäne zu geben, sondern nur für die OU, die seinen Kompetenzbereich darstellt. Durch Vererbung ist es weiterhin möglich, Gruppenrichtlinien und Rechte gleichzeitig allen Objekten innerhalb einer OU zuzuordnen.

OUs sind anders als Domänen nicht mit einem DNS-Namen verbunden. Während Domänen zwangsläufig einen DNS-Namen besitzen müssen, können OUs unabhängig von der Unternehmensstruktur und auch vom DNS erstellt werden.

Domänenstrukturen

Domänenstrukturen (Trees) bilden einen hierarchischen Zusammenschluss von einer oder mehreren Domänen, innerhalb eines kontinuierlichen Namenskontexts [Klement 2003]. Domänen innerhalb einer Domänenstruktur müssen sich ein gemeinsames Schema teilen und verwenden einen gemeinsamen Global Catalog. *Abbildung 8* zeigt die mögliche Verwendung von Domänenstrukturen.

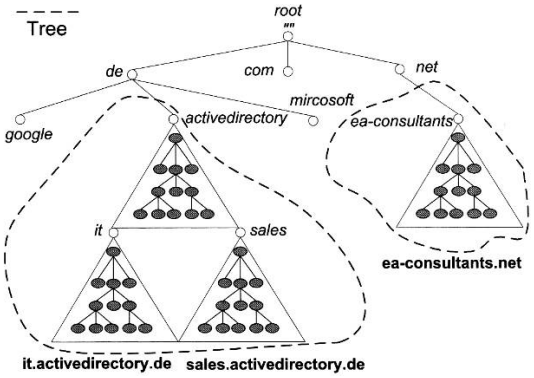

Abbildung 8 Trees im Active Directory [Klement 2003, S. 14]

Die Einrichtung der ersten Domäne ist gleichzeitig mit der Einrichtung einer neuen Domänenstruktur verbunden. Weitere Domänen können hierarchisch unterhalb der ersten Domäne erstellt werden. Sei der DNS-Name für eine Domäne *peterkirchner.de*, so kann z.B. eine untergegliederte Domäne für den Vertrieb *vertrieb.peterkirchner.de* erstellt werden.

Domänen sind abgeschlossene Verwaltungseinheiten. Innerhalb einer Domänenstruktur werden automatisch zwischen untergeordneten Domänen (also z.B. zwischen *peterkirchner.de* und *vertrieb.peterkirchner.de*) Vertrauensstellungen (Trusts) eingerichtet, aber nicht zwischen benachbarten Domänen, wie zum Beispiel bei den Domänen *vertrieb.peterkirchner.de* und *buchhaltung.peterkirchner.de* innerhalb einer Domänenstruktur.

Vertrauensstellungen in Domänenstrukturen verlaufen in beide Richtungen und sind transitiv. Das heißt, dass im vorherigen Beispiel zwar zwischen den beiden Domänen *vertrieb.peterkirchner.de* und *buchhaltung.peterkirchner.de* keine explizite Vertrauensstellung aufgebaut wurde, diese Domänen sich durch die Transitivität dennoch vertrauen. Das Prinzip der Vertrauensstellungen wird in *Abbildung 9* noch einmal dargestellt.

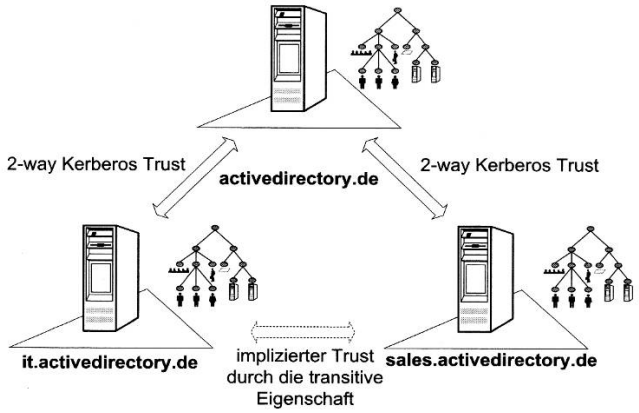

Abbildung 9 Vertrauensstellungen im Active Directory [Klement 2003, S. 15]

Gesamtstruktur

Die Gesamtstruktur (Forest) ist das gesamte Verzeichnis. Ein Forest kann aus nur einer Domäne oder auch aus mehreren Trees bestehen. Wie ein Tree hat ein Forest ein gemeinsames Schema und Global Catalog. Allerdings können Trees verschiedener Namensräume eingebun-

den werden. *Abbildung 10* verdeutlicht den Bezug der Gesamtstruktur zu den Domänenstrukturen. Mehr zur Planung von großen AD-Infrastrukturen ist [Klement 2003] zu entnehmen.

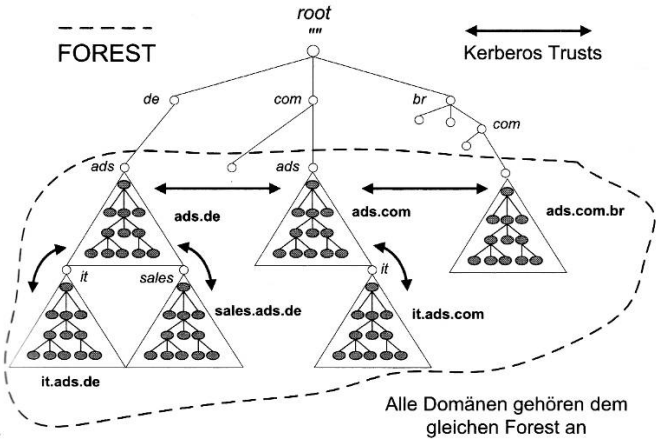

Abbildung 10 Forests im Active Directory [Klement 2003, S. 17]

Physischer Aufbau

Der logische Aufbau muss nicht den physischen Gegebenheiten des Netzwerkes entsprechen. Abteilungen können innerhalb eines Gebäudes liegen oder auch über verschiedene Länder verteilt sein. Unterscheiden sich die Verbindungen zwischen diesen Abteilungen dann ist ein wesentliches Merkmal die Geschwindigkeit des Netzwerks.

Um diesen Umständen gerecht zu werden, gibt es zusätzlich zur Konfiguration des logischen Aufbaus des Verzeichnisses, auch die Möglichkeit die physische Verteilung zu konfigurieren. Dazu gibt es eine AD-Verwaltungseinheit namens Standort.

Ein Standort definiert sich durch ein oder mehrere IP-Subnetze, die mit hoher Bandbreite miteinander verbunden sind. AD-Clients werden aufgrund ihrer IP-Adresse einem Standort zugeordnet. Innerhalb eines Standortes repliziert das AD nahezu in Echtzeit, wohingegen über Standortgrenzen hinweg eine deutliche längere Zeitspanne vergeht.

1.4.1.4 Active Directory in Application Mode

Active Directory in Application Mode (AD/AM) ist ein reiner LDAP-Verzeichnisdienst, der auf Windows Server 2003 und Windows XP installiert werden kann. Es können sogar mehrere Instanzen auf demselben Computer ausgeführt werden. AD/AM ist ähnlich dem AD aufgebaut, aber wesentlich schlanker. Es ist nicht wie das AD ins Betriebssystem integriert und kann auch keine sichere Authentifizierung wie Kerberos bereitstellen.

AD/AM bietet einige Vorteile gegenüber dem AD oder einem alternativen LDAP-Verzeichnisdienst [Klement 2003].

- AD/AM ist völlig ausreichend, um Applikationsdaten abzuspeichern. Es ist nicht nötig, das AD mit applikationsspezifischen Daten zu belasten, die nicht domänenweit verfügbar sein müssen.
- Ebenfalls ist es für applikationsspezifische Daten nicht notwendig, das Schema des unternehmensweiten AD zu verändern. AD/AM kann wie das AD benutzerdefinierte Schemata verwenden.
- AD/AM verwendet die Windows integrierte Sicherheit. Die Verwaltung und Berechtigungen können auf Benutzer der Domäne übertragen werden.
- AD/AM und AD sind technologisch und von der Bedienung weitgehend identisch. Dadurch muss das IT-Personal nicht zusätzlich geschult werden, wenn es bereits mit dem AD vertraut ist.

- AD/AM unterstützt ebenfalls die Replizierung. Statt applikationsspezifische Daten, die nur von bestimmten Anwendungen und Benutzern genutzt werden, unternehmensweit im AD zu replizieren, kann AD/AM auf einigen Servern installiert werden, die logisch unabhängig vom AD sind. Die Replikation zwischen diesen Servern kann genau konfiguriert werden. AD/AM verwendet wie das AD die Multimasterreplikation.

1.4.2 Active Directory Federation Services

Die Active Directory Federation Services (ADFS) [Microsoft TechNet 2005] bilden eine Schnittstelle zum AD, die es erlaubt, Authentifizierungen außerhalb des Intranets des AD durchzuführen. ADFS ist Teil der Windows Server 2003 R2 Produktfamilie [Microsoft Svr]. Der Aufbau einer Testumgebung, die ADFS einsetzt, wird im *Abschnitt 5.1 auf Seite 84* beschrieben. Die Verwaltung von ADFS wird über eine Management Konsole durchgeführt, wie sie in *Abbildung 11* zu sehen ist.

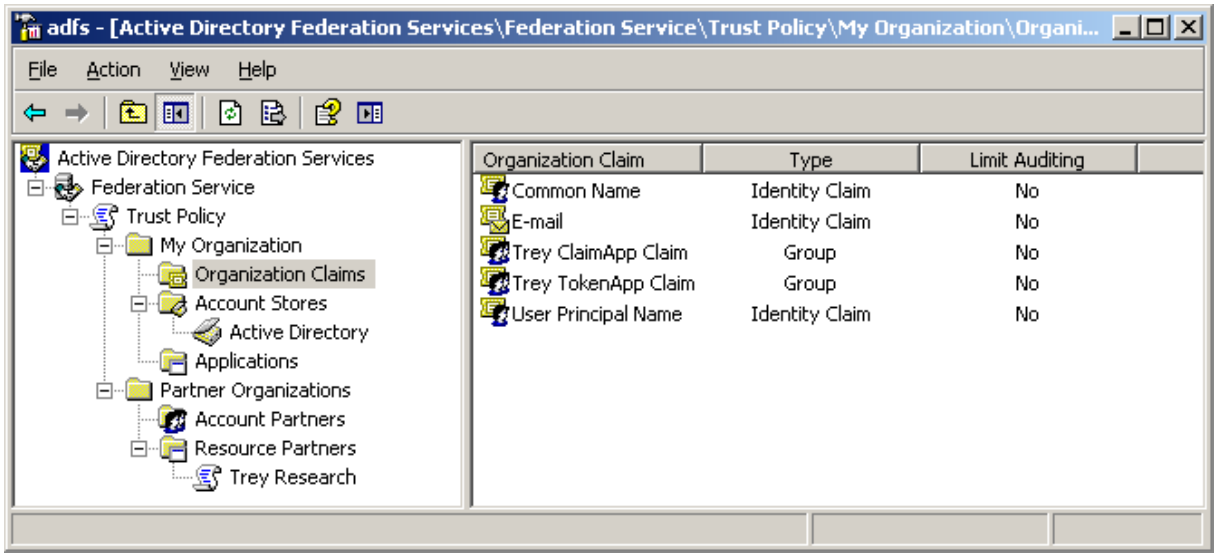

Abbildung 11 ADFS Verwaltung

ADFS ist ein Dienst, der SSO über das Web für mehrere Web-Anwendungen ermöglicht. Dabei werden den Web-Anwendungen Identität und Privilegien von Benutzern zur Verfügung gestellt. Welche Identitätsangaben und Privilegien dabei den Web-Anwendungen zur Verfügung gestellt werden, ist steuerbar.

Die übermittelten Privilegien werden in ADFS als Claims bezeichnet. Die Organisation, in der der Benutzer arbeitet, der auf eine Anwendung außerhalb der eigenen Organisation zugreifen möchte, wird als Account Partner bezeichnet. Die Organisation, die die Anwendung bereitstellt, ist der Resource Partner. Account Partner als auch der Resource Partner müssen ADFS verwenden, damit die gewünschte Kooperation zustande kommen kann.

Es ist zu betonen, dass ADFS weder eine Variation von .NET Passport, ein Verzeichnis bzw. eine Datenbank, eine Schema-Erweiterung des AD noch eine Art einer Domäne oder Vertrauensstellung des AD ist.

ADFS ist eng mit dem AD verbunden und verwendet es für die Abfrage von Benutzerattributen. Benutzer, die sich über ADFS authentifizieren, werden im Hintergrund von ADFS beim AD authentifiziert. Dabei kann ADFS gleichermaßen mit dem AD und AD/AM zusammen arbeiten. Unter Nutzung des AD können für die Authentifizierung Kerberos, X.509-Zertifikate und Smartcards genutzt werden. Unter Verwendung von AD/AM steht lediglich LDAP Bind zur Verfügung.

1.4.2.1 Partner in ADFS

Das Ziel von ADFS ist die Zusammenarbeit von Organisationen, die keine gemeinsame Benutzerdatenbank besitzen. Stattdessen wird mit ADFS eine föderierte Vertrauensstellung (Federated Trust) aufgebaut, die nur in eine Richtung verläuft und nicht transitiv ist (zu Vertrauensstellungen im AD siehe auch Abschnitt *Domänenstrukturen* auf Seite 12). ADFS unterscheidet zwischen dem Account Partner und dem Resource Partner.

Der Account Partner ist die Organisation, die die Benutzerdatenbank besitzt, welche entweder als AD oder AD/AM vorliegt. Der Account Partner ist dafür verantwortlich einen Benutzer zu authentifizieren, seine Claims zusammenzutragen und diese Informationen in einem Sicherheitstoken einzuschließen. Dieses Sicherheitstoken wird schließlich entlang der Federation Trust zum Resource Partner übermittelt, um auf eine Web-Anwendung zuzugreifen.

Der Resource Partner ist die Organisation, die Web-Ressourcen oder Web-Anwendungen anbietet. Sie vertraut gemäß der Federation Trust, dass der Account Partner die Authentifikation eines Benutzers vorgenommen hat. Der Resource Partner empfängt vom Account Partner ein Sicherheitstoken, das Claims (Aussagen zur Identität und zu Privilegien) über den aktuellen Benutzer enthält und verwendet diese für seine Autorisierungsentscheidungen. Web-Server des Resource Partner müssen den ADFS Web Agent betreiben, der in der Lage ist, die Informationen des Sicherheitstoken auszuwerten und an die Web-Anwendung weiterzugeben. Der Bezug zwischen Account Partner und Resource Partner wird in *Abbildung 12* dargestellt.

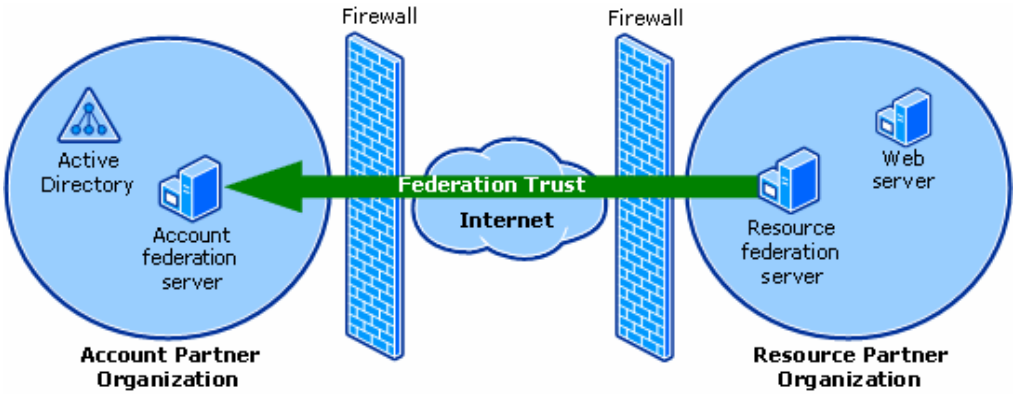

Abbildung 12 Federation Trust in ADFS [Microsoft TechNet 2005, S. 16]

Es gibt unterschiedliche Arten von Claims: vordefinierte und benutzerdefinierte. Die vordefinierten Claims sind UPN, E-Mail, Common Name und Group.

Die Web-Anwendungen können entweder auf die Verarbeitung von Claims vorbereitet sein oder existierende Web-Anwendungen sein, deren Sicherheitsmodell auf Windows basiert und Windows-NT-Sicherheitstoken erwarten. Der ADFS Web Agent kann in beiden Fällen die notwendigen Datenstrukturen bereitstellen.

1.4.2.2 Server Rollen & Komponenten

ADFS kennt drei Rollen, die während der Kommunikation zwischen einem Benutzer, der über ADFS authentifiziert, und einer Web-Anwendung in Aktion treten. Diese drei Rollen umfassen den Federation Server, den Federation Server Proxy und den Web-Server.

Der Federation Server betreibt den Federation Service. Dieser Dienst leitet Authentifizierungsanfragen von Benutzern an ein AD oder AD/AM. Zusätzlich ist auf dem Federation Server auch der Security Token Service (STS) zu finden. Der STS erstellt nach erfolgreicher Authentifikation Claims (vergleichbar mit Capabilities, siehe *Abschnitt 1.5 Autorisierung*), die an den Resource Partner weitergereicht werden.

Die Angaben, die in Claims gemacht werden, können frei konfiguriert werden. Dies passiert in so genannten Claim Mappings. Dadurch können Privilegien oder Rollenzugehörigkeiten vereinbart werden, die beide Seiten, also Account Partner und Resource Partner verstehen. Die Preisgabe der wahren Privilegien oder Rollenzugehörigkeiten, die eventuell aus Claim-Bezeichnungen des Account Partner abgeleitet werden könnten, kann dadurch vermieden werden. Auf Seiten des Resource Partner werden die Claims entsprechend der Claim Mappings in dort existierende Claims transformiert.

Federation Server Proxies dienen der Absicherung der Federation Server. Da der Federation Server ein Mitgliedsserver (ein Server, der einer Domäne im AD angehört) sein muss, ist es nicht empfehlenswert, dem Federation Server direkten Zugang zum Internet zu gestatten. Stattdessen führt man die Rolle der Federation Server Proxies ein, die in die DMZ des Firmennetzwerks positioniert werden können und deren Aufgabe nur in der Weiterleitung von Authentifizierungsanforderungen an die Federation Server besteht.

Web-Server, die für den Zugriff über ADFS konfiguriert sind, betreiben einen ADFS Web Agent, der die Verwaltung von Sicherheitstoken und Authentifikations-Cookies übernimmt. Der ADFS Web Agent wertet diese Informationen aus und übergibt sie in den Prozess, der für die Antwort des Web-Servers verantwortlich ist. Federation Server Proxies authentifizieren sich gegenüber den Federation Server mittels SSL-Client-Zertifikaten, deren öffentliche Schlüssel in den Richtlinien der Federation Server gespeichert werden und somit berechtigt sind, Weiterleitungen an diese Server durchzuführen.

1.4.2.3 Authentifizierung

Damit ADFS Benutzer mit Hilfe des AD authentifizieren kann, muss der Benutzername entweder im User Principal Name (UPN) Format (*user@domain.de*) vorliegen oder im Format des Security Account Manager (SAM) (*domain\user*). Der zu authentifizierende Benutzer muss nicht unbedingt ein Benutzerkonto in der Domäne besitzen, in der der Federation Server betrieben wird. Es reicht aus, wenn die beiden Domänen (die vom Federation Server und die des Benutzers) eine Vertrauensstellung besitzen.

Einem Federation Server können mehrere Benutzerdatenbanken (AD oder AD/AM) zugeordnet sein, die gemäß einer Prioritätsliste abgefragt werden. Das AD unterscheidet zwischen verbindlichen und unverbindlichen Misserfolgen (authoritative failure und non authoritative failure). Verbindliche Misserfolge sind fehlgeschlagene Authentifizierungen basierend auf Richtlinien. Beispiele für verbindliche Misserfolge sind gesperrte Benutzerkonten, abgelaufene Kennwörter und Beschränkungen der Anmeldezeiten. Bei einem verbindlichen Misserfolg beendet ADFS den Authentifizierungsprozess und der Benutzer ist nicht authentifiziert. Ist der Misserfolg nicht auf Richtlinien zurückzuführen, handelt es sich um einen unverbindlichen Misserfolg. Dieser Fall tritt zum Beispiel auf, wenn die angefragte Benutzerdatenbank den Benutzer nicht kennt. Im Falle eines unverbindlichen Misserfolgs fragt ADFS die nächste Benutzerdatenbank auf der Liste an.

Claims für E-Mail, Common Name und benutzerdefinierte Claims werden aus den Benutzerattributen gebildet, die ADFS per LDAP-Anfrage vom AD erhalten hat.

ADFS verwendet drei Typen von HTTP-Cookies: Authentifikations-Cookies, Account Partner Cookies, Abmelde-Cookies. HTTP-Cookies [Kristol 2000] sind kleine Dateien (in der Regel kleiner als 4 KB), die clientseitig von einem Internet-Browser gespeichert werden. Sie ermöglichen dem ansonsten statuslosen HTTP-Protokoll Statusinformationen mitzugeben. Es wird zwischen Sitzungs-Cookies und persistenten Cookies unterschieden [Microsoft 2003]. Sitzungs-Cookies werden nur im Speicher gehalten und gehen verloren, sobald die Internet-Browser-Anwendung beendet wird. Persistente Cookies werden auf die Festplatte geschrieben und sind auch verfügbar, nachdem der Internet-Browser beendet wurde.

Das Authentifikations-Cookie enthält in signierter Form die Claims über einen Benutzer und ermöglicht Web-SSO, da die Authentifikation bei Vorhandensein dieses Cookies nicht wiederholt werden muss. Bei dem Authentifikations-Cookie handelt es sich um ein Sitzungs-Cookie und es ist somit nur innerhalb einer Browsersitzung gültig. Das Cookie ist nicht verschlüsselt. Deswegen setzt ADFS voraus, dass alle Verbindungen mittels SSL/TLS verschlüsselt werden. Das Authentifikations-Cookie wird von einem Federation Server erstellt, nachdem die Authentifikation erfolgreich durchgeführt wurde.

Ein Federation Server auf der Seite des Resource Partner kann Beziehungen zu vielen Account Partner besitzen, die alle die Web-Anwendung des Resource Partner nutzen dürfen. Ein Benutzer muss daher bei einem solchen Resource Partner vor der eigentlichen Authentifikation auswählen, zu welchem Account Partner er gehört. Der zuständige Account Partner wird in diesem Kontext auch Realm genannt. Da ein Benutzer in der Regel nur einem Realm angehört, wird zur Vereinfachung ein Account Partner Cookie beim Benutzer erstellt. Dieses Cookie ist persistent und somit auch nach beendeten Browser-Sitzungen weiter verfügbar.

Ein Benutzer kann während einer Arbeitssitzung mehrere Web-Anwendungen bei unterschiedlichen Resource Partner nutzen. Um die Abmeldung von allen Sitzungen zu vereinfachen, wird bei der Abmeldung ein Abmelde-Cookie geschrieben, das alle Resource Partner veranlasst, alle Authentifikations-Cookies zu löschen.

1.4.2.4 Autorisierung

Bei ADFS kann zwischen zwei Varianten der Autorisierung unterschieden werden. Diese beiden Varianten unterscheiden sich in der Grundlage der Autorisierungsentscheidung, nämlich ob Web-Anwendungen in der Lage sind, Claims zu verarbeiten. Web-Anwendungen, die dazu in der Lage sind, werden als Claim-aware bezeichnet.

Solche, die dazu nicht in der Lage sind, werten auf herkömmliche Weise Windows-NT-Sicherheitstoken aus. Der ADFS Web Agent ist dafür verantwortlich, dass Claims in Windows-NT-Sicherheitstoken umgewandelt werden und so existierenden Anwendungen die Erreichbarkeit durch föderierte Nutzer ermöglicht wird.

Unabhängig, welche Variante genutzt wird, die Sicherheitsinformationen werden in Sicherheitstoken transportiert, die signiert werden, um die Echtheit und die Unversehrtheit sicherzustellen. Die Sicherheitstoken werden vom Account Partner signiert und sind vom Resource Partner überprüfbar. Öffentliche und private Zertifikate werden in ADFS in Anlehnung an die Funktion der Zertifikate Token-Signierungszertifikat (token-signing certificate) und Überprüfungszertifikat (validation certificate) genannt.

Nachdem die Claims aus dem Sicherheitstoken extrahiert wurden oder von ADFS ein Windows-NT-Sicherheitstoken erstellt wurde, wird dieses an die Web-Anwendung weitergegeben. Die eigentliche Zugriffskontrolle führt die Web-Anwendung selbst durch. ADFS übernimmt lediglich die Authentifizierung und die Übermittelung von Metadaten, die mit einem Benutzer verbunden sind. ADFS hat somit nach der Authentifizierung keinen weiteren Einfluss auf die Zugriffskontrolle.

1.4.3 Kerberos

Kerberos wurde am M.I.T. entwickelt und basiert auf dem Needham-Schroeder-Authentifizierungsprotokoll [Eckert 2002; Coulouris 2005; Tanenbaum 2003; Klement 2003]. Die aktuellste Version ist die Version 5 [Neuman 2005]. Kerberos wird in verteilten Umgebungen eingesetzt und sichert die Authentifizierung von Benutzern und Servern in einem Netzwerk. Kerberos ermöglicht die Einrichtung von vertraulichen Kommunikationskanälen unter Verwendung von geheimen Schlüsseln. Für die Autorisierung ist Kerberos nicht vorgesehen.

Eine Kerberos-Infrastruktur sieht zwei Server vor. Den Authentifizierungsserver (Authentication Server, AS) und einen Ticket-erteilenden Dienst (Ticket Granting Service, TGS). Tickets sind Nachweise der Identität. Ein Ticket ist immer nur für die Kommunikation zwischen zwei Teilnehmern gültig. Das heißt, mit einem Ticket von Benutzer A für den Dienst B kann der Benutzer A nur dem Dienst B seine Identität beweisen. Für einen Dienst C könnte dieses Ticket nicht verwendet werden.

Der AS authentifiziert einen Benutzer und übergibt ihm ein so genanntes Initialticket, das dem Benutzer die Kommunikation mit dem TGS ermöglicht. Der TGS stellt Tickets für die Kommunikation mit anderen Benutzern und Diensten aus. Authentifizierte Benutzer und Dienste werden in Kerberos als Principals bezeichnet. Die Identifizierung des Benutzers selbst kann auf unterschiedliche Weisen erfolgen, die von Kerberos nicht festgelegt werden. Benutzer können sich zum Beispiel mit Benutzername und Kennwort oder Smartcards identifizieren, Computer oder gar einzelne Prozesse identifizieren sich in der Regel mit Zertifikaten.

In Kerberos werden alle Nachrichten, bis auf die erste, mit geheimen Schlüsseln verschlüsselt. Die Schlüssel zwischen dem AS und einem Benutzer oder Dienst werden Masterschlüssel genannt und sind sonst keiner weiteren Partei bekannt [Eckert 2002]. Alle weiteren Schlüssel werden Sitzungsschlüssel genannt, da diese für die verschlüsselte Kommunikation zwischen zwei Principals eingesetzt werden. Ein Benutzer, der in den Besitz eines Masterschlüssels gelangt, kann sich als dieser Benutzer authentifizieren und erhält ohne Probleme Tickets vom TGS.

Im Folgenden wird etwas vereinfacht das Protokoll an einem Beispiel erklärt. Es gibt zwei Phasen. In der ersten Phase authentifiziert der Benutzer sich gegenüber dem AS. In der zweiten Phase fordert der Benutzer Tickets vom TGS an. Im Beispiel gebe es einen Benutzer *A*, der an einem Computer *C* arbeitet, und sich zu einem Dienst *D* verbinden möchte.

1.4.3.1 Phase 1: Authentication Server

In dieser Phase authentifiziert sich der Benutzer gegenüber dem AS und erhält das Initialticket, das er in der zweiten Phase benötigt.

1. Der Benutzer *A* möchte sich an seinem Computer anmelden. Dazu gibt er sein Benutzernamen und sein Kennwort ein. Sein Computer sendet im ersten Schritt im Klartext nur den Benutzernamen an den AS.
2. Der AS antwortet mit einer Nachricht, die mit dem geheimen Masterschlüssel $K_{A,AS}$ verschlüsselt ist. Diese Nachricht enthält zwei Elemente.
    a. Einen geheimen Sitzungsschlüssel $K_{A,TGS}$ für die Kommunikation mit dem TGS. Dieser Sitzungsschlüssel ist nur dem Benutzer *A*, also seinem Computer, und dem TGS bekannt. Somit kann kein anderer die Kommunikation verstehen.
    b. Ein Ticket $K_{AS,TGS}(A, K_{A,TGS})$, das auch Initialticket genannt wird. Das Tupel $(A, K_{A,TGS})$ ist mit dem geheimen Masterschlüssel $K_{AS,TGS}$ verschlüsselt und kann daher nur vom TGS entschlüsselt werden. Das Ticket enthält die Identität vom Benutzer *A* und den Sitzungsschlüssel $K_{A,TGS}$, den der TGS zur Verschlüsselung der Kommunikation mit *A* verwenden wird.
3. Der Benutzer *A* kann die Nachricht vom AS nur entschlüsseln, wenn er im Besitz des Masterschlüssels $K_{A,AS}$ ist.
    a. Bei Benutzern berechnet sich der Masterschlüssel aus dem Kennwort des Benutzers. Nachdem der Computer den Masterschlüssel aus dem Kennwort berechnet hat, besteht keine weitere Notwendigkeit, das Kennwort im Speicher

zu halten. Zur Erhöhung der Sicherheit kann der Benutzer auch einen Schlüssel verwenden, der beispielsweise auf einer Smartcard gespeichert ist.

b. Bei Diensten wurde der Masterschlüssel vorher vereinbart und zum Dienst übertragen. Wie diese Übertragung stattzufinden hat, schreibt das Kerberos-Protokoll nicht vor. Auf jeden Fall sollte der geheime Masterschlüssel nicht im Klartext über das Netzwerk übertragen werden.

4. Die Authentifikationsphase ist damit abgeschlossen. Der Benutzer ist nun im Besitz eines Sitzungsschlüssels für die Kommunikation mit dem TGS und einem Ticket, das die Identität vom Benutzer bestätigt.

   a. Der Benutzer hat zu diesem Zeitpunkt noch keinen Sitzungsschlüssel, um eine Kommunikation mit dem Dienst *D* aufzubauen. Erst in Phase 2 erhält der Benutzer diesen Schlüssel.

1.4.3.2 Phase 2. Ticket Granting Service

In dieser Phase fordert der Computer einen Sitzungsschlüssel für Kommunikation mit dem Dienst *D* und ein Ticket, das dem Dienst *D* die Identität vom Benutzer bestätigt, an.

5. Der Computer vom Benutzer fordert beim TGS einen Sitzungsschlüssel für die Kommunikation mit dem Dienst *D* an. Dabei werden an den TGS das Initialticket $K_{AS,TGS}(A, K_{A,TGS})$, der gewünschte Kommunikationspartner *D* und ein verschlüsselter Zeitstempel $K_{A,TGS}(t)$ übermittelt.
6. Der TGS antwortet mit der Rückgabe eines Sitzungsschlüssels $K_{A,TGS}(B, K_{A,D})$ für die Kommunikation zwischen *A* und *D* und einem Ticket $K_{D,TGS}(A, K_{A,D})$.
7. Damit ist die Ticket-Anforderung abgeschlossen.

   a. Benutzer *A* ist nun im Besitz des Sitzungsschlüssels $K_{A,D}$, mit dem eine verschlüsselte Kommunikation mit dem Dienst *D* möglich ist.

   b. Weiterhin hat der Benutzer das Ticket $K_{D,TGS}(A, K_{A,D})$, das dem Dienst *D* die Identität vom Benutzer bestätigt. Nur der Dienst *D* ist in der Lage, dieses Ticket zu entschlüsseln und somit die Identität von *A* und den Sitzungsschlüssel $K_{A,D}$ zu erhalten.

1.4.4 Single Sign-On

Kerberos ist eine Technologie, die Single Sign-On ermöglicht. Es findet einmalig eine Authentifizierung statt und der Benutzer erhält dafür sein Initialticket, das im weiteren Verlauf für weitere Kommunikationen mit Diensten als seine Identitätsbestätigung ausreicht. Das heißt, in dem Gültigkeitsbereich des Authorization Server wird echtes Single Sign-On verwirklicht. Der Gültigkeitsbereich eines Authorization Server wird als Realm bezeichnet und umfasst die Benutzer und Dienste, die der Authorization Server authentifizieren kann.

Aber auch außerhalb des eigenen Realm kann mit Kerberos SSO erreicht werden. Dazu kann ein Client ein so genanntes Remote Ticket beim TGS anfordern, das die Kommunikation zu einem fremden TGS erlaubt. Von da an folgt Kerberos dem bekannten Protokoll. Der Client benutzt dieses Remote Ticket, um eine Kommunikation zum fremden TGS aufzubauen und fragt dort nach einem Ticket für den Dienst im fremden Realm an. Dieses Prinzip wird Cross-Realm-Authentication genannt [Klement 2003]. In Kerberos Version 4 war es noch notwendig, dass innerhalb einer Hierarchie Vertrauensstellungen zu jedem TGS aufgebaut werden mussten, zu denen man vom eigenen Realm Kontakt aufnehmen wollte. Bei einer großen Hierarchie kann dies viele Vertrauensstellungen bedeuten, die eingerichtet werden müssen. Ab

Version 5 ist eine mehrstufige Cross-Realm-Authentication möglich und es sind nur noch Vertrauensstellungen zu Vorgänger und Nachfolger des eigenen Realm notwendig.

Abschnitt 1.4.2.3 hat gezeigt, dass auch ADFS SSO ermöglicht. Durch das Authentifizierungs-Cookie, das vom Federation Server bei der ersten Authentifikation hinterlegt wird, wird dem Benutzer erspart, sich selbst bei jedem Dienst aufs Neue auszuweisen. Dieses Authentifizierungs-Cookie ist der Nachweis, dass der Benutzer sich bereits authentifiziert hat. Das Authentifizierungs-Cookie vom Federation Server erfüllt somit den gleichen Zweck wie das Initialticket bei Kerberos, das vom Authentication Server ausgestellt wird.

Dieses Authentifizierungs-Cookie wird vom Federation Server des Account Partner genutzt, um die Identität und Attribute des Benutzers auszulesen, die in signierter Form in diesem Cookie gespeichert sind. Die in diesem Cookie gespeicherten Claims werden vom Federation Server des Account Partner nach dem für den Resource Partner definierten Mapping in ein neues Sicherheitstoken geschrieben, der zum Resource Partner übermittelt wird. Der Benutzer muss hierbei nicht in Aktion treten. Es ist zu beachten, dass die unterschiedlichen Authentifizierungs-Cookies alle beim Benutzer, genauer im Cookie-Cache des Browsers, vorliegen. Es ist wichtig und essentiell für das hier vorgestellte Protokoll, dass der Browser das Fremdauslesen von Cookies verbietet. Die Authentifizierungs-Cookies für die Resource Partner verbleiben im Browser, bis der Benutzer entweder seinen Browser schließt und damit seine Sitzung beendet oder bis der Benutzer seine Abmeldung einleitet, womit alle Resource Partner beauftragt werden, ihre Authentifikations-Cookies explizit zu löschen. Die globale Abmeldung wird durch das Abmelde-Cookie realisiert, das im *Abschnitt 1.4.2.3* angesprochen wurde.

1.5 Autorisierung

„Die Aufgaben der Rechteverwaltung und der Zugriffskontrolle eines IT-Systems bestehen darin, Mechanismen zur Vergabe von Zugriffsrechten bereitzustellen sowie bei Zugriffen auf zu schützende Objekte die Autorisierung zu prüfen.“ [Eckert 2002, S. 445]

Zur Unterscheidung zwischen Zugriffskontrolle und der Autorisierung trägt [Tanenbaum 2003, S. 503] bei. Die Überprüfung der Zugriffsrechte stellt die Zugriffskontrolle dar und die Erteilung der Zugriffsrechte die Autorisierung. Allerdings wird zugleich angemerkt, dass beide Begriffe oft synonym verwendet werden. In dieser Arbeit werden Grundlagen dargelegt, die zum Teil durch Sekundärliteratur gestützt werden, Softwareprodukte und existierende Arbeiten zum Thema analysiert. Im Rahmen dieser Ausführungen wird soweit möglich, die Terminologie der jeweiligen Urheber verwendet, wobei sich zeigt, dass diese beiden Begriffe in der Tat austauschbar verwendet werden. Soweit eine Differenzierung notwendig ist, wird auf die besondere Auslegung des Begriffs hingewiesen.

Der Zugriff auf ein Objekt ist definiert durch eine Menge an Operationen, die dieses Objekt anbietet. Die Überprüfung der Zugriffsrechte für jeweilige Operationen auf ein Objekt ist die Zugriffskontrolle und ist damit der Schutz eines Objekts vor einem Subjekt, das die Operation initiiert hat. Der Schutz selbst kann ein Programm sein, der je nach Typ der zu schützenden Objekte Teil des Betriebsystems sein kann oder auch nicht. So ein Programm wird auch als Referenz-Monitor bezeichnet.

1.5.1 Zugriffskontroll-Matrix

Eine Zugriffskontroll-Matrix ist ein Modell, worin die Spalten die zu schützenden Objekte und die Zeilen die zugreifenden Subjekte darstellen. In der Praxis wird dieses Modell nicht eins zu eins umgesetzt, da es eine ineffiziente Darstellung von Zugriffsrechten wäre. Auf bestimmte Objekte hat oft nur ein bestimmter Personenkreis Zugriffsrechte. Im Modell der Zugriffskontroll-Matrix gäbe es daher sehr viele freie Bereiche, da ein Subjekt in der Regel nur für den Zugriff auf eine geringe Untermenge aller zu schützenden Objekte autorisiert ist.

Abgeleitet von der Zugriffskontroll-Matrix gibt es zwei effizientere Ansätze, die heutzutage gebräuchlich sind. Die beiden Ansätze unterscheiden sich in der Art der Ableitung von der Zugriffskontroll-Matrix. Bei dem Konzept der Zugriffskontrolllisten (Access Control Lists, ACLs) wird pro Objekt (entspricht den Spalten im Modell der Zugriffskontroll-Matrix) eine Liste angelegt, die Paare von Subjekten und einer Menge Zugriffsrechten enthält. Jedes Subjekt, das autorisiert ist, Operationen eines Objekts auszuführen, erhält einen Eintrag in dieser Liste, wobei jeder Eintrag die Operationen aufführt, die dieses Subjekt ausführen darf. Der zweite Ansatz der Berechtigungen (Capabilities) setzt bei den Zeilen (den Subjekten) der Zugriffskontroll-Matrix an. Das heißt, Subjekte besitzen Listen, die Zugriffsrechte enthalten, für die das Subjekt autorisiert ist.

Die Zugriffskontrolle arbeitet unterschiedlich nach der Art des verwendeten Ansatzes. Werden ACLs eingesetzt, muss der Referenz-Monitor bei dem Zugriff auf ein Objekt die Zugriffsrechte des Subjekts ermitteln und auf dieser Grundlage entscheiden. Werden dagegen Capabilities eingesetzt, muss der Referenz-Monitor lediglich die Echtheit der übermittelten Capabilities überprüfen.

Ein besonderer Vorteil der Capabilities gegenüber den ACLs ist eine sehr einfache Delegierung von Rechten. Angenommen ein Benutzer *A* möchte Benutzer *B* für einige Zeit, gewisse Rechte übertragen, weil dieser zum Beispiel im Urlaub ist. Bei ACLs müssten in diesem Fall die ACLs für die betreffenden Objekte oder die Rollenzugehörigkeit von Benutzer *B* geändert werden. Die Änderung der ACLs kann sehr aufwändig sein, wenn es sich um viele Objekte handelt. Außerdem ist es schwierig sicherzustellen, dass die Rechte nur in einem bestimmten Zeitraum zugewiesen werden, denn die Rechte müssten zu einem ganz bestimmten Zeitpunkt gesetzt und zu einem ganz bestimmten Zeitpunkt wieder entzogen werden. Sofern es dafür keinen Automatismus gibt, kann dies sogar zu einem Sicherheitsrisiko führen, wenn vergessen wird, alle Rechte von jedem Objekt wieder zu entziehen. Die Änderung der Rollenzugehörigkeit ist dagegen wesentlich einfacher, birgt hier aber das gleiche Risiko, dass die Rollenzugehörig nur für einen bestimmten Zeitraum vorgenommen wird.

Die Zugriffskontrolle über Capabilities dagegen basiert nicht auf der Identität des Benutzers. Die Capabilities enthalten alle notwendigen Informationen, um eine Zugriffskontrolle durchzuführen, und können auch Zusatzinformationen enthalten, die den Zeitraum der Capabilities einschränken. Es ist daher wesentlich einfacher eine Capability, die in der Form einer Datei gespeichert werden kann, an jemanden weiterzugeben als die vorherige Methode auf Basis von ACLs.

1.5.2 Discretionary & Mandatory Access Control

Die Zuweisung von Zugriffsrechten (formal die Autorisierung) kann unterschieden werden in der Hinsicht, wer diese Zuweisung vornimmt. Zwei Konzepte stehen sich hier gegenüber.

Discretionary Access Control (DAC) ist „eine Zugriffskontrolle, bei der ein Benutzer festlegen kann, welche Rechte andere Benutzer für einen Zugriff auf seine Dateien erhalten“ [Fuhrberg 2001, S. 466]. Um diese Art der Autorisierung durchführen zu können, muss neben der ACL einer Ressource auch der Besitzer bzw. der Ersteller dieser Datei gespeichert werden [Ferraiolo 2003]. DAC wird wegen dieser notwendigen Zusatzinformation in der Literatur mit dem Eigentümer-Prinzip verbunden, da der Eigentümer einer Datei die Zugriffsrechte selbst festlegen kann. Wenn ein Mitarbeiter, der Eigentümer einer Datei mit vertraulichen Informationen ist, der Firma schaden möchte oder er sich der Vertraulichkeit seiner Daten nicht bewusst ist, kann dieses Prinzip ein erhebliches Sicherheitsrisiko darstellen.

Aus diesem Grund besteht auch die Möglichkeit, die Erteilung der Zugriffsrechte vollkommen dem System zu überlassen. Dieses Prinzip verkörpert Mandatory Access Control (MAC), bei dem das System aufgrund der Zugriffsrechte des Benutzers regelbasiert die Autorisierung

(Erteilung der Zugriffsrechte) für die erstellten Dokumente dieses Benutzers definiert. Die Zugriffsrechte des Benutzers können sich hierbei aus Rollenzugehörigkeiten oder durch separate Metadaten definieren, die die Freigabeklasse (clearance) des Benutzers beschreibt.

1.5.3 Role-Based Access Control (RBAC)

RBAC ist ein abstraktes und universelles Sicherheitsmodell. Das Ziel ist die Erleichterung der Verwaltung von Zugriffsrechten. Die Klassifizierung von RBAC sieht vier Modelle vor.

1. Core RBAC
2. Hierarchical RBAC
3. Static Constrained RBAC
4. Dynamic Constrained RBAC

Core RBAC enthält die Basisfunktionalität, die in allen RBAC-Systemen vorzufinden sind. Hierarchical RBAC unterstützt zusätzlich das Konzept einer Rollenhierarchie. Static Constrained RBAC fügt Bedingungen für die Zuordnung von Rollen hinzu und Dynamic Constrained RBAC erlaubt zusätzlich Bedingungen auf Basis von Benutzerattributen. RBAC ist ein sehr umfangreiches Modell. In diesem Abschnitt sollen die Grundprinzipien von Core RBAC kurz umrissen werden. Für eine weiterführende Lektüre ist [Ferraiolo 2003] zu empfehlen.

Sicherheitsrichtlinien in Core RBAC weisen fünf wesentliche Elemente auf: Benutzer, Rollen, Erlaubnisse, Operationen und Objekte. Wobei sich die Erlaubnisse sich aus Operationen, die Objekten zugeordnet sind, zusammensetzen. Es ist wichtig zu verstehen, dass in RBAC alle Sicherheitsrichtlinien auf Basis der Rollen verfasst werden. Erlaubnisse werden Rollen zugeordnet und Benutzer sind Mitglieder von Rollen. Benutzer werden Rollen aufgrund ihrer Kompetenzen, Vollmachten und Verantwortlichkeiten zugeordnet.

RBAC sieht vor, dass ein Benutzer bei der Ausführung von Operationen nur die für die Ausführung notwendigen Rollen annimmt. Daher ist es in RBAC möglich, dass ein Benutzer nur eine Teilmenge seiner verfügbaren Rollen aktivieren kann. Die Auswahl und Ausführung einer Operation ist in einer Sitzung (Session) gekapselt. Das heißt, der Benutzer kann gleichzeitig mehrere Sitzungen geöffnet haben und unterschiedliche Operationen mit einer jeweils unterschiedlichen Auswahl an Rollen ausführen.

Viele Softwaresysteme setzen bei der Zugriffskontrolle auf ACLs, die Zugriffsberechtigungen auf Basis von Benutzern oder Gruppen verzeichnen. Gruppen sind den Rollen von RBAC sehr ähnlich, da sie jeweils eine Menge an Benutzern repräsentieren. Allerdings muss bei RBAC differenziert werden, dass Rollen den Mittelpunkt der Modellierung von Sicherheitsrichtlinien darstellen. Rollen sind nicht nur eine Menge von Benutzern, sondern repräsentieren eine Menge von Eigenschaften, die begründen, warum Rollen bestimme Erlaubnisse erhalten und warum bestimmte Benutzer Mitglied dieser Rolle sind. In RBAC ist es Bedingung, dass Rollen mehreren Benutzern und Benutzer mehreren Rollen zugeordnet werden können.

1.6 Beispielszenario Prüfungsverwaltung

Anhand eines Beispielszenarios soll dargestellt werden, welche Rollen es bei der verteilten Autorisierung gibt und welche Möglichkeiten der Aufgabenteilung realisiert werden können. Zudem wird die Terminologie von ADFS (siehe hierzu *Abschnitt 1.4.2*) veranschaulicht und ein erster Einblick gegeben, welche Kommunikationswege existieren.

Teilnehmer in diesem Szenario sind ein Mitarbeiter, ein Institut und ein Prüfungsamt. Sie werden im Folgenden und in *Abbildung 13* näher erläutert.

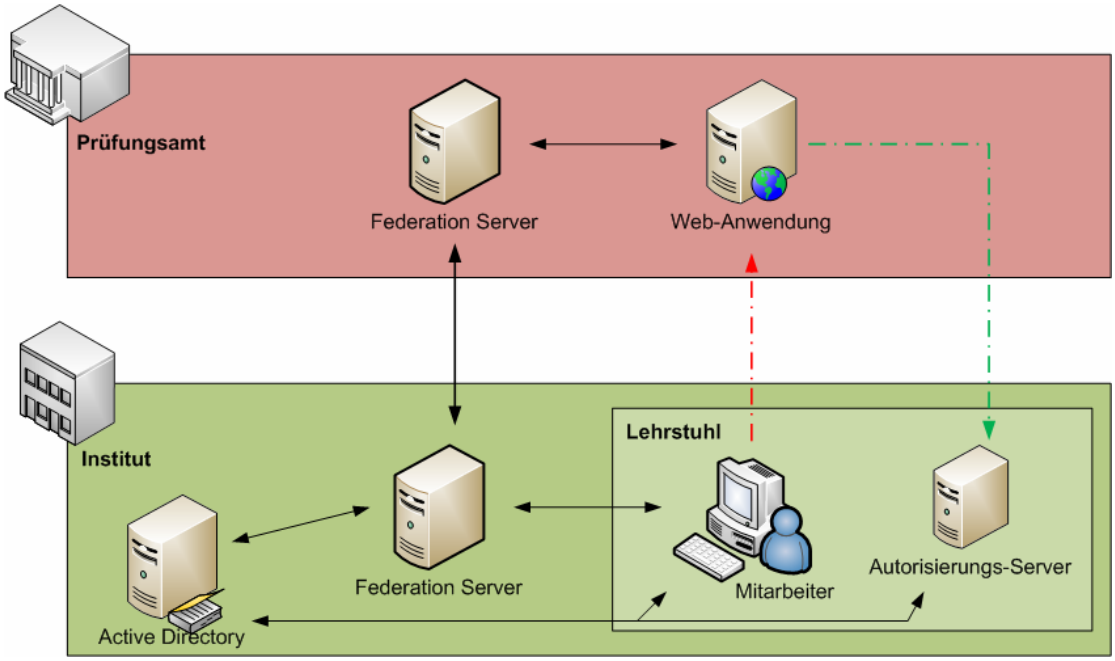

Abbildung 13 Szenario Prüfungsamt

Der Mitarbeiter darf Veranstaltungen (Vorlesungen, Seminare etc.), die in seinen Verantwortungsbereich fallen, verwalten. Mitarbeiter haben die Möglichkeit, Veranstaltungen anzulegen und Attribute für diese anzupassen. Folgende Aktionen und Attribute sind denkbar:

1. Ansicht vorhandener Veranstaltungen
2. Anlegen einer Veranstaltung
3. Ändern einer Veranstaltung
    a. Name und Beschreibung
    b. Typ
    c. Beleg- und Leistungspunkte
4. Freigeben oder Sperren einer Veranstaltung
5. Löschen von Veranstaltungen

Das Institut betreibt einen eigenen Verzeichnisdienst, der die zugehörigen Studenten, Mitarbeiter und Professoren verwaltet. Dazu werden den verwalteten Konten folgenden Rollen zugewiesen:

- Studenten
- studentische Hilfskräfte (SHK)
- wissenschaftliche Mitarbeiter
- technische Mitarbeiter
- Professoren
- Studiengang (Diplom Informatik, Bachelor Informatik, Master Informatik etc.)

Ein Konto kann gleichzeitig mehreren Rollen angehören. Weiterhin betreibt ein Lehrstuhl am Institut einen Autorisierungsserver, der Anfragen vom Prüfungsamt entgegennimmt und beantwortet. Die möglichen Aktionen und Attribute sind im Abschnitt Prüfungsamt aufgeführt. Der Autorisierungsserver am Institut protokolliert erfolgreiche und fehlgeschlagene Autorisierungsversuche.

Das Prüfungsamt stellt eine Web-Anwendung zur Verfügung, die dem Mitarbeiter ermöglicht, Veranstaltungen zu verwalten, für die sich Studenten einschreiben können.

Studenten haben die Möglichkeit, sich für Veranstaltungen einzuschreiben und unterschiedliche Aktionen auf diesen durchzuführen. Folgende Aktionen sind denkbar:

1. Ansicht der verfügbaren Veranstaltungen
2. Eintragen in verfügbare Veranstaltungen
3. Ansicht aller und aktueller Veranstaltungen
4. Austragen aus eingeschriebenen Veranstaltungen
5. Anmeldung zu Prüfungen
6. Abmeldung von Prüfungen
7. Ansicht und Eingabe von Prüfungsresultaten

Das Prüfungsamt ist dafür verantwortlich, die getätigten Aktionen nachvollziehbar und verbindlich zu protokollieren. Die verbindliche Protokollierung auf Seiten des Prüfungsamts in diesem Szenario ist nicht Teil des Autorisierungssystems, das in dieser Diplomarbeit erstellt wird.

2 Problemanalyse (State-of-the-Art)

Dieses Kapitel beschäftigt sich mit der Frage nach der Notwendigkeit der verteilten Autorisierung. Die Problemanalyse basiert auf drei Herangehensweisen.

1. Gegenüberstellung der lokalen und verteilten Autorisierung. Beide Konzepte werden in unterschiedlichen Aspekten verglichen und es werden die Vor- und Nachteile gegenübergestellt.
2. Analyse und Bewertung von vorhandenen Softwaresystemen, die lokale und verteilte Autorisierung anbieten.
3. Analyse und Bewertung von wissenschaftlichen Arbeiten, deren Thema die verteilte Autorisierung ist.

2.1 Gegenüberstellung lokale und verteilte Autorisierung

Ausgangspunkt für diesen Abschnitt ist eine Umgebung, in denen Benutzer mittels ADFS authentifiziert werden. Die folgenden Abschnitte zeigen Probleme bei der Verwendung der lokalen Autorisierung auf und geben gleichzeitig Ansätze mit, wie diese mittels verteilter Autorisierung gelöst werden können.

Zunächst werden die unterschiedlichen Workflows gezeigt, die bei der lokalen und verteilten Autorisierung Anwendung finden. Anschließend werden allgemeine Probleme der lokalen Autorisierung dargestellt, die durch die verteilte Autorisierung gelöst werden können. Abschließend wird eine Bilanz der Gegenüberstellung gezogen.

2.1.1 Workflows in SSO-Systemen mit lokaler Autorisierung

Bei der lokalen Autorisierung ergeben sich grundlegende Fragestellungen.

- Welche Workflows gibt es beim Account Partner bei der Einführung der möglichen Operationen beim Resource Partner?
- Und welche Workflows gibt es bei der Änderung dieser?

Dabei ist sicherlich der Aspekt der Änderungen (Update der Software) der wesentliche, da dieser wiederkehrend auftritt. Bei der lokalen Autorisierung kann man zunächst zwei Varianten unterscheiden.

1. Der Resource Partner gibt dem Account Partner einen festen Satz an SSO-Rollen.
2. Der Resource Partner lässt den Account Partner die SSO-Rollen selbst definieren.

Sehen wir uns beide Varianten genauer an. Die erste Variante (siehe auch *Abbildung 14*), die einen festen Satz an SSO-Rollen verwendet, beinhaltet die folgenden Prozesse und Dokumente.

1. Der Resource Partner erstellt oder ändert die Operationen in einer Anwendung, die er dem Account Partner zur Verfügung stellt.
2. Der Resource Partner übermittelt dem Account Partner die Liste mit SSO-Rollen, die der Resource Partner definiert hat.
3. Der Account Partner verbindet in seinem SSO-System die Domänen-Rollen mit den SSO-Rollen.
    a. Der Account Partner muss diesen Schritt wiederholen, wenn neue Rollen hinzugefügt werden oder sich das Aufgabenfeld von Rollen verändert.

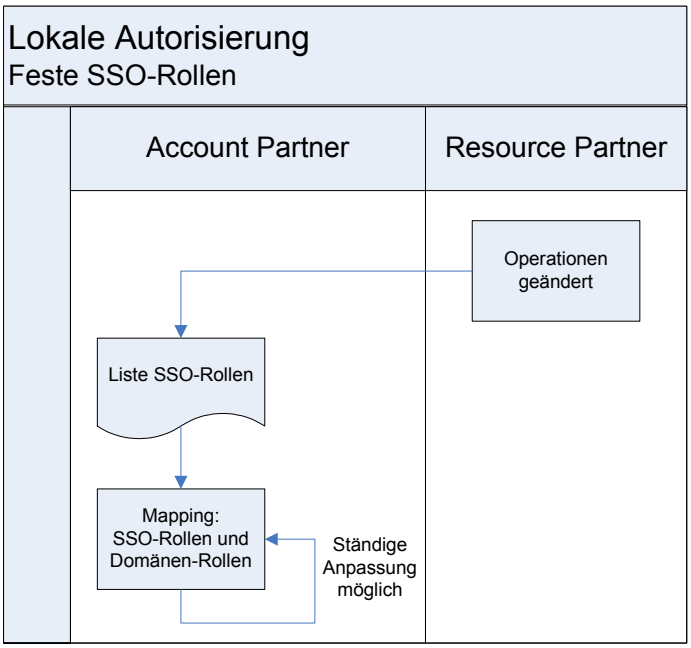

Abbildung 14 Lokale Autorisierung mit festen SSO-Rollen

Bei der zweiten Variante in *Abbildung 15* erhält der Account Partner eine Liste der verfügbaren Operationen und darf die SSO-Rollen selbst definieren. Diese Variante beinhaltet die folgenden Prozesse und Dokumente.

1. Der Resource Partner erstellt oder ändert die Operationen in einer Anwendung, die er dem Account Partner zur Verfügung stellt.
2. Der Resource Partner übermittelt dem Account Partner die Liste der Operationen für die Anwendung, die der Account Partner nutzt.
3. Basierend auf der Liste der Operationen definiert der Account Partner eine Liste mit SSO-Rollen, die der Account Partner benötigt.
    a. Der Account Partner verbindet in seinem SSO-System die Domänen-Rollen mit den SSO-Rollen.
        i. Der Account Partner muss diesen Schritt wiederholen, wenn neue Rollen hinzugefügt werden oder sich das Aufgabenfeld von Rollen verändert.
    b. Der Account Partner erstellt ein Mapping, das definiert welche SSO-Rollen für welche Operationen autorisiert sind.
    c. Der Account Partner übermittelt das Mapping aus Schritt 3b an den Resource Partner.
4. Der Resource Partner pflegt das Mapping, das er in Schritt 3c erhalten hat, in sein System ein.

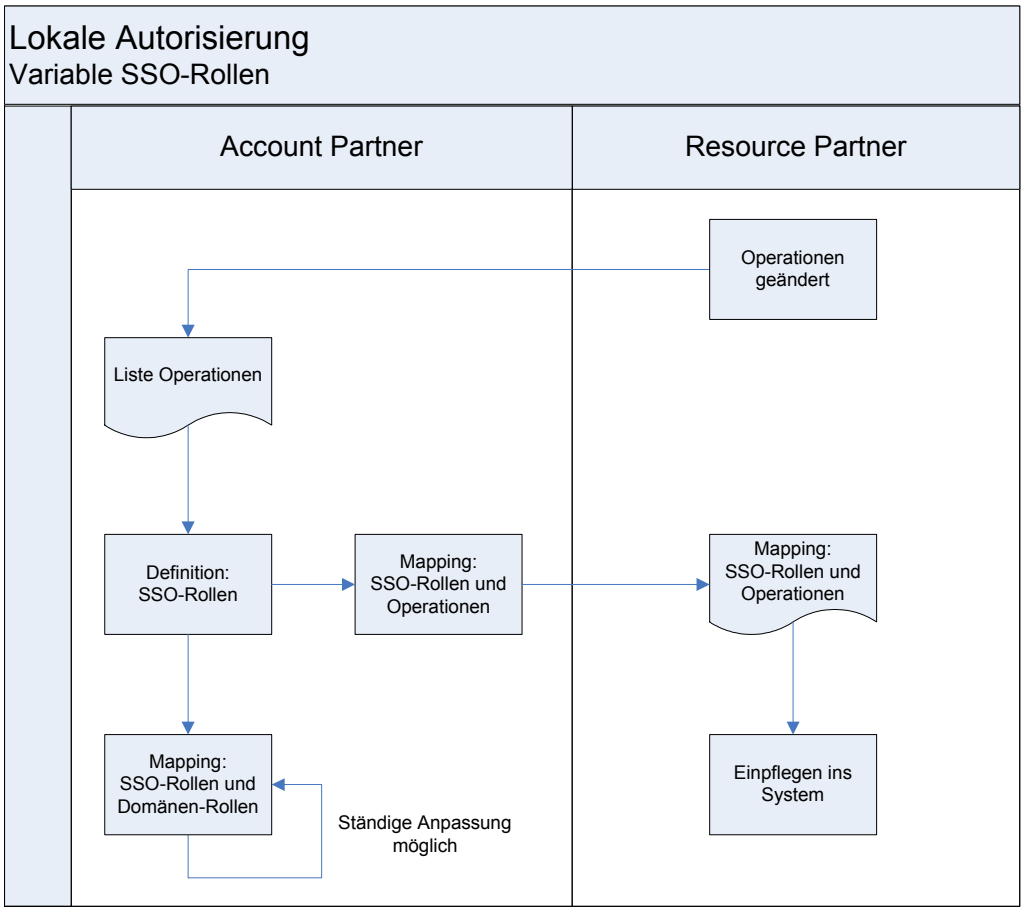

Abbildung 15 Lokale Autorisierung mit variablen SSO-Rollen

Tabelle 1 stellt die Vor- und Nachteile der beiden Varianten gegenüber.

Variante	Vorteile	Nachteile
1. Feste SSO-Rollen	• Der Account Partner hat keinen Aufwand für die Definition der SSO-Rollen. • Der Resource Partner hat wenig Verwaltungsaufwand.	• Der Account Partner hat keine direkte Möglichkeit, die SSO-Rollen an seine Bedürfnisse anzupassen. (Indirekt wäre dies über direkte Verhandlung mit dem Resource Partner möglich.)
2. Variable SSO-Rollen	• Der Account Partner kann die SSO-Rollen an seine Bedürfnisse anpassen.	• Der Account Partner hat Aufwand für die Definition der SSO-Rollen. • Der Resource Partner hat Aufwand mit dem Einpflegen der SSO-Rollen vom Account Partner. • Der Resource Partner muss die SSO-Rollen für jeden Account Partner verwalten.

Tabelle 1 Gegenüberstellung fester und variabler SSO-Rollen bei lokaler Autorisierung

Ein Problem, das beiden Varianten gemeinsam ist, ist der Umstellungszeitpunkt der Anwendung des Resource Partner. Ist ein Update der Operationen notwendig, bedeutet dies eine Änderung in der Anwendung. Die Account Partner werden über die Umstellung informiert und können entweder nach Variante 1 oder 2 handeln.

Es stellt sich allerdings die Frage, wie das Problem angegangen wird, dass die Account Partner das Mapping der SSO-Rollen zu den Domänen-Rollen anpassen (Variante 1 und 2) oder gar neue SSO-Rollen hinzufügen, vorhandene ändern oder neue hinzufügen (Variante 2)?

Die laufende Anwendung vom Resource Partner benötigt die SSO-Rollen und das Mapping vor der Umstellung. Mit dem Update der Anwendung beim Resource Partner ist jedoch eine zeitlich exakte Umstellung beim Account Partner notwendig. Mögliche Lösungen wären das Aufsetzen einer neuen Anwendung, die das Update beinhaltet. Die SSO-Rollen und das Mapping werden für diese neue Anwendung erstellt. Die Umstellung erfolgt hierbei, indem die Benutzer, die neue Anwendung verwenden und die alte abgeschaltet wird. Eine weitere Lösung wäre ein Wartungsfenster, in dem die Applikation updatet wird und die Account Partner Zeit haben, ihre SSO-Rollen und Mappings anzupassen. Die letzte Lösung ist mehr als schlecht, da hierbei ein Ausfall der Anwendung auftritt und zusätzlich bei vielen Account Partners die Wahrscheinlichkeit steigt, dass bei einem Probleme auftreten und den planmäßigen Start der neuen Anwendung in Gefahr bringt.

2.1.2 Workflows in SSO-Systemen mit verteilter Autorisierung

Betrachten wir nun die Prozesse bei der verteilten Autorisierung. Auch hier unterscheiden wir zwei Varianten. Unterschieden wird hierbei in der Verteilung der Rollen.

1. Account Partner und Access Control Partner sind zusammen. Das heißt, der Account Partner ist für die Autorisierung selbst verantwortlich.
2. Account Partner und Access Control Partner sind getrennt. Das bedeutet, der Account Partner ist nicht mehr für die Autorisierung verantwortlich, sondern eine dritte Partei.

Sehen wir uns Variante 1 in *Abbildung 16* genauer an.

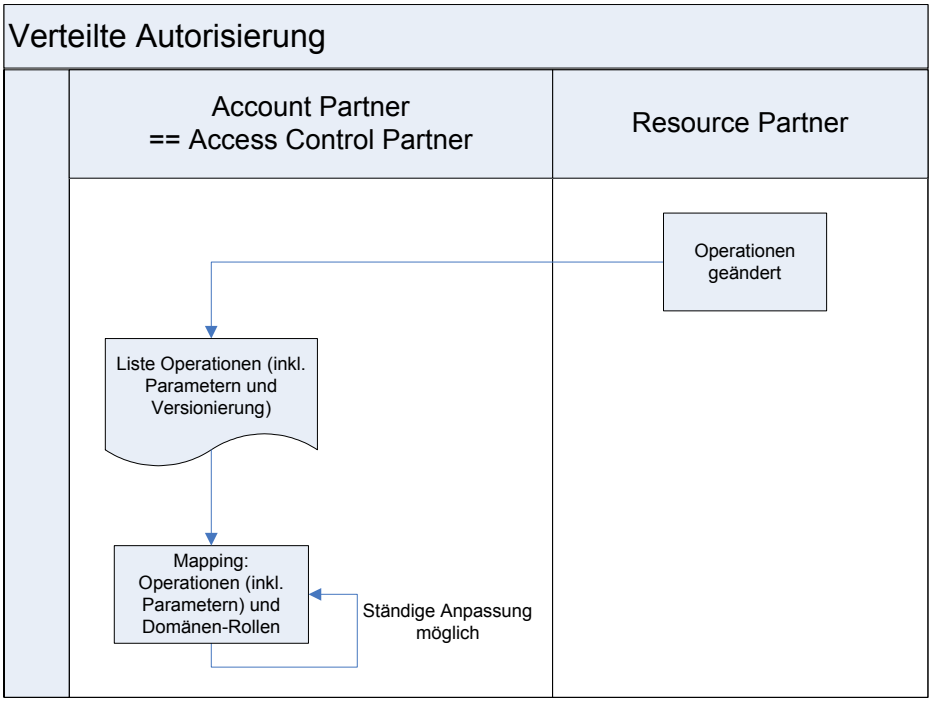

Abbildung 16 Verteilte Autorisierung mit zwei Parteien

1. Der Resource Partner erstellt oder ändert die Operationen in einer Anwendung, die er dem Account Partner zur Verfügung stellt.
2. Der Resource Partner übermittelt dem Access Control Partner (= Account Partner) die Liste der Operationen der Anwendung, die er dem Account Partner zur Verfügung

stellt. Diese Liste enthält neben den Operationen erweiterte Angaben zu Parametern der Operationen.

3. Der Access Control Partner weist den Operationen die Rollen zu, die autorisiert sind, die betreffenden Operationen auszuführen.
    a. Der Access Control Partner muss diesen Schritt wiederholen, wenn neue Rollen hinzugefügt werden oder sich das Aufgabenfeld der Rollen verändert.

Bei der zweiten Variante wird die Autorisierung ausgelagert und von einer dritten Partei übernommen. Die Anzahl der Prozesse ist gegenüber der 1. Variante größer, eröffnet allerdings auch neue Szenarien.

1. Der Resource Partner erstellt oder ändert die Operationen in einer Anwendung, die er dem Account Partner zur Verfügung stellt.
2. Der Resource Partner übermittelt dem Access Control Partner die Liste der Operationen der Anwendung, die er dem Account Partner zur Verfügung stellt. Diese Liste enthält neben den Operationen erweiterte Angaben zu Parametern der Operationen.
3. Der Access Control Partner definiert gemeinsame SSO-Rollen, die zur Authentifikation zwischen Account Partner und Access Control Partner verwendet werden.
    a. Der Access Control Partner weist den Operationen die SSO-Rollen zu, die autorisiert sind, die betreffenden Operationen auszuführen.
    b. Der ACP übermittelt die Liste der gemeinsamen SSO-Rollen an den Account Partner.
4. Der Account Partner verbindet in seinem SSO-System die Domänen-Rollen mit den SSO-Rollen.
    a. Der Account Partner muss diesen Schritt wiederholen, wenn neue Rollen hinzugefügt werden oder sich das Aufgabenfeld von Rollen verändert.

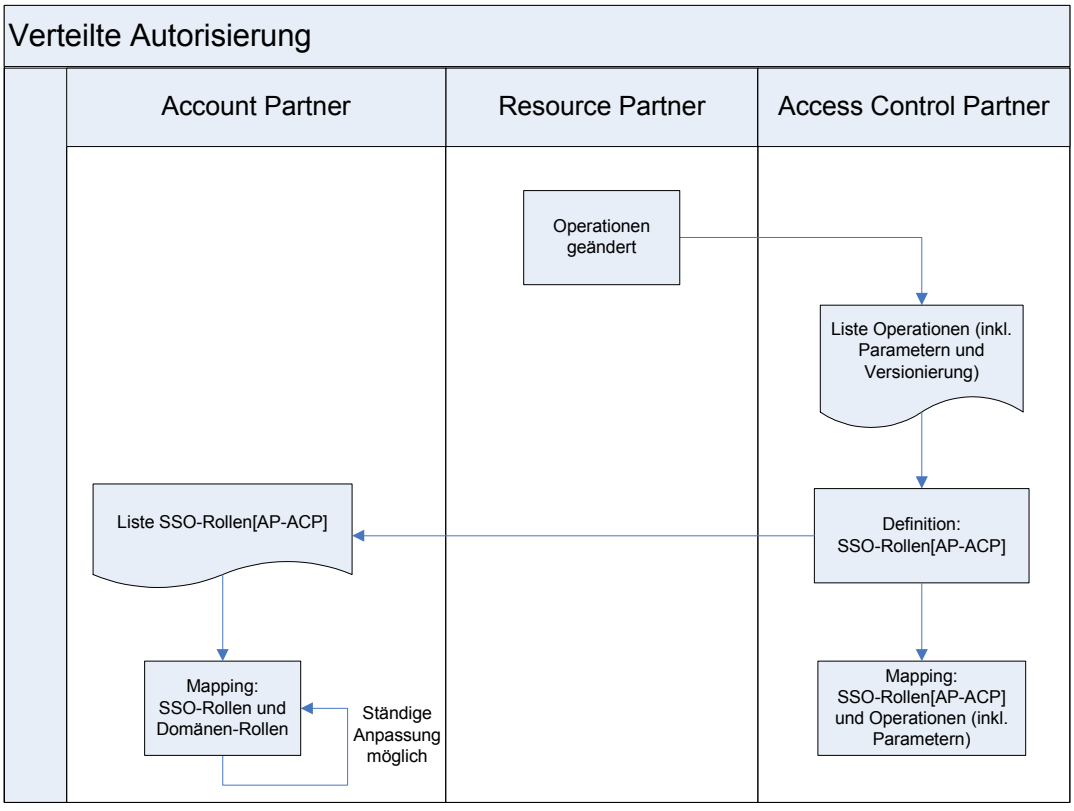

Abbildung 17 Verteilte Autorisierung mit drei Parteien

Die Autorisierung wird basierend auf diesen SSO-Rollen vorgenommen. Jede SSO-Rolle steht hierbei wie bei der lokalen Autorisierung für eine bestimmte Gruppe an Berechtigungen. Die dadurch entstandenen Probleme sind denen der lokalen Autorisierung ähnlich. Jedoch ist hier die Möglichkeit geschaffen worden, die Autorisierung an eine dritte Partei delegieren zu können.

Tabelle 2 stellt die Vor- und Nachteile der beiden Varianten der verteilten Autorisierung gegenüber.

Variante	Vorteile	Nachteile
1	• Der Account Partner kann Autorisierungsdetails selbst und feingranular verwalten. • Es sind keine SSO-Rollen für die Autorisierung notwendig. (Für die Authentifizierung beim Resource Partner weiterhin.)	• Der Account Partner muss einen Authorization Service betreiben, der die Autorisierungsanfragen entgegennimmt.
2	• Die Autorisierungsrichtlinien können von einer dritten Partei verwaltet werden.	• Autorisierungsmapping für SSO-Rollen beim Access Control Partner notwendig. • Definition von SSO-Rollen zwischen Account Partner und Access Control Partner notwendig. • Die Autorisierung ist grobgranularer als bei der 1. Variante. • Ähnlicher Aufwand wie bei der 2. Variante der lokalen Autorisierung. • Der Account Partner weiß nicht gewiss, welcher Nutzer vom Account Partner, welche Rechte besitzt und kann dies nur durch Rollenänderungen beeinflussen.

Tabelle 2 Gegenüberstellung zwei und drei Parteien bei verteilter Autorisierung

2.1.3 Rollen als Privilegien

In ADFS erhält der Resource Partner in der Regel Rolleninformationen über einen bestimmten Benutzer. Identitätsinformationen können auch übermittelt werden, jedoch werden diese meist nicht für Autorisierungsentscheidungen herangezogen, sondern nur die Rollenzugehörigkeit. Daher ist es bei der Abbildung von Privilegien für Benutzer notwendig, so viele Rollen zu definieren, wie Privilegien existieren. Hierbei kann zwar insofern optimiert werden, dass Privilegien für bestimmte Benutzergruppen zu Privilegiengruppen zusammengefasst werden, trotzdem entstehen in Abhängigkeit von der Komplexität der Anwendung sehr viele Rollen. Diese Rollen müssen sowohl auf Seiten des Account Partner und auf Seiten des Resource Partner angelegt und natürlich auch verwaltet werden.

Erschwert wird diese Situation, wenn Privilegien mit Parametern erstellt werden müssen. Das heißt, dass eine Bestellung zum Beispiel abhängig vom Bestellwert ist. Dies führt wiederum dazu, dass die Anzahl der zu definierenden Rollen erheblich steigt. Mit dem Anwachsen der

Anzahl der definierten Rollen steigen ebenso die Komplexität und der Verwaltungsaufwand, der sich schließlich in erhöhten Verwaltungskosten niederschlägt.

2.1.4 Dynamische Autorisierung

Mittels lokaler Autorisierung auf Seiten des Resource Partner sind keine Autorisierungen möglich, die abhängig von Parametern wie Identität, Rolle und Zeit auf Seiten des Account Partner sind. Diese so genannten dynamischen Autorisierungen müssen also aufgrund von Parametern zum Zeitpunkt der Autorisierungsanforderung (Autorisierungsnotwendigkeit) vom Account Partner durchgeführt werden.

Dabei darf nicht vergessen werden, dass verteilte Autorisierungen zweistufig erfolgen müssen. Der Resource Partner und der Account Partner müssen der Autorisierungsanfrage zustimmen, damit der Nutzer eine positive Autorisierungsantwort erhält. Das bedeutet zum Beispiel, dass ein Nutzer vom Account Partner autorisiert ist, eine Aktion durchzuführen, aber der Resource Partner dies momentan nicht erlauben kann.

2.1.5 Eingeschränkte Vertraulichkeit

Muss der Resource Partner die Autorisierung vornehmen (entspricht der lokalen Autorisierung), so muss der Account Partner dem Resource Partner ausreichend Informationen über den Nutzer übermitteln, damit der Resource Partner die Autorisierung durchführen kann. Es ist vorstellbar, dass dies für alle Parteien (Account Partner, Resource Partner und Nutzer) eine unerwünschte Weitergabe von vertraulichen Daten darstellt. Für den Account Partner und den Nutzer kann es insofern unerwünscht sein, da der Resource Partner erfährt, welche Person des Account Partner welche Aktionen durchführen darf und auch durchgeführt hat. Für den Resource Partner kann es ebenfalls ein Problem darstellen, da dem Resource Partner zusätzliche Sicherungsmaßnahmen zur Sicherung von Geheimnissen (wer hat wann was gemacht) aufgebürdet werden.

2.1.6 Fazit der Gegenüberstellung

Stellen wir nun die Vor- und Nachteile der Wartung bei lokaler und verteilter Autorisierung gegenüber.

Vorteile:

- Der Resource Partner hat bei der verteilten Autorisierung immer einen geringen Aufwand. Er übermittelt lediglich eine neue Liste an Operationen an den Access Control Partner.

Nachteile:

- Die Anwendung vom Resource Partner muss das Framework des verteilten Autorisierungssystems nutzen.
- Es gibt im Vergleich zur lokalen Autorisierung eine Leistungseinbuße aufgrund der notwendigen Netzwerkkommunikation.
- Ist der Access Control Partner nicht verfügbar, ist die Anwendung vom Resource Partner nicht oder nur eingeschränkt nutzbar.

2.2 Aktuelle Software: Microsoft Authorization Manager

Die Software Authorization Manager (AzMan) von Microsoft wird in der Windows Server-Variante ab der Version 2003 mitgeliefert. AzMan ist umfassendes Paket aus Diensten und einem Framework zur Erstellung von Anwendungen, die die Dienste von AzMan in Anspruch

nehmen. AzMan bietet verschiedene lokale Autorisierungsmöglichkeiten, die rollenbasiert sind. *Abbildung 18* zeigt die Verwaltungsoberfläche des Authorization Manager.

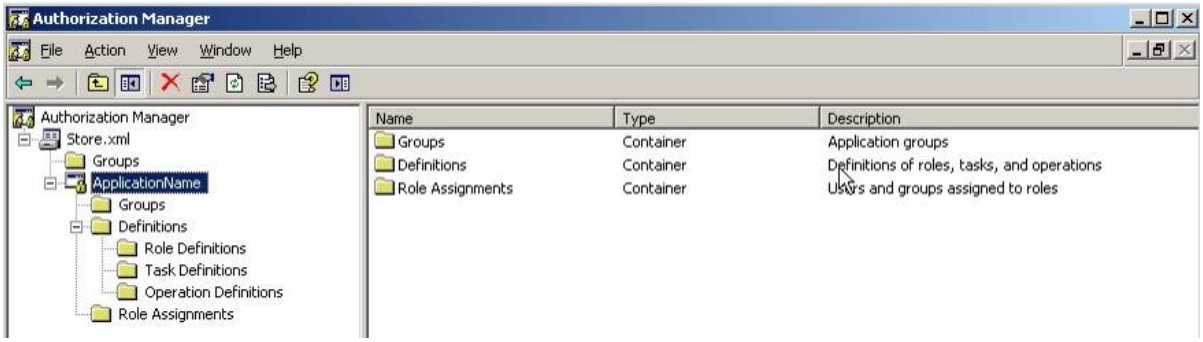

Abbildung 18 Microsoft Authorization Manager

2.2.1 Analyse

In AzMan werden einige Begriffe verwendet, die bereits Aufschluss über das Konzept dieser Software geben. In den folgenden Abschnitten werden diese Begriffe erläutert:

- Operationen und Tasks
- Application Groups und Rollen
- BizRules und Scopes
- Authorization Policy Store

2.2.1.1 Operationen und Tasks

Auf der niedrigsten Ebene der Definition von Autorisierungspunkten stehen die Operationen. Beispiele für solche Operationen können sein:

- SQL-Anfragen für die Ansicht von Bestellungen durchführen
- Ändern der Kontaktdetails von Kunden
- Ausführung bestimmter Methoden auf Implementierungsebene

Abbildung 19 zeigt die Verwaltung von Operation im Authorization Manager.

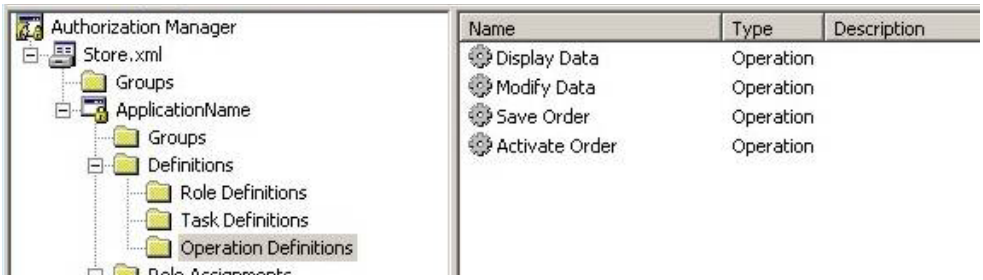

Abbildung 19 AzMan: Definition von Operationen

Operationen eigenen sich nicht für die Verwaltung durch Administratoren, da diese zu speziell und umfangreich sein können. Daher werden Operationen zu Tasks zusammengefasst, die Vorgänge auf einer höheren Ebene beschreiben. Beispiele für Tasks können sein:

- Kundendaten lesen und ändern
- Bestellungen erstellen
- Anwendungskonfiguration ansehen und ändern

Abbildung 20 zeigt die Verwaltung von Tasks im Authorization Manager.

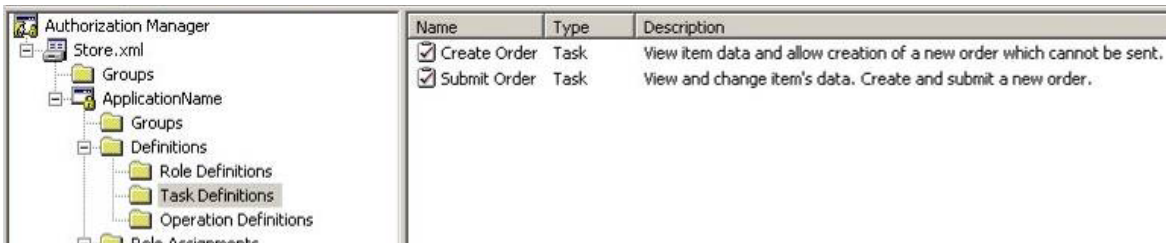

Abbildung 20 AzMan: Definition von Tasks

2.2.1.2 Application Groups und Rollen

Auf der Ebene der Tasks wird festgelegt, wer autorisiert ist, welche Vorgänge auszuführen. Diese Zuweisung basiert auf Application Groups, die Informationen darüber enthalten, wer zu der jeweiligen Gruppe gehört. Zugehörigkeit kann beim Authorization Manager auf unterschiedliche Weisen definiert werden:

- Windows Benutzerkonten und Windows Sicherheitsgruppen
- Benutzerdefinierte LDAP-Anfragen
- Application Groups

Die einfachste Zuweisung passiert aufgrund von Benutzerkonten und Sicherheitsgruppen. Diese können als zugehörig und auch als nicht-zugehörig (membership und non membership) definiert werden, wobei Nicht-Mitgliedschaften Vorrang vor Mitgliedschaften haben. Dies ist sinnvoll, wenn eine größere Sicherheitsgruppe berechtigt, aber eine Teilgruppe davon ausgeschlossen werden soll. *Abbildung 21* zeigt die Verwaltung von Gruppen im Authorization Manager.

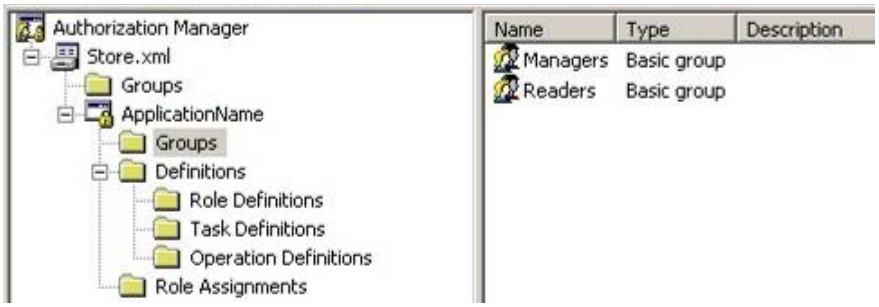

Abbildung 21 AzMan: Definition von Gruppen

Um dies zu verdeutlichen nehme man folgendes Beispiel an. In einer Firma werden interne Stellen ausgeschrieben, auf die sich alle Mitarbeiter dieser Firma bewerben können. Die Bewerbungen dieser Mitarbeiter gehen an die Personalabteilung, die die Bewerbungen prüfen müssen. Um die Gleichheit zu wahren, dürfen allerdings Mitarbeiter aus der Personalabteilung, die sich selbst am Bewerbungsverfahren beteiligt haben, die Bewerbungen der anderen Mitarbeiter nicht einsehen. Um diese Regelung umzusetzen bedarf es zweiter Gruppen. Die erste Gruppe besteht bereits im Unternehmen und umfasst die Mitarbeiter der Personalabteilung. Die zweite Gruppe umfasst all die Mitarbeiter, die sich am Bewerbungsverfahren beteiligt habe. Bei der Zugriffskontrolle auf die Bewerbungen werden diese beiden Gruppen verwendet. Bei der Definition der Zugriffsregel auf die Bewerbungen wird die erste Gruppe als Mitglieds-Gruppe und die zweite Gruppe als Nicht-Mitglieds-Gruppe definiert. Als Folge können alle Mitarbeiter der Personalabteilung, die sich nicht am Bewerbungsprozess beteiligt haben, die Bewerbungen einsehen.

Wesentlich komplexere Zugehörigkeitsdefinitionen können mit LDAP-Anfragen gegen das Active Directory gestellt werden. Hierbei können alle Attribute in der Anfrage verwendet werden, um auf diese Weise dynamische Mitgliedschaften herzustellen.

Beispiele:

- Es sei eine Ressource in einer Firma gegeben, die voraussetzt, dass der Nutzer volljährig ist, damit dieser auf diese zugreifen darf. Da die besagte Firma zum Teil auch Schüler einstellt, ist diese Art der Überprüfung notwendig. Für die Überprüfung könnte das Attribut *Geburtsdatum* aus dem Verzeichnisdienst der Firma verwendet werden, um den Zugriff auf die betreffende Ressource vom Alter abhängig zu machen.
- Es sei eine Ressource in einer Firma gegeben, die erst ab einer bestimmten Stufe der Führungsebene eingesehen werden darf. Über ein Attribut im Verzeichnisdienst, dass einen Wert über die Stufe in der Führungshierarchie enthält, kann sehr dynamisch der Zugriff bei Beförderungen ermöglicht werden.

Wäre es ohne diese Art der Mitgliedschaftsdefinition notwendig, dass Administratoren regelmäßig Anpassungen an den Gruppen vornehmen, ermöglichen LDAP-Anfragen hierbei eine Echtzeit-Mitgliedschaft basierend auf Attributen im Verzeichnisdienst.

Schließlich können Application Groups selbst wieder Application Groups enthalten und vereinfachen auf diese Weise die Administration, indem Application Groups für bestimmte Anwendungsfälle wiederverwendet werden können.

2.2.1.3 BizRules und Scopes

Der Authorization Manager besitzt zwei weitere Elemente, um Autorisierungen flexibel durchführen zu können: BizRules und Scopes.

BizRules sind Skripte, die in Echtzeit Parameter überprüfen können, die entweder mit der Autorisierungsanfrage mitgegeben wurden oder extern bezogen werden.

Beispiele für mitgegebene Parameter können sein:

- Betrag einer Bestellung, die der Benutzer tätigen möchte.
- Klassifizierung eines Dokumentes, auf das der Benutzer zugreifen möchte.
- Reiseklasse und Ziel einer Reisebuchung, die der Benutzer tätigen möchte.

Beispiele für externe Parameter:

- Tageszeit
- Auslastung des Servers
- Freier Festplattenspeicherplatz

Eine Kombination von mitgegebenen und externen Parametern wäre zum Beispiel eine Regel, dass ein Nutzer keine Personendaten von Angestellten ansehen darf, die in der Firmenhierarchie höher angestellt sind als der Nutzer. Der mitgegebene Parameter ist die Hierarchiestufe der Personendaten, die angezeigt werden sollen, der externe Parameter die eigene Hierarchiestufe. Angaben zum Subjekt sollten allerdings über Rollen statt über externe Parameter definiert werden. Beispielsweise sollte die Freigabeklasse (clearance), die den Zugriff auf klassifizierte Dokumente beschränkt, nicht über externe Parameter definiert werden, sondern über die Zuweisung einer entsprechenden Rolle an den Benutzer.

Scopes erlauben die Differenzierung von Berechtigungen innerhalb der Anwendung. Sie sind Parametern ähnlich, da er frei interpretiert werden darf. Die Auswertung des Scopes erfolgt allerdings nicht in einem Skript, sondern wird direkt in AzMan für die Zuweisung von Rollen verwendet.

Beispiel: Es sei der Fall gegeben, dass Büromittelbestellungen in Abteilungen jeweils nur durch die Team-Assistentin bzw. Sekretärin durchgeführt werden dürfen. Hierbei kann der Scope mit der Bezeichnung der Abteilung belegt werden und in AzMan wird dem Task „Bü-

romittelbestellung“ unter der Differenzierung des Scope (der Abteilung) der entsprechende Benutzer (die Team-Assistentin) zugewiesen.

2.2.1.4 Authorization Policy Store

Die Definitionen für die Anwendungen und die Zuweisungen von Berechtigungen kann der Authorization Manager entweder in XML-Dateien oder im Active Directory abspeichern. Die Speicherung als XML bietet sich an, wenn kein Active Directory zur Verfügung steht.

Die Speicherung der Richtlinien im Active Directory bringt einige Vorteile mit sich. Dazu gehören:

- Verfügbarkeit und Replizierung
- Delegierung

Der erste Punkt resultiert daraus, dass das Active Directory über alle Domänencontroller repliziert wird und dadurch die Verfügbarkeit erhöht wird. Der zweite Punkt resultiert aus der Eigenschaft des Active Directory Zweige im Verzeichnisdienst an Benutzer zur Delegation zu vergeben.

2.2.2 Bewertung

Der Authorization Manager erlaubt eine klare Trennung zwischen der Definition der Rollen, der Operationen und dem Mapping dieser beiden Gruppen. Die Autorisierung ist sehr flexibel, da Rollen zu Operationen und Tasks beliebig zugewiesen werden können. Die Verwendung von eigenen Skripten, den BizRules, die in VB Skript oder Java Skript geschrieben werden können, erweitert die Funktionalität des Authorization Manager um eigene Bedürfnisse. Zudem können durch BizRules Autorisierungen von Parametern abhängig gemacht werden, die auf zeitlichen, systembedingten oder von der Anwendung übergebenen Werten abhängig sind.

Die Verwendungsmöglichkeit von LDAP-Anfragen, um Rollen zu definieren, ist ein sehr interessanter Ansatz. Statt bei der Zugriffskontrolle dynamische Parameter zu verwenden, kommt der Ansatz der LDAP-Anfragen aus der entgegengesetzten Richtung. Rollen sind dynamisch und werden aktuell zum Zeitpunkt der Zugriffskontrolle durch die LDAP-Anfrage überprüft. Beide Verfahren, Parameter bei der Zugriffskontrolle und dynamische Rollen durch LDAP-Anfragen, können kombiniert werden.

Die Nutzung des Active Directory ist für den Authorization Manager ein erheblicher Zugewinn, da dadurch Replikation und Administrations-Delegation ermöglicht wird. Die Nutzung von reinen XML-Dateien wird auch angeboten und erlaubt so die Nutzung ohne ein Active Directory, allerdings ohne die Funktionen der Replikation und Delegation.

Die Schnittstelle des Authorization Manager, die im Client im Code verwendet werden soll, ist einfach gehalten, allerdings durch die Verwendung von Zahlenwerten zur Identifizierung auch fehleranfällig. Wenn Operationen mit aufsteigenden Zahlenwerten benannt werden und im Code eine falsche Operationskennziffer angegeben wird, können sich unbemerkt Fehler einschleichen. Die Verwendung von Zeichenketten oder GUIDs könnte dieses Risiko reduzieren, da es bei einer fehlerhaften Eingabe eines Bezeichners wahrscheinlicher wäre, dass dieser nicht existiert und von der Anwendung ein Fehler geworfen wird. Generell ist es empfehlenswert im Programmcode sprechende Bezeichner für die zu autorisierenden Operationen zu verwenden, damit der Code lesbar bleibt und Fehler schneller entdeckt werden können.

2.3 Aktuelle Arbeiten

Die folgenden Arbeiten sind eine Auswahl einer breiten Palette zum Thema Autorisierung insbesondere in verteilten Umgebungen. Sie werfen Problemstellungen auf, stellen Konzepte und Lösungsansätze vor.

Jeder der folgenden Abschnitte ist gegliedert in einer neutralen Darstellung der einzelnen Arbeiten, der eine Bewertung folgt, wie die einzelnen Konzepte und Lösungsansätze in der aktuellen Arbeit Verwendung finden können.

2.3.1 A Framework for Distributed Authorization

2.3.1.1 Analyse

In [Woo 1993] sind frühe Überlegungen zur Absicherungen von verteilten Diensten zu finden. Es werden Konzepte und ein Framework dargestellt, die die Thematik von der Authentifizierung bis zur Autorisierung umfasst. Das Ziel ist, generelle Sicherheitsfragen, die Dienste betreffen, zu beantworten. Es werden vier grundlegende Problemkreise angesprochen: Authentifikation, Autorisierung, Abrechnung und Protokollierung. Besonderes Interesse in dieser Arbeit gilt der verteilten Autorisierung.

Nach [Woo 1993] gibt es zwei Schlüsselprobleme: Repräsentation und Protokolldesign. Die Repräsentation betrifft das Problem, eine geeignete, repräsentative Abstraktion für Autorisierungsrichtlinien zu finden. Das Protokolldesign ist der Entwurf eines geeigneten Protokolls für die Clients, Autorisierungsserver und Dienstserver. Die Lösungsansätze für diese beiden Schlüsselprobleme lauten für die Repräsentation Generalized Access Control List (GACL) und für das Protokoll Authenticated Delegation.

GACL ist eine Erweiterung der bekannten Access Control Lists (ACLs), die um einige Eigenschaften erweitert wurden. Die wichtigsten innerhalb der Beschreibungssprache für GACL sind: strukturierte Eigenschaften, Vererbung, Regeln und die Angabe von Standardwerten. Ebenfalls ist es in GACL möglich, Autorisierungen neben der Rückgabe von den beiden Standardergebnissen „Zugriff genehmigt“ und „Zugriff verweigert“ einen Fehler zurückzugeben, wenn keine Entscheidung getroffen werden kann. Ein „Zugriff verweigert“ ist bei einer Fehlerbedingung („denial by default“) aber dennoch möglich. Weitere Flexibilität und Konfigurationsmittel bieten die Deklarationssektionen, die zum Beispiel anonyme Autorisierung erlauben oder angeben, ob Regeln sequentiell oder parallel auszuwerten sind. GACL wurde mit dem Ziel entwickelt, unabhängig von jeglicher Implementierungssemantik zu sein, um Interoperabilität zwischen verschiedenen Systemen zu gewährleisten.

Hinter dem Konzept des Authenticated Delegation verbirgt sich ein für die heutige Zeit vertrautes Prinzip, für die Kommunikation sichere Kanäle zu verwenden, die die Verbindlichkeit (durch gegenseitige Authentifikation der Server), Vertraulichkeit und Integrität wahren. Anwendung findet dieses Prinzip in zwei Protokollen.

1. Im Vertragsprotokoll (contracting protocol) zwischen einem Dienst- und Autorisierungsserver.
2. Im Protokoll zwischen einem Client und einem Autorisierungsserver.

Architektur

Das Framework sieht fünf Servertypen vor, wie sie in *Abbildung 22* dargestellt sind.

- Service Locator: Eine Art Verzeichnisdienst, der Informationen über Ort und Funktion der anderen Servertypen besitzt.

- Authentication Server: Der Autorisierungsserver übernimmt die Funktion, einen Client zu authentifizieren und als Bestätigung der Authentifizierung dem Client ein Authentifizierungszertifikat auszustellen.
- Authorization Server: Der Autorisierungsserver liefert Autorisierungszertifikate an authentifizierte Clients. Diese Autorisierungszertifikate werden vom Client später an einen Dienstserver weitergereicht.
- Group Server: Der Gruppenserver verwaltet Gruppenzugehörigkeiten für Benutzer und stellt Mitglieds- und Nicht-Mitgliedszertifikate (membership und non membership) aus. Dieser Server hat eine wichtige Funktion im Zusammenhang mit GACL, wenn Entscheidungen aufgrund von Gruppenzugehörigkeit getroffen werden müssen.
- System Monitor: Die Systemüberwachung verfolgt Systemparameter, die ebenfalls Teil von Entscheidungsregeln sein können. Beispielsweise können Autorisierungen verweigert werden, wenn die Systemlast ein festgesetztes Limit überschreitet.

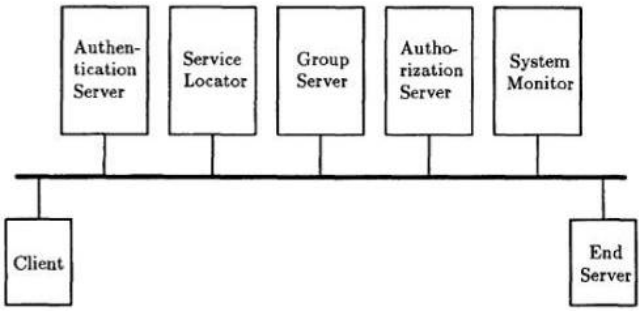

Abbildung 22 Distributed Authorization Framework nach [Woo 1993, S. 116]

Operationen und Protokolle

Betrachten wir nun zwei grundlegende Operationen.

1. Start eines Dienstservers
2. Nutzung eines Dienstes

Wenn ein Dienstserver *E* startet, der seine Autorisierung auslagern möchte, kontaktiert dieser einen Service Locator und fragt diesen nach einem Autorisierungsserver. Der Service Locator gibt *E* in der Antwort einen Autorisierungsserver *A* an. Im nächsten Schritt geht *E* einen Vertrag mit *A* ein, dass *A* die Autorisierung von *E* übernimmt. Dazu wird ein sicherer Kommunikationskanal zwischen *E* und *A* aufgebaut, wobei sich *E* und *A* gegenseitig authentifizieren. Über diesen Kommunikationskanal wird ein so genannter Delegierungsschlüssel (delegation key) *kd* vereinbart, der von *A* verwendet wird, um Autorisierungszertifikate zu signieren. *E* verwendet *kd*, um die Echtheit der Autorisierungszertifikate zu überprüfen. Schließlich wird der Vertragsaufbau mit der Übermittlung von Autorisierungsrichtlinien (authorization specification) von *E* an *A* abgeschlossen. Die Autorisierungsrichtlinien, die im Format GACL vorliegen, werden schließlich von *A* verwendet, um Autorisierungsentscheidungen für *E* zu treffen. Nach dem Vertragsaufbau registriert *E* im Service Locator *A* als seinen Autorisierungsserver.

Möchte nun ein Client *C* den Dienst von *E* nutzen, so fragt *C* zuerst den Service Locater an, welcher Autorisierungsserver für *E* verantwortlich ist. Nach Erhalt dieser Information fordert *C* ein Autorisierungszertifikat von *A* an. Der Client *C* ist schließlich in der Lage, den Dienst von *E* unter Vorlage dieses Autorisierungszertifikat zu nutzen.

2.3.1.2 Bewertung

Die angesprochenen Schlüsselprobleme waren Probleme damaliger Zeit. Repräsentation und Protokolldesign werden heutzutage mit Technologien wie Web Services und diversen XML-

Standards beantwortet. Gerade im Bereich der Kommunikation haben sich derzeit zustandslose Protokolle wie HTTP durchgesetzt, die als Basis für SOAP und darauf aufbauende Web Services.

Der Lösungsansatz von [Woo 1993] für die Repräsentation GACL ist sehr mächtig. Allerdings liegen in dieser Mächtigkeit auch die Nachteile. Durch die Möglichkeit jeder Ressource umfangreiche Autorisierungsrichtlinien mitzugeben, die über mehrere Stufen vererbt werden können, zusätzlich diese Vererbung noch durch Bedingungen (Regeln) gesteuert werden kann, wird diese Beschreibungssprache sehr kompliziert. Meiner Einschätzung nach ist jede komplizierte Sprache, die von einem Menschen geschrieben werden muss, fehleranfällig, was gerade im Sicherheitsumfeld ein Problem darstellt. Auswege wären Hilfsprogramme, die beim Schreiben dieser Richtlinien assistieren, um die Gefahr von Fehler zu reduzieren und die Übersicht über die Richtlinien zu erhöhen. Solche Überlegungen oder Werkzeuge werden allerdings in dem Artikel nicht erwähnt.

Interessant an GACL ist der Ansatz einen Rückgabewert zu definieren, der die Unbestimmtheit einer Entscheidung darstellt. In GACL werden aus den zugrunde liegenden Richtlinien, die Antworten „Zugriff genehmigt“ und „Zugriff verweigert“ expliziert abgeleitet. Im Gegensatz wird im Konzept von ACLs das Fehlen von Berechtigungen als eine Zugriffsverweigerung interpretiert. Allerdings wird in dem Artikel nicht der Nutzen dieses alternativen Rückgabewerts, der als „incomplete authorization“ bezeichnet wird, vorgestellt. Das Resultat für einen Benutzer ist das gleiche, ob nun ein „Zugriff verweigert“ oder ein Autorisierung nicht vollständig ist. Eventuell kann diese Art von Rückgabewert gesondert protokolliert werden und vom Administrator genutzt werden, um die Richtlinien zu vervollständigen.

Eine bemerkenswerte Eigenschaft für die damalige Zeit ist die automatische Suche eines Dienstservers nach einem Autorisierungsserver. Dieser Vorgang ist vergleichbar mit dem UDDI-Standard, der heutzutage Suchen nach Webservices ermöglicht.

Die Authentifizierungs- und Autorisierungsserver bilden die Kernelemente des vorgestellten Frameworks. Das Framework ist logisch strukturiert und die einzelnen Dienste (Elemente des Frameworks) sind autonom in ihrer Arbeitsweise – die Dienste sind in ihrer Funktion disjunkt. Bemerkenswert ist die lose Kopplung zwischen den Diensten, da der Client das einzige Bindeglied zwischen der Authentifikation, der Autorisierung und dem Dienstserver ist.

2.3.2 Security Agent Based Distributed Authorization: An Approach

2.3.2.1 Analyse

Ein völlig anderer Ansatz für die Autorisierung von Operation wird in [Varadharajan] vorgestellt. Hier werden Softwareagenten zum Ort der Dienstausführung gesendet, die dort stellvertretend für den Benutzer die gewünschte Operation unter Vorlage von Privilegien starten.

Bei der Autorisierung sind die folgenden zwei Fragen am wichtigsten. Wer ist jemand? Was darf dieser Jemand? Diese Fragen werden insbesondere im Kontext von verteilten Anwendungen gestellt, wobei drei Mobile Code-Paradigmen erkannt werden.

1. Code On Demand
2. Remote Evaluation
3. Mobile Agents

Der dritte Punkte, Mobile Agents, ist das Kernthema in [Varadharajan]. Da der Begriff Mobile Agents mittlerweile etwas abgenutzt ist, wird seine Bedeutung konkretisiert. Ein Mobile Agent in diesem Kontext ist ein Programm zusammen mit seinem Zustand, der u.a. Variablen, einen Programmzähler und einen Aufrufstack besitzt.

Es gibt drei generelle Sicherheitsprobleme bei der Arbeit mit Mobile Agents.

1. Der Host, der den Mobile Agent ausführt, muss vor diesem geschützt werden.
2. Verschiedene Mobile Agents auf einem Host müssen voreinander geschützt werden.
3. Der Mobile Agent muss vor dem Host, der ihn ausführt, geschützt werden.

Punkt 1 und 2 sind relativ einfach durch das Sandbox-Konzept zu realisieren. Punkt 3 ist derzeit nicht trivial zu lösen und bedarf weiterer Forschungsarbeit.

Da die Mobile Agents in dieser Arbeit in bestimmten Eigenschaften der Sicherheit, wie Anforderungen von Diensten, Authentifizierung und Autorisierung, betrachtet werden, werden sie als Teil eines größeren Konzepts im Folgenden als Security Agent (SA) bezeichnet.

Autorisierungsmechanismen

[Varadharajan] begründet die Notwendigkeit von SAs über die Bewertung von vorhandenen Autorisierungsmechanismen in verteilten Anwendungen. Insbesondere die Pflege von Access Control Lists, die an Ressourcen in einer verteilten Umgebung gebunden sind, ist ein aufwändiger Vorgang, da die Pflege dezentral erfolgt. Diese Fragmentierung der Privilegien kann bei der Verwendung von Capabilities (siehe auch *Abschnitt 1.5 Autorisierung*) vermieden werden. Jedoch muss es hier eine Autorisierungsstelle außerhalb des gewünschten Dienstes geben, der über alle notwendigen Autorisierungsanforderungen des gewünschten Dienstes verfügt.

SAs enthalten Code und Daten, womit sie in der Lage sind, autonom handeln können. Sie sind in der Hinsicht mit Capabilities vergleichbar, da sie ebenfalls Privilegien in sich tragen. Allerdings sind SAs vielseitiger, da sie sowohl Authentifizierungs- als auch Autorisierungsvorgänge durchführen können. Damit sind SAs in der Lage, beim Ziel Operationen stellvertretend für den Nutzer aufzurufen. Unabhängig von den eingebetteten Privilegien können die Zugriffsrechte eines SA jedoch durch ACLs auf dem Zielsystem eingeschränkt werden.

Systemoperationen

[Varadharajan] beschreibt wichtige Systemoperationen, wie die Erstellung, die Verteilung und die Übertragung von SAs und den Schutz der Daten vor Manipulation. Diese Operationen sind im Kontext dieser Diplomarbeit irrelevant und werden hier nicht vorgestellt. Interessant dagegen sind die Systemoperationen, die für die Delegation und den Rückruf von SAs verantwortlich sind.

Der Einsatz eines SAs an sich kann bereits als Delegation angesehen werden, da hier der Nutzer seine Privilegien an eine Instanz weitergibt und diese Instanz stellvertretend Operationen durchführt. SAs können allerdings auch dafür verwendet werden, nur Privilegien zu transportieren. Dazu wird ein so genanntes Delegate Token erstellt und in den SA von dem Nutzer kopiert, an den die Privilegien delegiert werden. Durch entsprechende Attribute wie Zeitstempel und Lebenszeiten und durch Signierung wird Missbrauch der Delegation reduziert.

Die Systemoperationen für den Rückruf von SAs werden in [Varadharajan] auf zwei Eigenschaften beschränkt.

1. Die Lebenszeit eines SA läuft ab. In diesem Fall werden die Privilegien nach dem Ablauf der Lebenszeit automatisch ungültig.
2. Die Integrität des SA wird zerstört. Eine solche Situation kann durch eine fehlerhafte Übertragung oder absichtliche Manipulation eines SA auftreten. In diesem Fall ist es die Aufgabe des Frameworks, den fehlerhaften SA nicht weiter zu verbreiten, nicht auszuführen und eine Warnung an den Besitzer des SA zu senden.

2.3.2.2 Bewertung

[Varadharajan] zählt drei Verfahren auf, wie in verteilten Anwendungen Zugriffskontrollen durchgeführt werden können: Code On Demand, Remote Evaluation und Mobile Agents. Code On Demand ist ein Anforderungskonzept, bei dem ein ausführender Dienst Code-Segmente (Programmteile) anfordert. Remote Evaluation ist die Auslagerung der Zugriffskontrolle außerhalb des Dienstprozesses, wie er zum Beispiel im *Abschnitt 2.3.1 A Framework for Distributed Authorization* beschrieben wird. Auch das verteilte Autorisierungssystem, das in dieser Arbeit entwickelt wird, verwendet das Konzept der Remote Evaluation. In den *Abschnitten 2.1 Gegenüberstellung lokale und verteilte Autorisierung* und *3 Entwurf* wird darauf genauer eingegangen. Die Arbeit [Varadharajan] konzentriert sich auf den dritten Punkt, der Mobile Agents. Es ist ein Konzept, das Code (Programme) zum ausführenden Dienst sendet. Die Initiation der Code-Übermittlung ist somit genau dem Prinzip des Code On Demand entgegengesetzt.

Die Beschreibung des Frameworks in [Varadharajan] wurde ausgelassen, da es irrelevant für die Bewertung im Sinne dieser Arbeit ist. Wichtig ist der Ansatz, eine andere Art der Dienstnutzung zu finden.

Die Mobile Agents sind Träger der Identität und der Privilegien des Besitzers sowie der Aufgabe, für die sie fort gesandt werden. Das Konzept sieht vor, dass ein Mobile Agent ohne die Kontrolle des Besitzers in Netzwerken auf unterschiedlichen Hosts ihre Aufgaben durchführen können. Die Hosts müssen dabei dem Besitzer nicht bekannt sein. Es werden drei generelle Sicherheitsprobleme der Mobile Agents im Zusammenhang mit der Ausführung auf fremden Hosts angesprochen. Der Schutz des Hosts und anderer Mobile Agents ist heutzutage relativ leicht durch die Sandbox-Technik zurealisieren. Allerdings gibt es keine Möglichkeit, den Mobile Agent selbst zu schützen. Der Mobile Agent liegt dem Host in seiner Gesamtheit vor. Das sind alle Identitäts- und Berechtigungsinformationen, der Programmcode und die Daten, die dem Mobile Agent mitgegeben wurden oder die er im Laufe seiner Reise gesammelt hat. Der Mobile Agent kann diese Daten zwar verschlüsseln, muss diese aber irgendwann, spätestens zum Zeitpunkt des erneuten Zugriffs auf diese Daten, entschlüsseln. Das heißt, hier sind mindestens zwei Angriffsszenarien vorstellbar. Entweder der Host sucht im Mobile Agent nach dem Schlüssel oder er wartet, bis der Mobile Agent die Daten selbst entschlüsselt hat und liest dann den Speicherbereich des Mobile Agent aus. Da der Host, auf dem der Mobile Agent ausgeführt wird, volle Kontrolle über die Prozesse auf seinem System hat, können der Mobile Agent und damit auch der Besitzer niemals sicher sein, dass seine Daten nicht ausspioniert wurden. Diese Tatsache grenzt das Ausführungsgebiet für Mobile Agents nur auf vertrauenswürdige Netzwerke ein, wenn der Mobile Agent vertrauliche Daten mit sich trägt.

[Varadharajan] gibt auch Einsatzfähigkeiten der SAs für die Delegierung und des Rückrufs von (delegierten) SAs mit. Die Delegierung ist unter der Nutzung von Capabilities elegant gelöst, was allerdings auf die Natur der Capabilities zurückzuführen ist. Es ist allerdings interessant zu sehen, dass dieses Konzept auch im Bereich der Mobile Agents genutzt werden kann. Eher unzureichend ist der Rückruf gelöst worden. Es werden zwei Möglichkeiten aufgeführt, wie ein Rückruf durchgeführt werden kann. Die Wirkungsweise des Rückrufs beschränkt sich hierbei auf das automatische Deaktivieren nach der Überschreitung der Lebenszeit und der Verletzung der Integrität eines Mobile Agent. Es gibt somit keine Aktionen, die durch den Besitzer nach der Aktivierung des Mobile Agent aufgerufen werden können. Weitere Überlegungen dazu werden in [Varadharajan] nicht aufgeführt.

Da die Integrität eines Mobile Agent auf der Signierung basiert, wäre es denkbar, für jeden Mobile Agent ein eigenes Zertifikat zu generieren und bei einem Rückruf, dieses Zertifikat zurückzuziehen. In Abhängigkeit wie schnell die unterschiedlichen Hosts ihre Zertifikatsrückruflisten aktualisieren, kann ein Mobile Agents zurückgerufen werden.

2.3.3 Authorization-Based Access Control for the Services Oriented Architecture

2.3.3.1 Analyse

Nach [Karp 2006] haben heutige SOA-Architekturen in den Bereichen Skalierbarkeit, Sicherheit und Administrierbarkeit versagt. Als Ursache wird die identitätsbasierte Zugriffskontrolle (identity-based access control, IBAC) ausgemacht. Basierend auf dieser Erkenntnis wird eine Lösung vorgestellt, die stattdessen vorgelegte Autorisierungsentscheidungen als Zugriffskontrolle (authorization-based access control, ABAC) verwendet.

Wie der Name andeutet, basiert die Zugriffskontrolle bei IBAC auf der vorgelegten Identität. Diese Identität kann auf unterschiedlichste Weise bei einer Authentifizierung erstellt bzw. bewiesen werden. Häufige Verfahren hierbei sind Zwei-Faktor-Authentifizierungen durch Benutzername und Kennwort oder Smartcard und PIN. Die Regeln, die zur Autorisierungsentscheidung herangezogen werden, werden häufig in ACLs gespeichert, wobei Identitäten oder Rollen Privilegien zugeordnet werden. Diese Privilegien beinhalten unterschiedliche Operationen, die einem Benutzer erlaubt sind durchzuführen wie beispielsweise das Lesen und Schreiben von Dateien. Im Umfeld von Web Services können diese Privilegien auch das Aufrufen eines Dienstes darstellen.

Probleme von IBAC

In Abgrenzung zu der hier vorgestellten Lösung von ABAC weisen die Konzepte IBAC und RBAC die gleichen Probleme aus und sind in diesem Kontext austauschbar.

In verteilten Anwendungen kommt es vor, dass Dienste über mehrere Domänen (siehe dazu auch *Abschnitt Domänen auf Seite 11*) hinweg genutzt werden. In solchen Situationen werden so genannte Vertrauensstellungen zwischen diesen Domänen eingerichtet. Vertrauensstellungen sind gerichtet und können daher auch nur in eine Richtung aufgebaut sein. Der Name hat seinen Ursprung darin, dass eine Domäne *B*, die einen Dienst anbietet, einen Benutzer der Domäne *A* Zugriff gewährt. Dabei vertraut Domäne *B* der Domäne *A*, dass individuelle Verträge, die vereinbart wurden, eingehalten werden. Domäne *B*, die vertraut und einen Dienst anbietet, hat keine Möglichkeit, übermittelte Identitätsinformationen nachzuprüfen.

Haben zwei Domänen keine Vertrauensstellung, ist eine explizite Authentifizierung notwendig. Sei eine Domäne *C* gegeben, die keinerlei Vertrauensstellung zu *A* hat, so muss ein Benutzer der Domäne *A* ein Benutzerkonto in der Domäne *C* besitzen, um Dienste dieser Domäne nutzen zu können. Dies führt zu einem weiteren Problem, sobald es notwendig wird, neue Benutzerkonten zu erstellen oder vorhandene zu ändern, zu sperren oder zu löschen. Hierbei seien zwei Extremfälle dargestellt.

1. Ein neuer Benutzer in Domäne *A* möchte auch Dienste in Domäne *C* nutzen. Der neue Benutzer kann allerdings so lange nicht die Dienste in Domäne *C* nutzen, bis sein Konto in der Domäne *C* angelegt wurde. Abhängig von Faktoren in Domäne *C*, auf die Domäne *A* keinen Einfluss hat, kann der neue Benutzer so lange nicht seine Arbeit auf Basis der Dienste in Domäne *C* durchführen, bis für ihn ein Konto in Domäne *C* eingerichtet wurde.
2. Ein vorhandener Benutzer in Domäne *A* erhält eine neue Position innerhalb seiner Firma und damit neue Aufgaben. Dies bedeutet, dass er unter Umständen neue Dienste nutzen muss (siehe Punkt 1). Gleichzeitig kann dies auch bedeuten, dass er Dienste, die er bisher benutzt hat, nun nicht mehr verwenden darf. Hier entsteht das Problem, dass diesem Nutzer Zugriff zu Diensten gewährt wird, für die er keine Befugnis mehr hat. Die Gründe dafür entsprechen denen aus Punkt 1.

Aber selbst zwischen Domänen mit Vertrauensstellungen können Probleme bei der Verwendung von Rollen bzw. Gruppen entstehen. Bestimmte Rollen können in ähnlicher Form in beiden Domänen existieren, werden allerdings im Regelfall nicht komplett die gleiche Bedeutung haben. Um hier ein gutes Mapping zwischen den Rollen in den verschiedenen Domänen herzustellen, kann es ggf. notwendig sein, sehr viele Rollen anzulegen. Der Aufwand für die Verwaltung steigt dementsprechend.

Die Verwendung von Rollen kann auch zu einem nicht gewünschten Informationsleck führen, da hiermit einer fremden Domäne mitgeteilt wird, welche Rollen ein bestimmter Benutzer besitzt. Dadurch kann die fremde Domäne Rückschlüsse auf die Position und die Hierarchie der Firma ziehen, in der der Benutzer arbeitet. Die Preisgabe der Identität allein kann schon ein Informationsleck darstellen, da Benutzernamen häufig Rückschlüsse auf Personennamen erlauben. Ebenso können auch aus Änderungen von Rollenzuweisen Veränderungen in Position des Benutzers oder der Firmenstruktur hergeleitet werden.

Auch SSO-Systeme bieten durch Verwendung von Identitäten als Basis eine Angriffsfläche. Bei IBAC bzw. RBAC werden die Rechte eines Benutzers durch seine Identität bzw. Rolle bestimmt. In der Regel kann ein Benutzer alle Dienste in einer Sitzung basierend auf seinen Rechten in dieser Sitzung nutzen. Bei SSO-Systemen erweitert sich diese Sitzung für den Benutzer unbemerkt auch auf entfernte Dienste. Gerade dieser Komfort für den Benutzer kann von Schadsoftware ausgenutzt werden.

ABAC

Das Schema von ABAC wird in *Abbildung 23* dargestellt. Bei IBAC bzw. RBAC übermittelt der Client die Identität bzw. die Rolle zum Dienst, der schließlich für diese Identität bzw. Rolle die Berechtigungen bei einem Server abfragt, der die Richtlinien verwaltet.

Bei ABAC hingegen authentifiziert sich der Client selbst gegenüber dem Richtlinien-Server und erhält von diesem eine Menge an Privilegien. Diese Menge oder auch Teilmenge an Privilegien legt der Client dem Dienst vor. Dieser muss nun nur noch die Echtheit der Privilegien überprüfen und gewährt bei erfolgreicher Prüfung den Zugriff.

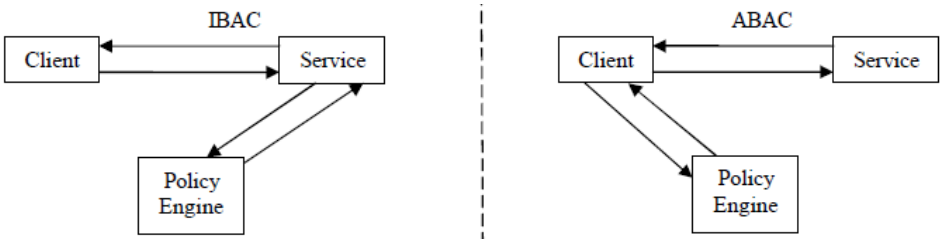

Abbildung 23 Vergleich zwischen IBAC und ABAC [Karp 2006, S. 6]

ABAC löst durch dieses Konzept viele der vorher angesprochenen Probleme des Identity Management. Dadurch dass der Dienst nicht die Authentifizierung übernehmen muss, fällt die Notwendigkeit weg, dass der Dienstanbieter selbst eine Benutzerdatenbank führen und pflegen muss. Das Mapping von Rollen unterschiedlicher Domains ist ebenfalls nicht erforderlich. Sofern es der Dienstnutzer nicht möchte, müssen auch keine Informationen über die Identität oder die Position anhand von Rollenzugehörigkeit herausgegeben werden.

Delegation wird mit ABAC sehr elegant gelöst. Während bei IBAC und RBAC Benutzerkonten und Rollenzugehörigkeit angepasst werden müssen, kann ein ABAC-Nutzer schlicht seine Privilegien jemandem weiterleiten. Der Rückruf soll ebenfalls einfach zu realisieren sein. Wie dies jedoch umgesetzt wird, wird in [Karp 2006] nicht erwähnt.

Auf eine umfassende Protokollierung und Verbindlichkeit muss auch bei ABAC nicht verzichtet werden. Bei der Übergabe der Privilegien können diese mit einem Personenzertifikat des Benutzers signiert werden, wodurch der Dienst die Identität erhalten kann. Wichtig hier-

bei ist, dass diese Übermittlung freiwillig ist und die Autorisierung nicht auf diesen Daten beruht.

In SSO-Systemen begrenzt ABAC auch das Potential von Schadsoftware. Innerhalb einer Sitzung fordert ein Client nur die Privilegien an, die er für die Nutzung des gewünschten Dienstes benötigt. Schadsoftware wäre somit in so einer Sitzung maximal in der Lage, die angeforderten Privilegien auszunutzen.

2.3.3.2 Bewertung

[Karp 2006] stellt einen interessanten Artikel vor, der die Konzepte und Vor- und Nachteile einer Architektur aufzeigt, die vollständig auf Capabilities (siehe auch Abschnitt *1.5 Autorisierung*) basiert.

Der Vollständigkeit halber sei hier noch erwähnt, dass es zwar richtig ist, dass bei IBAC der Aufbau von Vertrauensstellungen einen gewissen Arbeitsaufwand bedeutet, dies allerdings bei ABAC ebenso notwendig ist. ABAC hat lediglich Vorteile gegenüber Systemen, die keine Vertrauensstellungen besitzen und daher individuelle Benutzerdatenbanken führen müssen.

Hervorzuheben ist das angesprochene Informationsleck, das bei IBAC-basierenden Systemen zwangsläufig auftritt. [Karp 2006] stellt überzeugend dar, wie dieses Problem mithilfe von Capabilities umgangen werden kann. Eine andere Art dieses Problem zu lösen, zeigt die Ihnen vorliegende Arbeit auf, die das Informationsleck sogar besser schließt als das Konzept der Capabilities. Bei ABAC erfährt der Dienstleister unter Umständen durch die Menge der mitgeführten Privilegien innerhalb einer Capability, welche Privilegien ein Benutzer hat. Es gibt zwar die Möglichkeit bei dem Aufruf einer Operation nur eine Teilmenge der verfügbaren Privilegien mitzuliefern, allerdings wird dieses Konzept die gleichen Schwierigkeiten in der Durchsetzung haben wie das Prinzip der geringsten Privilegien (Principle of Least Privileges). Bei der vorgestellten Lösung dieser Arbeit muss der Dienstleister weder die Identität noch die Privilegien eines Benutzers erfahren.

In dem Artikel wird auch erwähnt, dass Schadsoftware potenziell weniger Schaden anrichten kann, wenn der Benutzer nur im Besitz der Capabilities ist, die er momentan benötigt. Hier entsteht wieder das gleiche Problem, dass der Benutzer oder die Software, die er verwendet, das Principle of Least Privileges befolgt. Weiterhin kann Schadsoftware, wenn sie auf ein A-BAC-basierendes System vorbereitet ist, für den Benutzer die Privilegien anfordern und daraufhin Schaden anrichten. Der Schutz, der momentan noch besteht ist der, dass Schadsoftware in der Regel nicht auf diese Art von Systemen ausgerichtet ist. Dies würde sich natürlich ändern, wenn ABAC-basierende Systeme eine gewisse Verbreitung finden.

2.3.4 SOA-aware Authorization Control

2.3.4.1 Analyse

Die Arbeit [Emig 2006] legt den Fokus auf Web Services und beschreibt einen Ansatz der Autorisierung namens Secure Service Agent (SSA). Es wird besonderer Wert darauf gelegt, existierende Anwendungen einzubinden und über Web Services zugänglich zu machen.

Der Ansatz von SSA basiert auf zwei vorhandenen Sicherheitsmodellen, dem Secure Service Proxy (SSP) und dem Intercepting Web Agent (IWA).

Der SSP ist ein Wrapper, der alle Dienstanfragen entgegennimmt. Er überprüft die Authentifizierung und Autorisierung, ist in der Lage die Anfrage zu ändern und leitet diese schließlich zum eigentlichen Dienst durch. Der SSP ist somit der Sicherungspunkt, hat aber zugleich die Fähigkeit Protokolltransformationen abhängig vom verwendeten Dienst durchzuführen. Da der SSP für jede Operation, die der eigentliche Dienst anbietet, ebenfalls eine Signatur bereit-

stellen muss, ist der Implementationsaufwand abhängig vom Funktionsangebot des eigentlichen Dienstes und es entsteht eine starke Bindung zum Wrapper, dem SSP, und dem Dienst. Änderungen am Dienst bedeuten auch die Notwendigkeit für Anpassungen im SSP.

Der IWA dagegen wird vor den Web Service geschaltet und fängt unabhängig von der verwendeten Operation alle Anfragen ab und übernimmt die Authentifizierung und Autorisierung. Der Vorteil gegenüber dem SSP ist hier klar die Unabhängigkeit von dem angebotenen Dienst und dessen Operationen. Nachteilig beim IWA ist die starke Bindung an den Applikationsserver. Es ist davon auszugehen, dass der IWA an jeden Applikationsserver individuell angepasst werden muss.

Nach [Emig 2006] enthält jede Autorisierungs-Architektur einen Policy Enforcement Point (PEP) und einen Policy Decision Point (PDP). Der PEP ist dafür verantwortlich, Bereiche zu schützen, die eine Autorisierung verlangen, und Autorisierungsentscheidungen vom PDP anzufordern. Der PDP wertet diese Autorisierungsanfragen aufgrund seiner Richtlinien aus und antwortet dem PEP, der schließlich die Programmausführung basierend auf der Antwort fortsetzt.

Secure Service Agent

Der SSA (siehe *Abbildung 24*) ist eine Kombination der beiden Sicherheitsmodelle SSP und IWA, wobei der PEP aus der Geschäftskomponente ausgelagert wird. Stattdessen ist der PEP eine eigenständige Komponente mit einem Interface, das die Aktivierung des PEP aus der Geschäftskomponente heraus erlaubt.

Die Geschäftskomponente (Business Related Component, BRC) ist die Schnittstelle innerhalb des Applikationsservers, die Aufrufe zu vorhandenen, nicht SOA-fähigen Anwendungen weiterleitet. Die BRC hat bis auf den Aufruf zum PEP keinen weiteren Schutz und hat auch keine Notwendigkeit, selbst die Authentifikation durchzuführen.

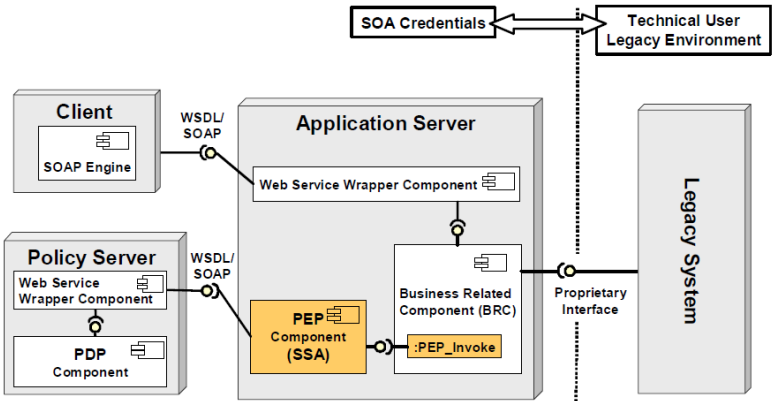

Abbildung 24 Secure Service Agent – Komponentendiagramm [Emig 2006, S. 3]

Es ist davon auszugehen, dass die BRC und der PEP auf dem gleichen System ausgeführt werden und daher die Kommunikation sehr effizient durchgeführt werden kann. Da der PEP die Autorisierungsanfrage jedoch zum PDP weiterleitet, liegt der höchste Kommunikationsaufwand zwischen PEP und PDP.

Es ist zu beachten, dass alle Identifizierungsinformationen von der Web Service Wrapper Component (WSWC) und der BRC als Geschäftsparameter durchgeleitet werden. Erst beim Aufruf des PEP werden diese Daten benötigt und gewinnen an Bedeutung. Weder die WSWC noch die BRC müssen diese Daten verstehen noch auswerten.

2.3.4.2 Bewertung

Durch die Kombination der Konzepte Secure Service Proxy (SSP) und Intercepting Web Agent (IWA) gelingt [Emig 2006] eine sehr interessante Lösung, die Secure Service Agent (SSA) genannt wird.

Der SSP hat nach [Emig 2006] eine sehr starke Bindung an die Software, die ummantelt wird, und ist damit sehr wartungsaufwändig. Allerdings wird hier vorwiegend von Altanwendungen gesprochen, denen der Weg in die SOA-Welt geöffnet werden soll. Daher sollte nicht von umfassenden Änderungen bei den Signaturen ausgegangen werden.

Eine Anpassung der Signaturen ist allerdings notwendig, ob nun das Konzept des SSP genutzt wird oder eine Geschäftskomponente, die BRC (siehe *Secure Service Agent, Seite 44*). Ein Vorteil liegt allerdings in der strikten Trennung zwischen dem Code der Dienstsicherung und dem Code zur Ansteuerung der Geschäftsprogramme.

In der Tat hat das Konzept des SSA Ähnlichkeit mit der ausgelagerten Autorisierung der vorgestellten Lösung dieser Diplomarbeit. Das Autorisierungskonzept in dieser Diplomarbeit geht allerdings in der Hinsicht weiter, da der Ort der Zugriffskontrolle variabel ist. Siehe hierzu auch *Abschnitt 3 Entwurf*.

2.3.5 ABAC – Ein Referenzmodell für attributbasierte Zugriffskontrolle

2.3.5.1 Analyse

In [Priebe 2005] stellen die Autoren einen flexiblen Ansatz der Autorisierung vor. Ausgehend von dem Standpunkt, dass die Menge an Benutzern dynamisch und heterogen ist, stellen sich die bisherigen Methoden der Autorisierung für eine große Anzahl an Benutzern (Subjekten) und Ressourcen (Objekten) als nicht optimal heraus. Autorisierung ist bereits auf Basis von Benutzerattributen (Credentials) und Metadaten möglich. [Priebe 2005] baut auf diesen Erkenntnissen auf und stellt eine Erweiterung unter dem Namen Attribute-Based Access Control (ABAC) vor. Es wird zusätzlich gezeigt, dass ABAC als Generalisierung bisheriger Autorisierungsmodelle gesehen werden kann.

Grundlage für die Autorisierung in ABAC sind Eigenschaften und Attribute von Subjekten und Objekten. In dem Grundmodell, auf dem die Arbeit von [Priebe 2005] basiert, werden Subjekte und Objekte durch Attribute repräsentiert. Die Erweiterung des Grundmodells sieht die Einführung von Sitzungen (Sessions) und Bedingungen (Conditions) vor.

Sitzungen ermöglichen es dem Benutzer, nur eine Teilmenge der vorhandenen Benutzerattribute bei der Autorisierung vorzulegen. Damit ist die Erfüllung des Konzepts Least Privileges möglich, da nur die für die Autorisierung notwendigen Attribute in die Sitzung übernommen werden müssen. Im Grundmodell können bei der Autorisierung Attribute nur mit konstanten Werten verglichen werden. Bedingungen erlauben hingegen auch den Vergleich zwischen Subjekt- und Objektattributen und auch einer neuen Attributklasse, den Umgebungsattributen, um gegen dynamische Eigenschaften der Umgebung vergleichen zu können.

Autorisierungsrichtlinien werden in ABAC in einer UML-Notation wiedergegeben. Drei Beispiele sind in *Abbildung 25* dargestellt. Beispiel a entspricht dem Grundmodell der Attributbasierten Autorisierung. Das Subjekt und das Objekt werden mit konstanten Attributwerten verglichen. Beispiele b und c zeigen die Verwendung von Bedingungen. Die verwendeten Attribute in der Bedingung sind innerhalb der „Klassen“ parameterlos aufgeführt. Die Operation enthält in den geschweiften Klammern die angewandte Bedingung. Bei Beispiel b kann die Operation *rent* nur autorisiert werden, wenn das Alter des Kunden über der Alterseinstufung des Films ist. Bei Beispiel c kann ein Bewerber eine Bewerbung nur dann lesen und schreiben, wenn er der Ersteller dieser Bewerbung ist.

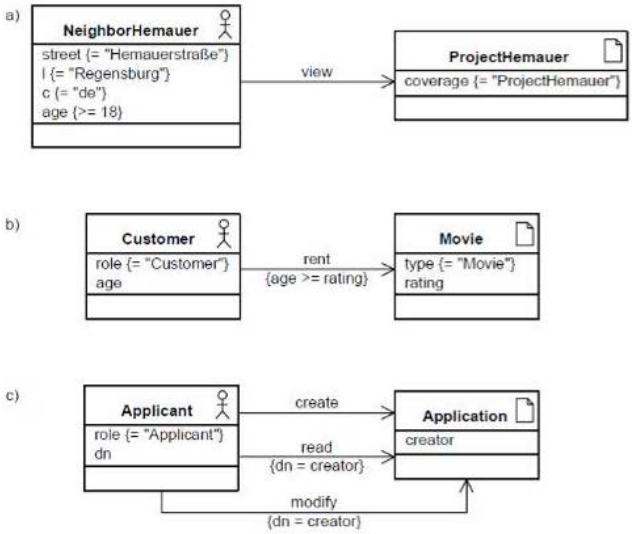

Abbildung 25 Beispielrichtlinien in UML-Notation [Priebe 2005, S. 8]

Klassische Autorisierungsmethoden

ABAC kann als Generalisierung der klassischen Autorisierungsmethoden gesehen werden. Nach [Priebe 2005] ist eine Abbildung auf DAC, MAC und RBAC möglich. Zu den Autorisierungsmethoden siehe auch *Abschnitt 1.5 Autorisierung*.

Das Eigentümerprinzip von DAC kann direkt umgesetzt, indem Identifizierungsattribute für Subjekt und Objekt eingesetzt werden. Ein Beispiel dafür ist in *Abbildung 26* zu sehen.

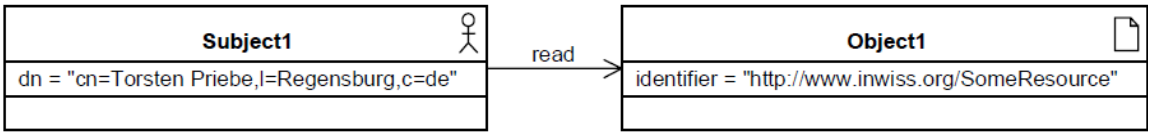

Abbildung 26 Beispiel für DAC als ABAC-Richtlinie [Priebe 2005, S. 9]

Das Prinzip von RBAC ist ebenfalls direkt in ABAC umsetzbar. Bereits in Beispiel b und c in *Abbildung 25* wurde gezeigt, dass dem Subjekt ein Attribut Rolle zugeordnet werden kann, nach dem schließlich verglichen wird.

2.3.5.2 Bewertung

Das Konzept von [Priebe 2005] ist in der Autorisierung sehr flexibel, da beiden Seiten, Subjekte und Objekte, gleichermaßen durch Attribute klassifiziert werden können.

In dem Sinne, dass den ACLs der Ressourcen keine Subjekte hinzugefügt werden, sondern Eigenschaften bzw. Attribute, die durch ein Subjekt erfüllt werden müssen, kann man ABAC als eine mächtige Erweiterung von RBAC (siehe auch in den *Grundlagen im Abschnitt 1.5.3*) ansehen, wie in dem Artikel von [Priebe 2005] auch schon festgestellt wurde. Rollen von RBAC stellen eine Untermenge der Bedeutungsmöglichkeiten von Attributen dar. ABAC kann damit auch als eine mögliche Lösung des bedingungsbehafteten RBAC gesehen werden, worin RBAC-Rollen mit zusätzlichen Bedingungen erstellt werden. In ABAC sind zusätzliche Bedingungen, seien sie statischer oder dynamischer Natur, einfach zu realisieren, indem Subjekten und Objekten weitere Attribute hinzugefügt werden.

Bemerkenswert an dieser Lösung ist, dass bei dem Zugriff auf eine Ressource die Identität des Subjekts nicht bekannt sein muss, da die Zugriffskontrolle nur auf den mitgeführten Attributen basiert. Damit weisen diese Attribute eine Ähnlichkeit zu Capabilities auf, allerdings auch erhebliche Unterschiede. Der wesentliche Unterschied ist der, dass bei ABAC die Zugriffskontrolle bei dem Zugriff auf die Ressource stattfindet, während bei Capabilities die Zu-

griffskontrolle von dem Zugriff auf Ressourcen entkoppelt ist. Der Vorteil kann allerdings gleichermaßen einen Nachteil darstellen, da durch die fehlende Identität die Protokollierung des Zugriffs auf Ressourcen erschwert wird. Es treffen hier die gleichen Überlegungen zu wie bei der Protokollierung unter Verwendung von Capabilities.

Ein erheblicher Nachteil ist die fehlende Funktion der Delegation in ABAC. Ähnlich anderen Konzepten wie zum Beispiel bei ACLs ist eine Delegation nur möglich, wenn dem Delegierten zusätzliche Rechte, im Konzept von ABAC sind dies Attribute, zugewiesen werden. Es treten die altbekannten Probleme der Zuweisung und Rücknahme dieser Rechte auf, wie sie schon im *Abschnitt 1.5.1 Zugriffskontroll-Matrix auf Seite 20* beschrieben wurden.

2.3.6 Zusammenfassende Bewertung

Die Auswahl der vorgestellten Arbeiten in diesem Abschnitt haben Konzepte der Zugriffskontrolle gezeigt, die sich in der Herangehensweise unterscheiden. Angesprochene Unterschiede sind vorwiegend die Grundlage der Autorisierung und Speicherort der Zugriffsrechte. Auch angesprochen, aber merklich nicht Schwerpunkt der meisten Arbeiten, sind die Kommunikation und Repräsentation im Zusammenhang mit der Autorisierung und Zugriffskontrolle.

Wie bereits in den Grundlagen dargestellt, basiert die Zugriffskontrolle entweder auf Identitäten bzw. Rollen, also wer jemand ist oder in welcher Funktion er agiert, oder auf seinen Befugnissen, also was ein Benutzer machen darf. Der wesentliche Unterschied der Autorisierungsverfahren ist demnach der Ort der Feststellung der Identität. Unabhängig ob eine Identität (Benutzername oder Rollen) oder Zugriffsrechte (Capabilities oder Attribute) bei der Zugriffskontrolle vorgewiesen werden, irgendwann wurde dennoch die Identität überprüft. Die in diesem Kapitel vorgestellten Verfahren unterscheiden sich im Wesentlichen darin, wo diese Identität festgestellt wurde. Der Ort der Feststellung der Zugriffsrechte ist entweder direkt den zu schützenden Ressourcen vorgeschaltet oder ist von den Ressourcen entkoppelt.

Mit der Grundlage der Autorisierung entscheidet sich auch der Ort der Speicherung der Zugriffsrechte. In den Arbeiten [Woo 1993] und [Priebe 2005] ist gut zu sehen, dass bei der Verwendung von Capabilities und Attributen, die Zugriffsrechte nicht mehr bei der Ressource liegen, sondern in einer separaten Datenbank. Bei den anderen Konzepten liegen die Zugriffsrechte in der Regel bei den Ressourcen selbst, obwohl es dafür keine Notwendigkeit gibt. Es dient lediglich der einfacheren Verwaltung und der Performanz beim Zugriff auf die Zugriffsrechte.

3 Entwurf

Die Basis der Implementation ist der Entwurf, der in diesem Kapitel beschrieben wird. Zu Beginn werden die grundlegende Architektur und die verschiedenen Rollen beschrieben. Anschließend werden die Anwendungsfälle gesammelt, die die Workflows für das verteilte Autorisierungssystem repräsentieren. Darauf folgt die Modellierung der Klassen, die bei der verteilten Autorisierung von Bedeutung sind. Klassen, die nicht wesentlicher Bestandteil des Autorisierungssystems sind, werden in diesem Kapitel nicht aufgeführt. Anschließend wird die Interaktion zwischen den Klassen verdeutlicht.

Abschließend werden die XML-Schemata der Anwendungsdefinitionen und -konfigurationen entworfen. Definitionen für Anwendungen und die Konfiguration derer werden in XML-Dateien gespeichert. In den *Abschnitten 3.5.1 und 3.5.2* werden die Datenstrukturen für die Anwendungsdefinitionen und -konfigurationen beschrieben. Bei dem Entwurf der XML-Schemata gibt es vielfältige Möglichkeiten Daten zu beschreiben. In Anlehnung an die Best Practices von [Stephenson 2004] wurden die Entwurfsentscheidungen getroffen.

Die Architektur sieht vier Komponenten vor.

- Framework für den Client
- Framework für den Server
- Autorisierungsserver
- Verwaltung für den Autorisierungsserver

Das Framework sieht Programmierschnittstellen (application programming interface, API) für den Client und den Server vor. Der Client muss in der Lage sein, Autorisierungsanfragen abzuschicken und Autorisierungsantworten zu empfangen. Auf der Serverseite soll es möglich sein, die Autorisierung durch eigene Bibliotheken zu erweitern.

Der Autorisierungsserver ist ein Serverdienst, der für die Verarbeitung von Autorisierungsanfragen verantwortlich ist. Das Framework für den Client kommuniziert per Web Services mit dem Autorisierungsserver. Der Entwurf der Verwaltung für den Autorisierungsserver zeigt auf, welche Funktionalität die Verwaltung mindestens besitzen muss, damit Anwendungen konfiguriert werden können. Sie gibt einen guten Überblick über die Zusammenhänge der verschiedenen Klassen im Framework.

Abbildung 27 zeigt den Nachrichtenaustausch zwischen den Partnern der verteilten Autorisierung. Es ist zu beachten, dass es sich bei diesen Rollen um logische Rollen handelt. Im Extremfall könnten sich also alle auf demselben physikalischen System befinden. Wie der Abbildung zu entnehmen, ist der Autorisierungsdienst von dem Benutzer und dem SSO-System entkoppelt. Der Autorisierungsdienst tritt nur mit der Anwendung des Resource Partner und anderen Autorisierungssystemen in Kontakt.

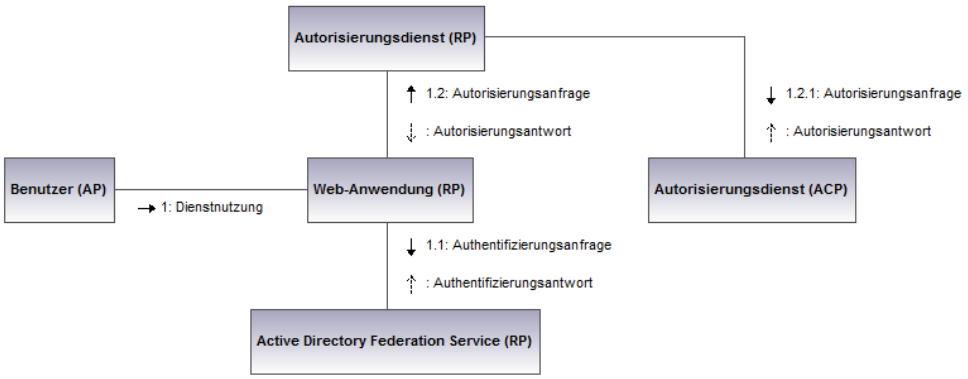

Abbildung 27 Ablauf einer delegierten Autorisierung

Die Architektur und die verwendeten Namensräume des Frameworks werden im *Abschnitt 3.3 Modellierung der Klassen* vorgestellt.

3.1 Rollen der verteilten Autorisierung

In Single Sign-On-Umgebungen mit verteilter Autorisierung gibt es drei Rollen.

- Account Partner
- Resource Partner
- Access Control Partner

In SSO-Umgebungen sind nur der Account Partner und der Resource Partner von Bedeutung. Mit der Hinzunahme der verteilten Autorisierung wird die neue Rolle Access Control Partner eingeführt. Der Resource Partner und der Access Control Partner verwenden die Dienste des verteilten Autorisierungssystems. Diese Dienste sind die Realisierung des Clients und des Servers für die verteilte Autorisierung, da diese lokal beim Resource Partner und fern beim Access Control Partner installiert werden. Zur Ansteuerung des Autorisierungssystems beinhaltet es u.a. die API und Tools zum Erstellen und Testen der verteilten Autorisierung.

Abbildung 28 zeigt das Standardszenario zur Verwendung von SSO (hier ADFS) und verteilter Autorisierung. Der Account Partner ist in diesem Szenario gleichzeitig auch der Access Control Partner. Das bedeutet, dass der Account Partner alle Autorisierungsanfragen für die eigenen Nutzer der Anwendung beim Resource Partner selbst bearbeitet. Wie bereits in der Problemanalyse im *Abschnitt 2.1.2* aufgezeigt, ist dies die einfachste zu erstellende und zu wartende Konfiguration.

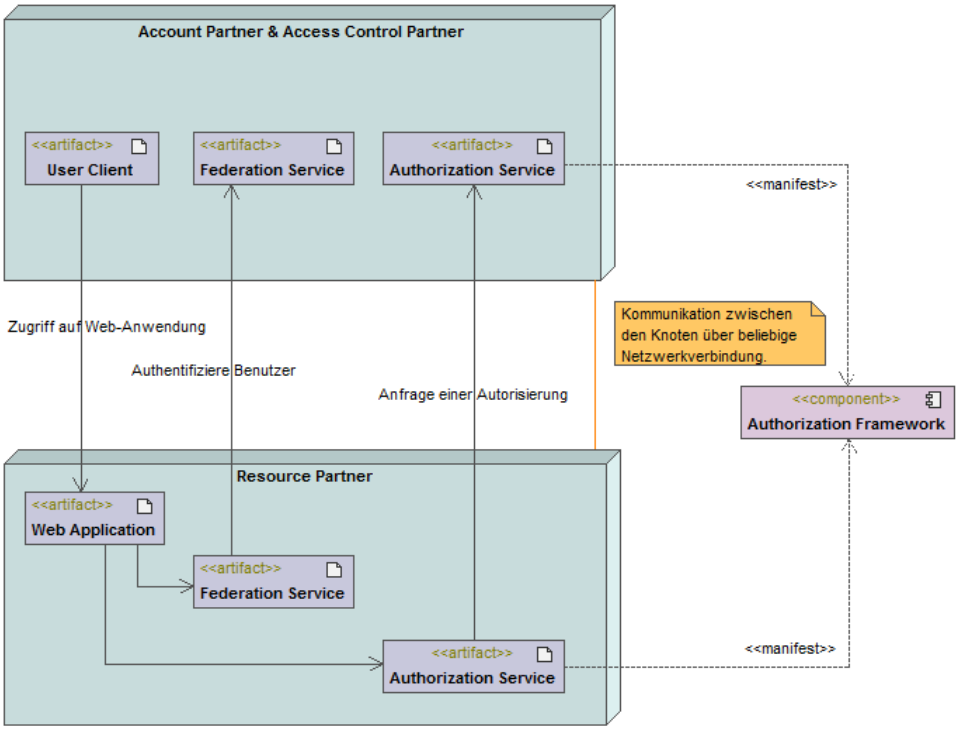

Abbildung 28 Verteilung bei zwei Parteien

1. Ein Nutzer beim Account Partner greift auf eine Anwendung beim Resource Partner zu.
2. Der Nutzer wird vom Resource Partner mittels ADFS authentifiziert.
3. Greift der Nutzer auf eine geschützte Ressource zu, wird vom Resource Partner eine Autorisierungsanfrage an den lokalen Autorisierungsdienst gestellt. Dieser ist entwe-

der in der Lage diese Anfrage selbst zu beantworten oder leitet die Anfrage an den fernen Autorisierungsdienst des Account Partner weiter.

Die verteilte Autorisierung ermöglicht auch neue Szenarien, die mittels lokaler Autorisierung nicht realisierbar wären. In diesem erweiterten Szenario, werden alle drei Rollen (Account Partner, Resource Partner und Access Control Partner) durch jeweils eine Partei realisiert. Der Ablauf wird in *Abbildung 29* dargestellt.

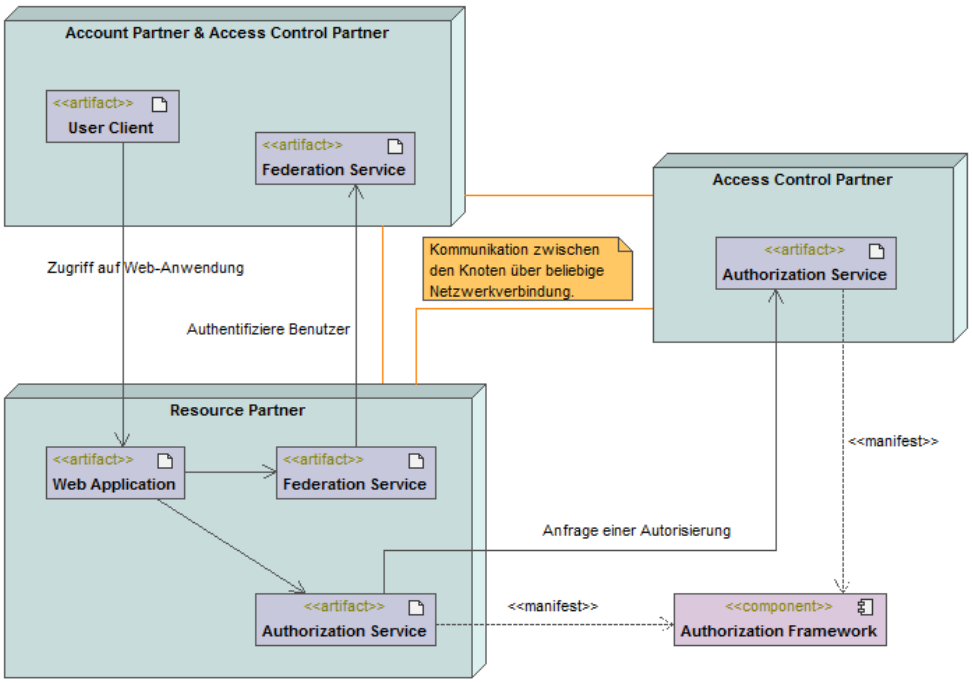

Abbildung 29 Verteilung bei drei Parteien

1. Ein Nutzer beim Account Partner greift auf eine Anwendung beim Resource Partner zu.
2. Der Nutzer wird vom Resource Partner mittels ADFS authentifiziert. Die Authentifizierung erfolgt hierbei zwischen Resource Partner und Account Partner. Der Access Control Partner ist in diesem Schritt nicht involviert.
3. Greift der Nutzer auf eine geschützte Ressource zu, wird vom Resource Partner eine Autorisierungsanfrage an den lokalen Autorisierungsdienst gestellt. Dieser ist entweder in der Lage diese Anfrage selbst zu beantworten oder leitet die Anfrage an den fernen Autorisierungsdienst weiter, der in diesem Szenario die dritte Partei, der Access Control Partner ist.

3.2 Modellierung der Anwendungsfälle

Die Modellierung der Anwendungsfälle enthält die Vorgänge, die das Framework und die Verwaltung bieten müssen, um eine verteilte Autorisierung im Sinne der Problemanalyse anbieten zu können. Sie stellen zugleich die Anforderungen und damit die Ziele an die zu entwickelnde Lösung dar.

3.2.1 Framework

Der Client erstellt lokal im Prozess eine Autorisierungsanfrage, die beim Client die Zugriffskontrolle darstellt. Auf Basis der Autorisierungsantwort wird der Programmfluss beim Client gesteuert. Die Autorisierungsanfrage wird an die Dienste der verteilten Autorisierung über-

mittelt. Diese Übermittlung stellt den Anwendungsfall *Autorisierung prüfen* dar, der in *Abbildung 30* gezeigt wird.

Dieser Anwendungsfall verwendet je nach Konfiguration einen der beiden Anwendungsfälle *Autorisierung lokal prüfen* oder *Autorisierung delegieren*. Ist der Autorisierungsdienst konfiguriert, die Anfrage selbst zu beantworten, wird der Anwendungsfall *Autorisierung lokal prüfen* verwendet. Ist der Autorisierungsdienst nicht in der Lage, die Autorisierung selbst zu erteilen oder ist derart konfiguriert, diese an einen weiteren Autorisierungsdienst zu delegieren, wird der Anwendungsfall *Autorisierung delegieren* verwendet.

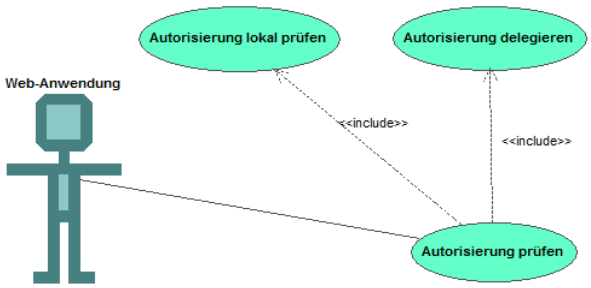

Abbildung 30 Anwendungsfälle des Aktors Web-Anwendung

3.2.2 Verwaltung

Die Verwaltung der Anwendungen und Richtlinien sind die Themen der Modellierung dieses Abschnitts. Die Verwaltung sieht zwei Aktoren vor:

1. Applikationsadministrator
2. Richtlinienadministrator

Es gibt somit eine strikte Trennung zwischen der Einrichtung von Applikationen und der Konfiguration der Richtlinien. Es sind zwei getrennte Rollen notwendig, um eine Richtlinie für eine Applikation zu aktivieren.

3.2.2.1 Applikationsadministrator

Der Applikationsadministrator (*siehe Abbildung 31*) erstellt, ändert oder löscht Applikationen im Autorisierungsdienst. Zudem aktiviert er gemeinsam mit dem Richtlinienadministrator eine Richtlinie für eine Anwendung.

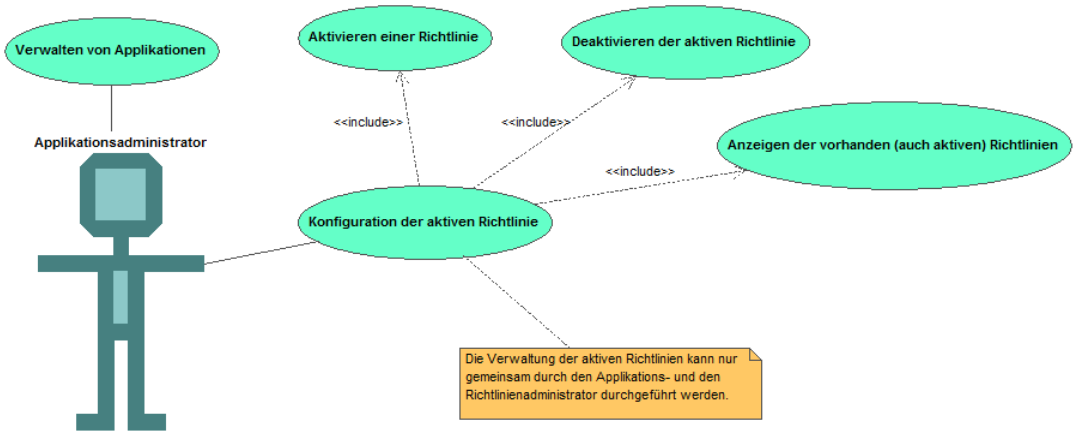

Abbildung 31 Anwendungsfälle des Aktors Applikationsadministrator

Verwalten von Applikationen

Der Anwendungsfall *Verwalten von Applikationen* enthält die Anwendungsfälle, wie sie in *Abbildung 32* aufgeführt sind.

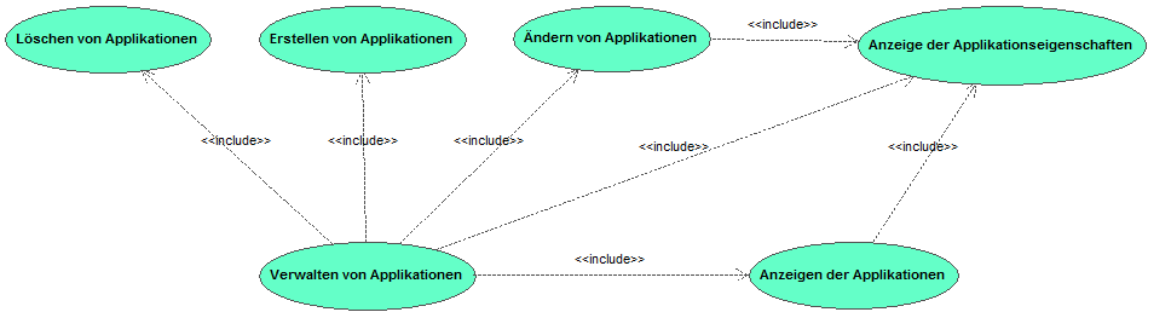

Abbildung 32 Anwendungsfälle zur Verwaltung von Applikationen

Erstellen von Applikationen bedeutet in der Diensteverwaltung das Importieren einer Anwendungsdefinition, nicht das Erstellen einer solchen. Eine Anwendungsdefinition stellt eine Vorlage für eine Anwendungskonfiguration dar. Die Anwendungskonfiguration besteht aus zwei Teilen:

1. Konfiguration der Anwendung
2. Richtlinien für die Anwendung

Die Konfiguration der Anwendung beinhaltet u.a. einen Anzeigenamen, eine ID und weitere Metadaten wie den Pfad zur Anwendung. Diese Daten können vom Richtlinienadministrator nicht verändert werden. Im Gegensatz dazu ist der Applikationsadministrator nicht in der Lage die Richtlinien der Anwendung zu verändern.

3.2.2.2 Richtlinienadministrator

Dem Richtlinienadministrator ist der umfangreiche Anwendungsfall *Verwalten von Richtlinien* zugewiesen. Dieser Anwendungsfall ist in *Abbildung 33* dargestellt.

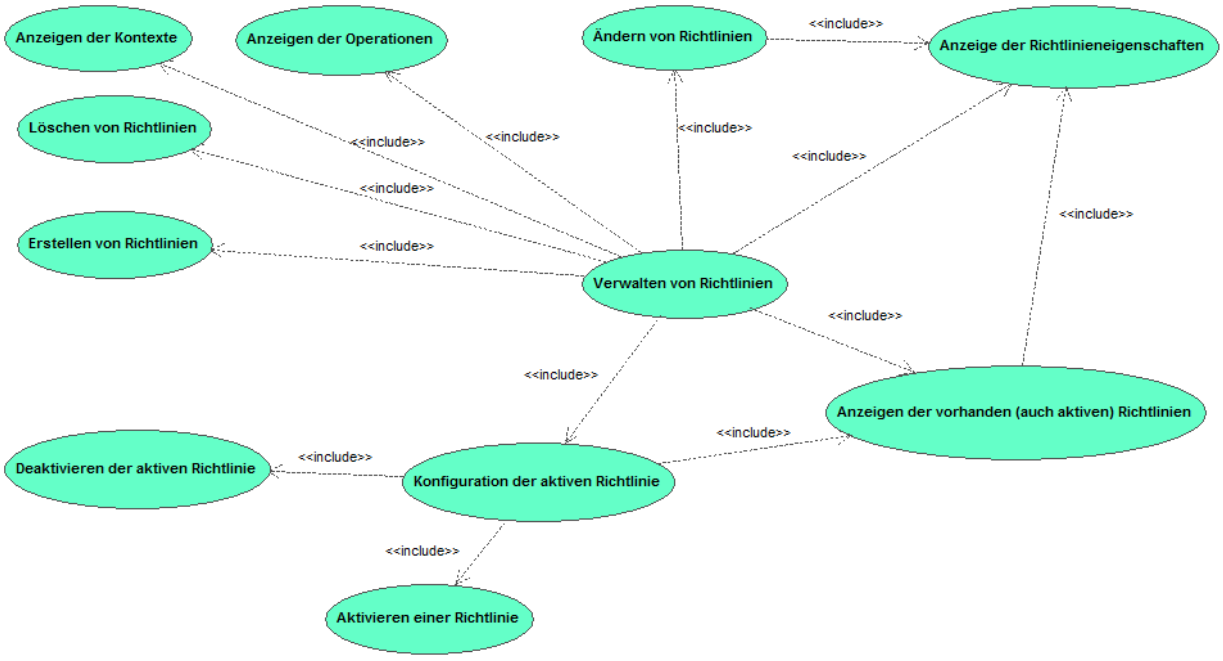

Abbildung 33 Anwendungsfälle zur Verwaltung von Richtlinien

Das Gewicht dieser Arbeit liegt nicht in der Modellierung. Daher werden im Folgenden nur die nicht trivialen Anwendungsfälle erläutert.

Der Anwendungsfall *Erstellen von Richtlinien* erstellt auf Basis einer hinterlegten Anwendungsdefinition einen neuen Konfigurationssatz. Dieser dient als Rahmen für den Anwendungsfall *Ändern der Richtlinien*, der weiter unten beschrieben wird.

Für eine Anwendung, die vom Applikationsadministrator angelegt wurde, können mehrere Richtliniensätze angelegt werden, aber nur einer dieser Richtliniensätze kann zur gleichen Zeit aktiv sein. Der Anwendungsfall *Anzeigen der vorhandenen (auch aktiven) Richtlinien* listet alle für diese Anwendung verfügbaren Richtlinien auf, wobei die jeweils aktive Richtlinie besonders gekennzeichnet wird.

Ändern von Richtlinien

Der Anwendungsfall *Ändern von Richtlinien* enthält die Anwendungsfälle, wie sie in *Abbildung 34* aufgeführt sind. Wie zu sehen ist, enthält dieser Anwendungsfall die Verwaltung von untergeordneten Gruppen, deren Verwaltung im Folgenden durch eigene Anwendungsfälle beschrieben werden.

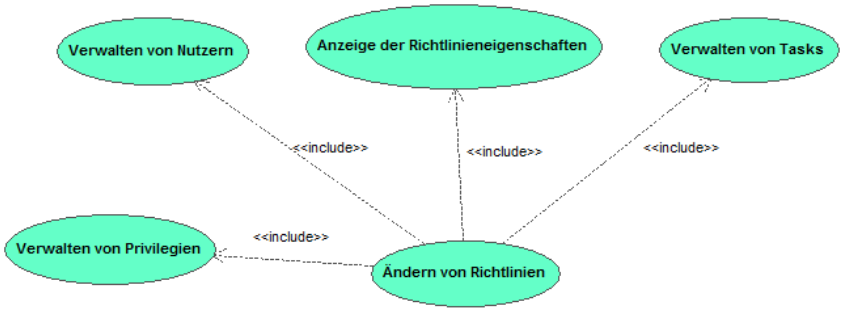

Abbildung 34 Anwendungsfälle zum Ändern von Richtlinien

Verwalten von Nutzern

Das Autorisierungsframework entscheidet bei Autorisierungsanfragen auf der Basis von Nutzern. Nutzer sind Container, die Definitionen für Identitäten und Rollenzugehörigkeiten enthalten. Identitäten sind beispielsweise Windows-Benutzerkonten. Rollenzugehörigkeiten können aufgrund von Windows-Sicherheitsgruppen oder LDAP-Anfragen definiert werden. Der Anwendungsfall *Verwalten von Nutzern* enthält die Anwendungsfälle, wie sie in *Abbildung 35* aufgeführt sind.

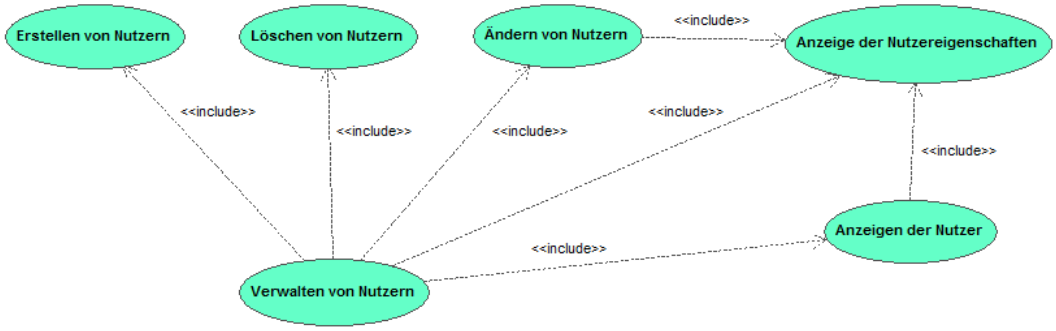

Abbildung 35 Anwendungsfälle zur Verwaltung von Nutzern

Nutzer werden innerhalb einer Anwendungskonfiguration verwaltet. Eine Anwendungsdefinition enthält keine Nutzer. Das bedeutet, dass die Definition von Nutzern eine Konfigurationsentscheidung des Richtlinienadministrators ist und separat für jede Anwendungskonfiguration vorgenommen werden muss. Überschneidungen oder Konflikte von Nutzern in unterschiedlichen Anwendungskonfigurationen sind daher nicht möglich.

Verwalten von Tasks

Tasks sind Container, die Operationen und auch Tasks enthalten können. Eine rekursive Einbettung von Tasks, auch indirekt, ist nicht erlaubt. Privilegien werden auf Basis von Tasks konfiguriert, um die Verwaltung zu vereinfachen. Tasks bestehen in der Regel aus vielen Operationen.

Der Anwendungsfall *Verwalten von Tasks* enthält die Anwendungsfälle, wie sie in *Abbildung 36* aufgeführt sind. Die Anwendungsfälle *Erstellen von Tasks* und *Ändern von Tasks* nutzen den bereits aus dem Anwendungsfall *Verwalten von Richtlinien* bekannten Anwendungsfall *Anzeigen von Operationen*. Die Konfigurationssoftware erlaubt zur Vereinfachung der Konfiguration die Anzeige der genutzten Operationen und Nutzer.

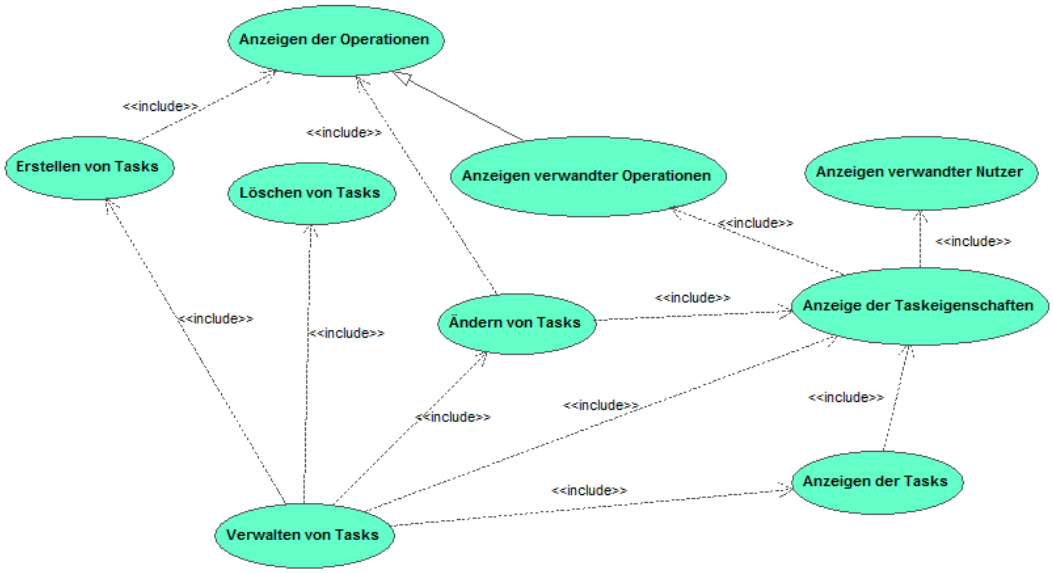

Abbildung 36 Anwendungsfälle zur Verwaltung von Tasks

Verwalten von Privilegien

Die Privilegien werden als eine separate Liste gespeichert. Ein Privileg ist eine Zuordnung von Nutzern zu Tasks. Solch eine Zuordnung kann nicht verändert, sondern nur erstellt oder gelöscht werden. In einer späteren Version der Verwaltungssoftware ist das Ändern von Privilegien und das Aktivieren bzw. Deaktivieren durchaus möglich. Der Anwendungsfall *Verwalten von Privilegien* enthält die Anwendungsfälle, wie sie in *Abbildung 37* aufgeführt sind.

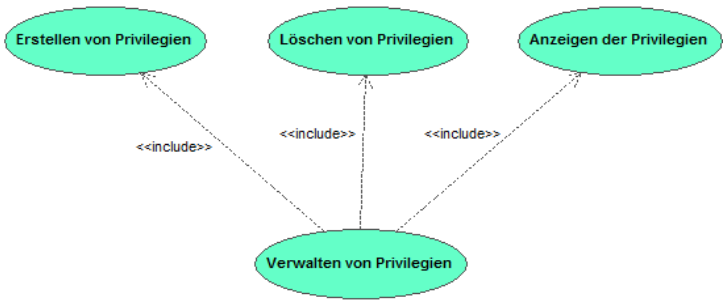

Abbildung 37 Anwendungsfälle zur Verwaltung von Privilegien

3.2.2.3 Konfigurationsadministrator

Der Aktor Konfigurationsadministrator konfiguriert die Einstellungen des Autorisierungsdienstes. Zu den Anwendungsfällen dieses Aktors gehören die Anwendungsfälle, wie sie in *Abbildung 38* aufgeführt sind.

Die *Konfiguration des Autorisierungs-Servers* umfasst die Netzwerk- und Protokollkonfiguration. Die Netzwerkkonfiguration steuert, unter welcher Adresse der Dienst im Netzwerk verfügbar ist. Die Protokollkonfiguration enthält die Einstellungen, welche globalen Ereignisse protokolliert werden sollen. Dazu gehören zum Beispiel das Hinzufügen oder Löschen von Anwendungskonfigurationen, aber keine lokalen Ereignisse wie die Auswertungsergebnisse von Autorisierungsanfragen in den einzelnen Anwendungen.

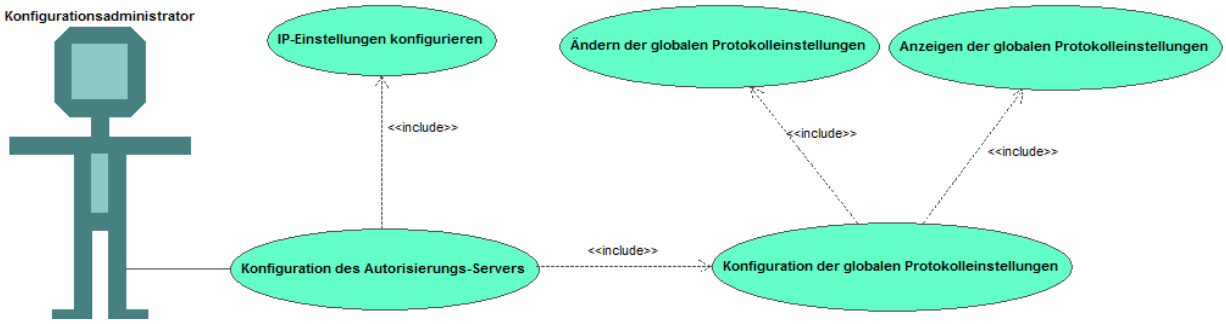

Abbildung 38 Anwendungsfälle des Aktors Konfigurationsadministrator

3.3 Modellierung der Klassen

Aus den Anwendungsfällen, die im *Abschnitt 3.2* modelliert wurden, lässt sich die notwendige Architektur der Anwendung herleiten. Die gesamte Anwendung, die in dieser Arbeit erstellt wird, gliedert sich in vier Bereiche, die in den folgenden Abschnitten detailliert dargestellt werden.

1. Common
2. Configuration
3. Client
4. Server

Namensräume, die in Anlehnung der Bereichsfunktion benannt sind, strukturieren den Quellcode der Anwendung.

1. *Thesis.Authorization*
2. *Thesis.Authorization.Configuration*
3. *Thesis.Authorization.Client*
4. *Thesis.Authorization.Server*

Die Anwendung verwendet als Ausführungsumgebung .NET in der Version 2.0. In .NET werden Bibliotheken und ausführbare Anwendungen als Assembly bezeichnet. Die Datenklassen in dieser Anwendung sind alle in einer Assembly *AuthorizationLib* zusammengefasst. Der Autorisierungsdienst sowie die Konfigurationsverwaltung sind eigenständige, ausführbare Anwendungen, die die *AuthorizationLib* referenzieren.

3.3.1 Namensraum Common

Der Bereich *Common* enthält Klassen, die gemeinsam von allen anderen Namensräumen genutzt werden. Zu diesem Bereich gehören die Klassen, die in *Abbildung 39* abgebildet sind.

Die Klassen enthalten etliche Attribute und Methoden, auf die hier zum Teil eingegangen werden, sofern sie relevant für das Konzept der verteilten Autorisierung sind. Elemente, die lediglich für den Programmablauf relevant sind, so zum Beispiel Methoden, die der Serialisierung und Deserialisierung dienen, werden nicht erläutert.

Die meisten Klassen enthalten Methoden, die in zwei Anwendungsbereiche eingeordnet werden können. Der erste Bereich dient der Erstellung von Anwendungsdefinitionen. Diese Methoden sind als internal deklariert, was bedeutet, dass diese Methoden nur von der Autorisierungssoftware selbst aufgerufen werden können. Nutzer der Autorisierungs-Bibliothek werden diese Methoden nicht sehen.

Die restlichen Methoden und Attribute sind Teil des Frameworks für die Nutzer der Autorisierungs-Bibliothek. Diese Methoden dienen dem Aufbau von Autorisierungsanfragen und dem Auswerten von Antworten.

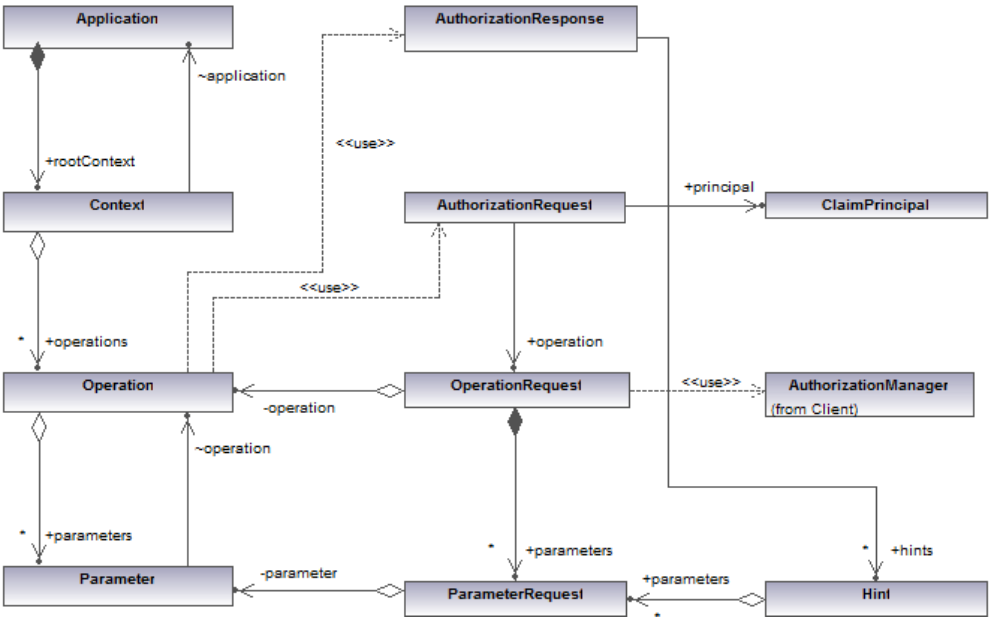

Abbildung 39 Klassendiagramm Fx Common

3.3.1.1 Klassen der Anwendungsdefinition

Die Klassen *Application*, *Context*, *Operation* und *Parameter* (*Abbildung 40*) definieren eine Anwendung. Allen Klassen ist gemeinsam, dass sie die Attribute *id*, *revision* und *name* enthalten, die entsprechend für einen eindeutigen Bezeichner, die Revisionsnummer und einen benutzerfreundlichen Namen stehen.

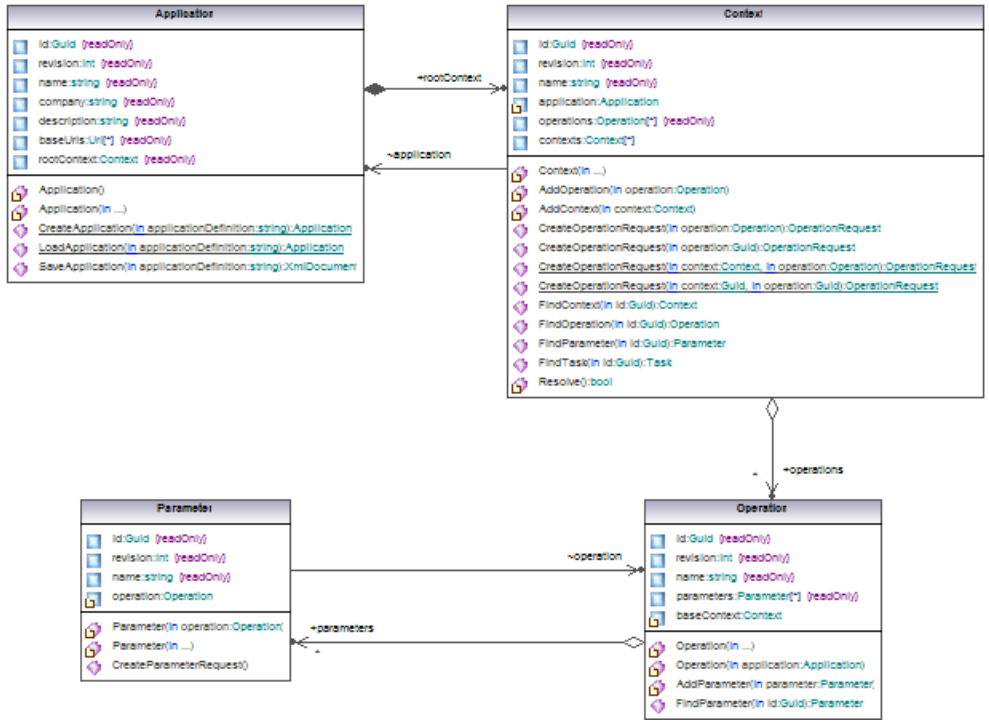

Abbildung 40 Klassendiagramm Fx Common - Anwendungsdefinition

Die Klasse *Application* bildet die Wurzel einer Anwendungsdefinition. Die Anwendungsdefinition enthält einige Metadaten, die u.a. die Herkunft und den Zweck der Anwendung beschreiben. Das wichtigste Attribut der *Application*-Klasse ist das Attribut *context*, das einen Verweis auf den Basiskontext enthält.

Kontexte werden durch die Klasse *Context* repräsentiert. Es gibt einen Basiskontext, den jede Anwendung enthalten muss. Dieser Kontext ist immer vorhanden und wird implizit bei der Erstellung einer Anwendung angelegt. Kontexte können weitere Kontexte und die Operationen enthalten, die eine Anwendung zur Verfügung stellt. Kontexte bilden Sichtbarkeitsbereiche für die Operationen. Das heißt, Operationen sind in untergeordneten Kontexten sichtbar, aber nicht in nebenläufigen Kontextzweigen. Kontexte können somit den logischen Zusammenhang von Prozessen in Anwendungen abbilden. Die Klasse *Context* enthält mehrere überladene Methoden *CreateOperationRequest*, die für den Aufbau einer Autorisierungsanfrage verwendet werden können. Welche Bedeutung diese Methoden haben, wird im nächsten *Abschnitt Klassen des Nachrichtenaustauschs* erläutert.

Operationen werden durch die Klasse *Operation* repräsentiert und bilden die Ebene der Autorisierung. Autorisierungsanfragen werden auf Basis von Operationen gestellt. Operationen können Parameter enthalten, die für die Autorisierungsentscheidung relevant sind.

Die Klasse *Parameter* ist ein Platzhalter für die Parameter einer Operation. Sie enthält keine Informationen über Datentyp oder die Verwendungsweise.

3.3.1.2 Klassen des Nachrichtenaustauschs

Dieser Abschnitt beschäftigt sich mit den Klassen (*Abbildung 41*), die für den Nachrichtenaustausch notwendig sind. Zum Nachrichtenaustausch gehören die Autorisierungsanfrage und die Autorisierungsantwort, die entsprechend durch die Klassen *AuthorizationRequest* und *AuthorizationResponse* dargestellt werden.

Die *AuthorizationRequest*-Klasse hat mehrere überladene Konstruktoren, die die Erstellung einer Autorisierungsanfrage ermöglichen. In jedem Fall ist die Übergabe eines *ClaimPrincipal* notwendig. Optional bei der Initialisierung einer *AuthorizationRequest*-Klasse sind die Angaben von *OperationRequest*-Instanzen und einer Ablaufzeit.

Die Klasse *ClaimPrincipal* enthält Informationen über die Identität und Rollen eines Nutzers.

Die Klasse *OperationRequest* stellt die Operation mit ihren Parametern dar, für die eine Autorisierung angefragt wird. Die Parameter sind in *ParameterRequest*-Instanzen gespeichert.

Die Antwort auf eine Autorisierungsanfrage stellt die Klasse *AuthorizationResponse* dar. Diese enthält zur Referenz die komplette Autorisierungsanfrage. Besonders wichtig in dieser Klasse sind die beiden Attribute *accessGranted* und *responseValidUntil*, die entsprechend angeben, wie die Autorisierungsanfrage entschieden wurde und bis wann diese Entscheidung gültig ist. Auf Basis dieser beiden Attribute wird der Programmfluss in der anfragenden Anwendung gesteuert. Die Attribute *message* und *hints* sind besonders dann hilfreich, wenn die Autorisierungsanfrage negativ beantwortet wurde. Das Attribut *message* enthält dann einen menschenlesbaren Text mit der Begründung der Ablehnung. Das Attribut *hints* ist ein Array von *Hint*-Instanzen, die Korrekturhinweise enthalten, wie eine positive Antwort erreicht werden kann. Das Vorhandensein von Korrekturhinweisen ist optional, da dies nicht in allen Situationen möglich bzw. gewünscht ist.

Die *AuthorizationResponse*-Klasse bietet zwei Methoden, die die Überprüfung der Gültigkeit der Autorisierungsantwort vereinfachen. Die überladene Methode *IsValid* überprüft, ob die Autorisierungsantwort noch gültig ist. Entweder kann sie parameterlos verwendet werden, wobei sie die aktuelle Systemzeit verwendet, oder es kann eine Zeit- und Datumsangabe mitgegeben werden, auf dessen Basis die Gültigkeit überprüft wird. Die Methode *IsAccessGranted* kombiniert die Überprüfung der Gültigkeit mit der Autorisierungsentscheidung, wobei immer die aktuelle Systemzeit verwendet wird.

Die Klasse *Hint* enthält Korrekturhinweise, wie eine positive Autorisierungsantwort erreicht werden kann. Jede *Hint*-Instanz für sich ist ein abgeschlossener Korrekturhinweis. Die Klasse enthält ein Array von Parametern mit Werten, die verwendet werden müssten, um eine positive Antwort zu bekommen. Die Attribute *validFrom* und *validTo* sind Datumswerte, die angeben, in welchen Zeitraum dieser Korrekturhinweis Gültigkeit hat. Zusätzlich enthält die Klasse das Attribut *message*, das einen menschenlesbaren Text enthält. Dieser kann dem Benutzer angezeigt werden, damit dieser seinen Arbeitsvorgang abändern kann, um dennoch eine Autorisierung zu erhalten.

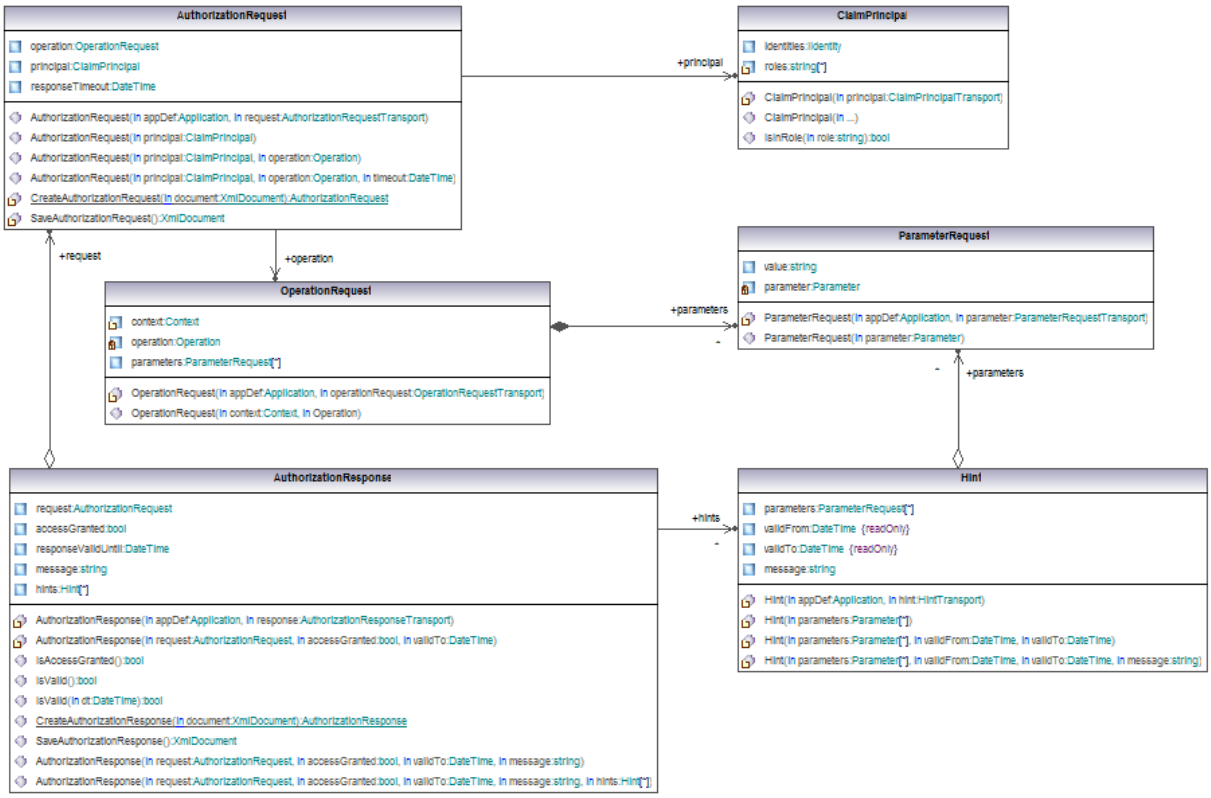

Abbildung 41 Klassendiagramm Fx Common - Nachrichtenaustausch

3.3.2 Namensraum Configuration

Der Bereich *Configuration* enthält Klassen, die für die Anwendungskonfiguration notwendig sind. Dieser Bereich wird daher vorwiegend von der Verwaltungssoftware genutzt. Zu diesem Bereich gehören die Klassen, die in *Abbildung 42* abgebildet sind, deren Zweck und Arbeitsweise im Folgenden genauer erklärt werden.

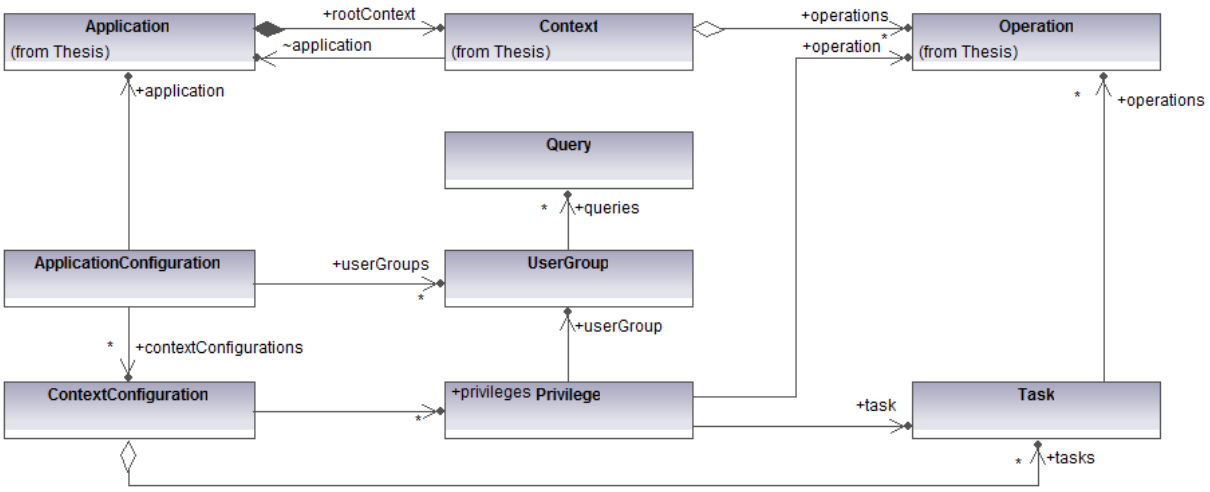

Abbildung 42 Klassendiagramm Fx Configuration - Übersicht

Die Konfigurationsklassen haben ähnlich den Anwendungsdefinitionsklassen, die im *Abschnitt 3.3.1.1* besprochen wurden, ebenfalls zwei Aufgabenbereiche. Zum einen dienen diese Klassen der Erstellung von Anwendungskonfigurationen, zum anderen werden diese Klassen serverseitig bei der Auswertung von Autorisierungsanfragen verwendet.

3.3.2.1 Klassen der Konfigurationsdefinition

Die Basis für die Konfigurationsdefinition bildet die Klasse *ApplicationConfiguration*. Sie enthält Arrays von *UserGroup*-, *ContextConfiguration*- und *AuthorizationAssembly*-Klassen. Bemerkenswert ist hierbei, dass es sich im Gegensatz zu den Kontexten, die eine Hierarchie bilden, hier um flache Arrays handelt. Diese Verflachung ist notwendig, um den Suchaufwand bei Autorisierungsentscheidungen zu minimieren. Während in einer Hierarchie eine rekursive Suche notwendig wäre, kann nun eine optimierte, flache Liste verwendet werden, die zum Beispiel hashbasiert ist.

Die Klasse *ContextConfiguration* spiegelt einen Kontext wieder, der mit Attributen ergänzt wurde, die die Konfiguration darstellen. Das bedeutet, eine *ContextConfiguration* enthält zusätzlich zu einem Verweis zu einem Kontext Arrays von Tasks und Privilegien, die diesem Kontext innerhalb einer Konfiguration zugewiesen wurden.

Tasks werden durch die Klasse *Task* repräsentiert, die neben einem eindeutigen Bezeichner und einem Namen zwei Arrays für Operationen und Tasks enthält.

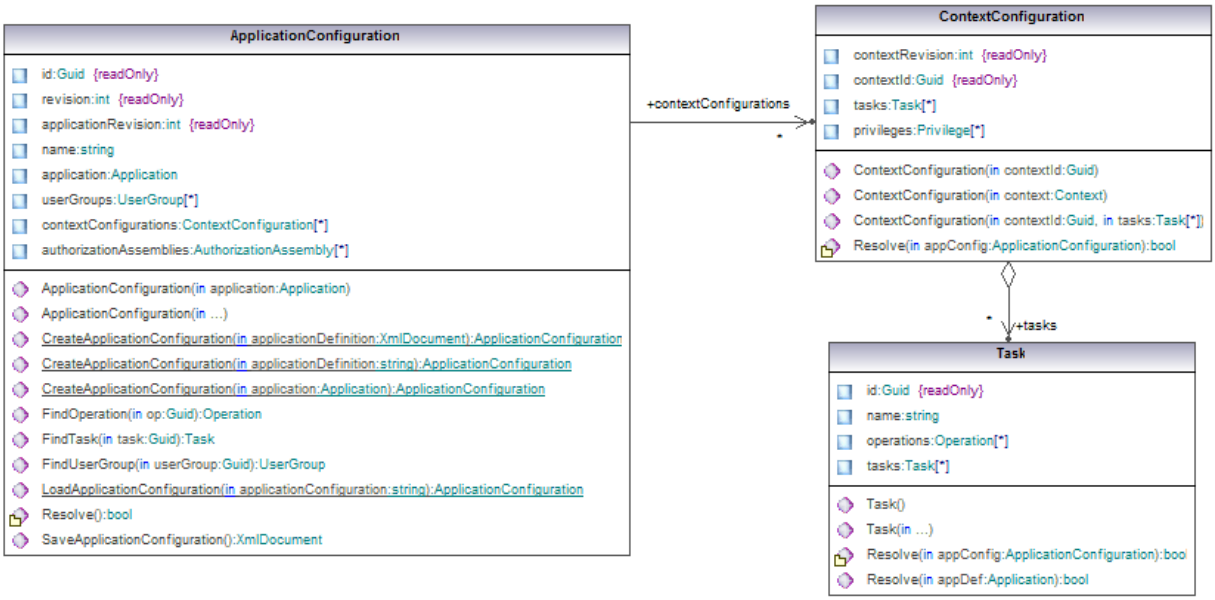

Abbildung 43 Klassendiagramm Fx Configuration - Konfigurationsdefinition

3.3.2.2 Klassen der Rechteverteilung

Privilegien bzw. Rechte werden in der Konfiguration an Tasks gebunden. Während der Auswertung werden Privilegien auf die Operationen abgebildet, um Autorisierungsentscheidungen auf Basis von Operationen durchzuführen. Dieser Prozess wird im *Abschnitt 3.4.4 Prozesse des AuthorizationEvaluator* genau erläutert.

Kernelement bei der Vergabe von Privilegien bildet die Klasse *Privilege*. Diese Klasse stellt ein Mapping zwischen Tasks und Rollen her. Solch ein Mapping ist als Erlaubnis zu verstehen, dass eine bestimmte Rolle berechtigt ist, den mit diesem Privileg verbundenen Task und die darin enthaltenen Operationen auszuführen.

Rollen werden durch die Klasse *UserGroup* repräsentiert. Rollen können durch Angabe von Identitäten, Windows NT-Benutzerkonten und LDAP-Anfragen definiert werden. Identitäten können je nach Kontext unterschiedliche Informationen enthalten. Je nach Implementation können Identitäten zum Beispiel E-Mail-Adressen, Kontoname des Benutzers oder Rolleninformationen enthalten. Im Rahmen von Single Sign-On-Systemen, die wesentlicher Gegenstand dieser Arbeit sind, werden Identitätsinformationen aus den Claims bestehen, die das SSO-System übermittelt.

Windows NT-Benutzerkonten umfassen einzelne Benutzerkonten sowie Windows NT-Sicherheitsgruppen, denen ein Benutzer angehören kann. Sehr flexibel, aber auch langsam ist die Zuordnung durch LDAP-Anfragen, die durch die Klasse *Query* abgebildet werden. Diese Klasse speichert Informationen zum LDAP-Server und die Anfrage selbst.

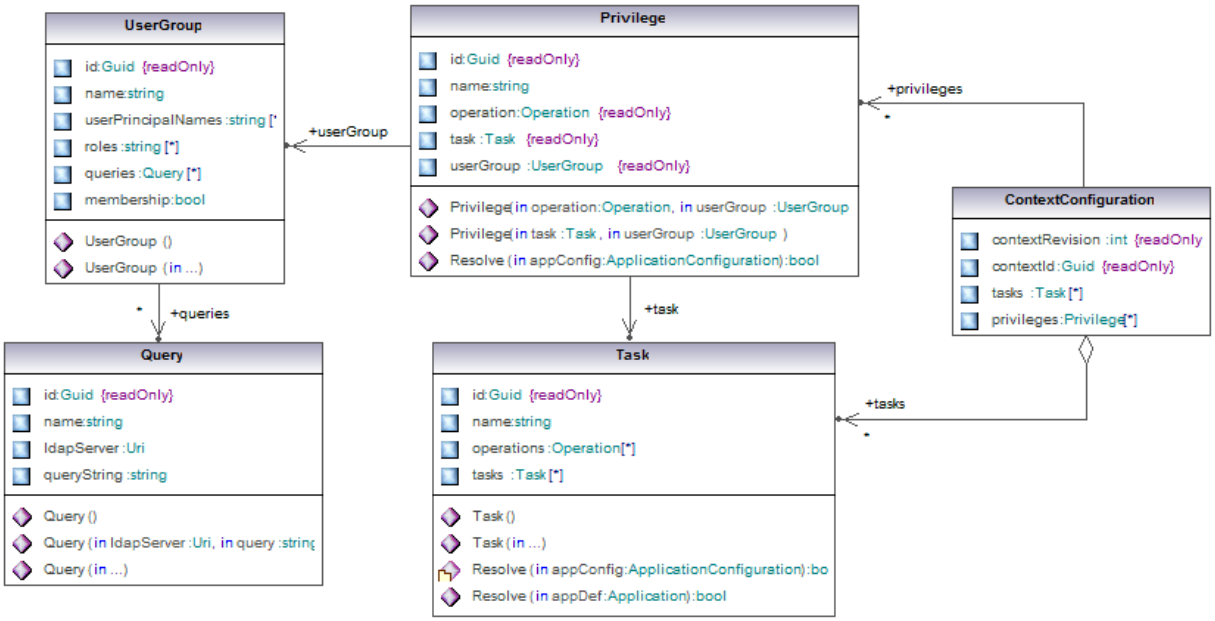

Abbildung 44 Klassendiagramm Fx Configuration - Rechteverteilung

3.3.3 Namensraum Client

Der Bereich *Client* enthält die Klassen, die vom Anwendungsentwickler benutzt werden, der das Autorisierungsframework verwendet. Der Entwickler wird während der Autorisierung auch Klassen aus dem Bereich *Common* nutzen müssen, aber über den Bereich *Client* erhält er erst Zugriff auf diese Klassen.

Dieser Bereich enthält nur eine Klasse namens *AuthorizationManager* (*Abbildung 45*), die nur zwei Konstruktoren und eine wichtige Methode, *CheckAccess*, enthält. Über die Konstruktoren wird der Klasse mitgeteilt, wie der Autorisierungsserver zu erreichen ist und welche Anwendung verwendet wird. Für die Angabe der Anwendung kann entweder der eindeutige Bezeichner der Anwendung verwendet werden oder aber es wird ein Pfad zur Anwendungsdefinition übergeben.

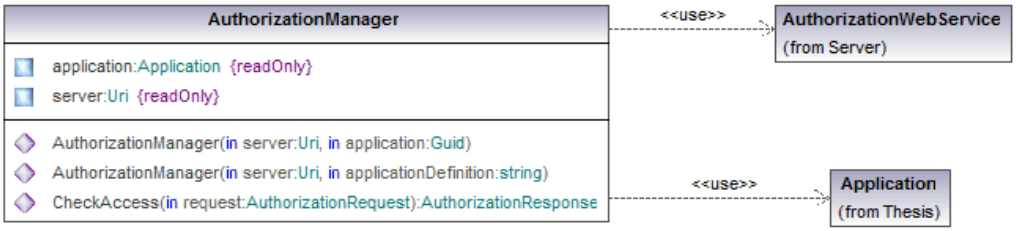

Abbildung 45 Klassendiagramm Fx Client

Die Methode *CheckAccess* übermittelt eine Autorisierungsanfrage an den Autorisierungsserver, der bei der Instanziierung der Klasse angegeben wurde. Die Übermittlung erfolgt über Web Services, wobei die Nachricht vor der Übermittlung in ein Format transformiert werden muss, das für Web Services notwendig ist. Mehr zu dieser Transformation ist auch dem *Abschnitt 4.3 Kommunikation* zu entnehmen. Als Rückgabewert erhält der Client eine Autorisierungsantwort.

3.3.4 Namensraum Server

Im Bereich *Server* spielen zwei Klassen eine wichtige Rolle. Die Klasse *AuthorizationWebService* ist die Schnittstelle für Client-Anfragen. Die wichtigste Methode ist zweifelsohne *CheckAccess*. Da diese Klasse einen Web Service darstellt, können dessen Methoden von jeder Art Client aufgerufen werden, der mit Web Services umgehen kann. Wahrscheinlich ist es jedoch, dass diese Methode vom *AuthorizationManager* aufgerufen wird, der die Autorisierungsanfrage an den *AuthorizationWebService* weiterleitet.

Die Klasse *AuthorizationEngine* stellt den Autorisierungsdienst dar. Sie steuert den Web Service *AuthorizationWebService*. Die Klasse *AuthorizationEngine* ist in einen Windows-Dienst eingebettet und kapselt somit die gesamte Funktionalität des Dienstes.

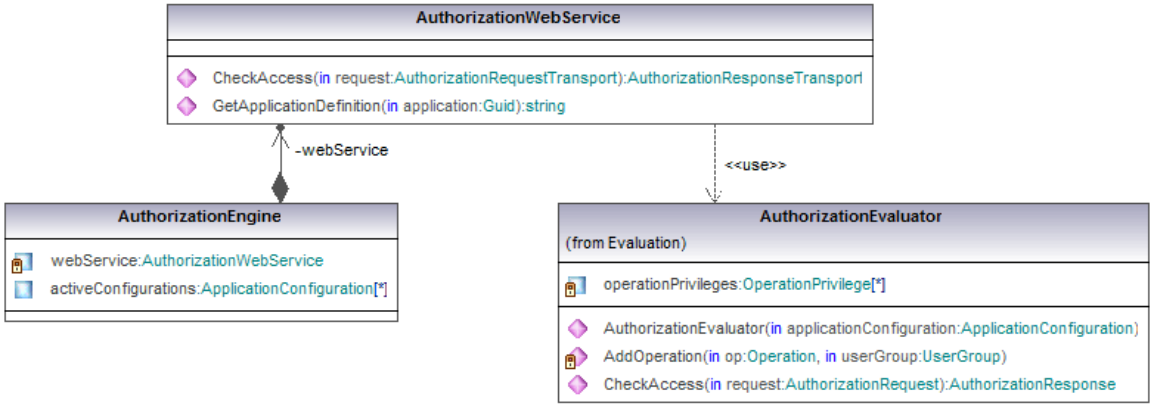

Abbildung 46 Klassendiagramm Server

In einer späteren Version des Autorisierungsdienstes ist vorgesehen, dass Autorisierungsentscheidungen an benutzerdefinierte Assemblies delegiert werden können. Sollte die Konfiguration mittels Identitäten, Windows NT-Konten und LDAP-Anfragen nicht ausreichen, kann mittels des Interfaces *AuthorizationResponder* (*Abbildung 47*) die Funktionalität des Autorisierungsdienstes beliebig erweitert werden. Autorisierungsentscheidungen können somit vollkommen an die Bedürfnisse angepasst werden.

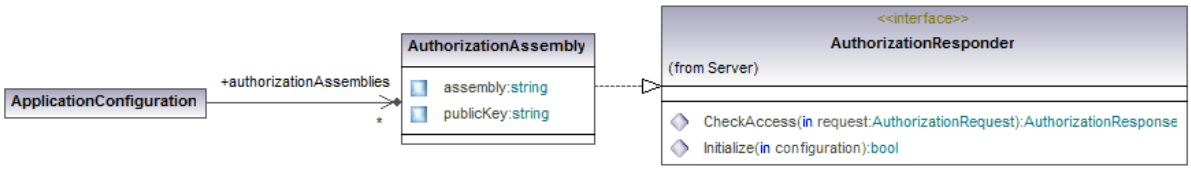

Abbildung 47 Klassendiagramm Fx Server - Erweiterungssystem

Das Interface fordert zwei Methoden *CheckAccess* und *Initialize*. Die Methode *Initialize* wird vom Autorisierungsdienst aufgerufen, wenn die benutzerdefinierte Assembly das erste Mal geladen wird. Sie erhält Konfigurationsdaten, die derzeit nicht weiter spezifiziert sind. Die Konfigurationsdaten enthalten Informationen über die Konfiguration des Autorisierungsdienstes und zusätzliche Daten, die die benutzerdefinierte Assembly benötigt.

Die Methode *CheckAccess* ist äquivalent zu den bisher kennengelernten *CheckAccess*-Methoden. Die benutzerdefinierte Assembly erhält die vollständige Autorisierungsanfrage zur Auswertung und muss ebenfalls eine vollständige Autorisierungsantwort zurückgeben.

3.4 Interaktion der Klassen

Im *Abschnitt 3.2* wurden die Anwendungsfälle gesammelt, die im Rahmen der verteilten Autorisierung auftreten können. Im *Abschnitt 3.3* wurden die Klassen modelliert, die notwendig sind, um die Anwendungsfälle zu realisieren. In diesem Abschnitt wird nun eine Auswahl an wichtigen Prozessen gezeigt, die die Interaktion zwischen den Klassen beschreiben. Wichtig sind Prozesse, die an den Aufbau von Autorisierungsanfragen und -ergebnissen, der Autorisierungsentscheidung und an der Kommunikation beteiligt sind.

3.4.1 Initialisierung des Autorisierungsmanagers

Abbildung 48 zeigt den generellen Ablauf für den Aufbau einer Autorisierungsanfrage eines Clients an das verteilte Autorisierungssystem. Der Client steht für eine beliebige Anwendung, die die verteilte Autorisierung nutzt. Verglichen mit dem Beispielszenario aus *Abschnitt 1.6* steht der Client hier für die Web-Anwendung, die von den Studenten und Mitarbeiten genutzt wird.

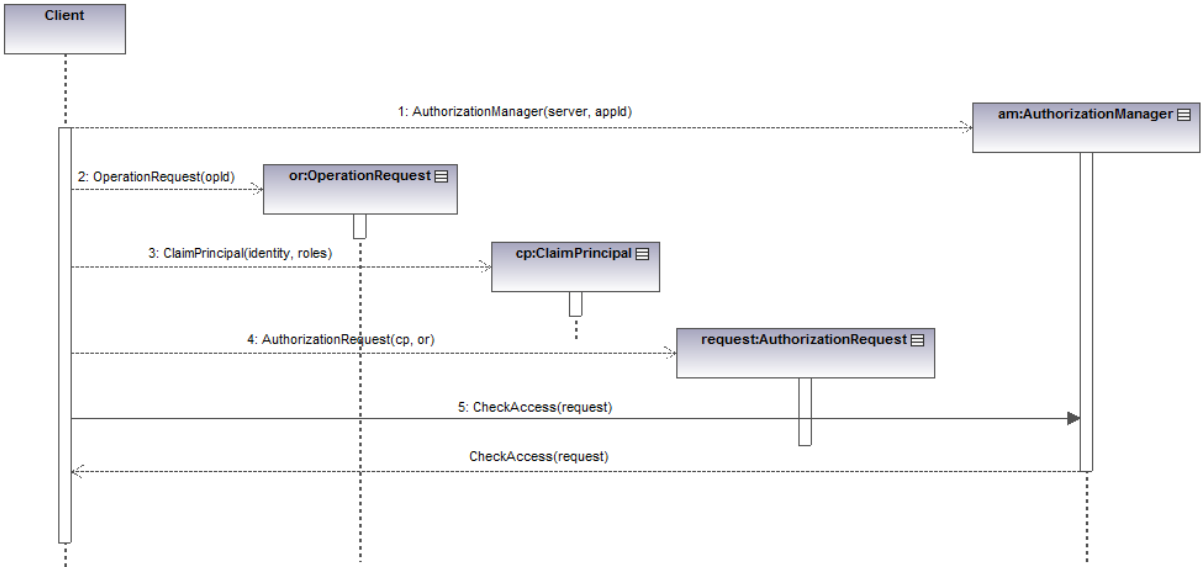

Abbildung 48 Sequenzdiagramm: Autorisierungsanfrage eines Clients

1. Der Client beginnt damit, eine Instanz des *AuthorizationManager* zu erstellen. Diese Instanz kann auch für weitere Autorisierungsanfragen vom Client benutzt werden.
2. Der Client erstellt eine Instanz der Klasse *OperationRequest*. Als Parameter wird die ID der Operation übergeben, für die die Autorisierung überprüft werden soll.
3. Der Client erstellt eine Instanz der Klasse *ClaimPrincipal*. Als Parameter können die Identität und die Rollen des zu autorisierenden Benutzers übergeben werden. Üblicherweise werden diese Informationen der Anwendung über Authentifizierungsmethoden mitgeteilt. Wie diese Informationen in ASP.NET und ASP.NET mit ADFS ermittelt werden, wird im *Abschnitt 4.4 Authentifizierung* beschrieben.
4. Der Client erstellt eine Instanz der Klasse *AuthorizationRequest* und übergibt die erzeugten Instanzen aus den Schritten 2 und 3. Die Autorisierungsanfrage besitzt damit alle notwendigen Informationen, die für eine Autorisierung zwingend notwendig sind. Die Klasse besitzt weitere Eigenschaften, um den Prozess der Autorisierung genauer

zu steuern. Weitere Informationen zu diesen Eigenschaften können im *Abschnitt 3.3.1.2 Klassen des Nachrichtenaustauschs auf Seite 57* nachgelesen werden.

5. Der Client ruft die Methode *CheckAccess* des *AuthorizationManager* auf und übergibt die aus Schritt 4 erzeugte Instanz der *AuthorizationRequest*. Der *AuthorizationManager* antwortet mit der Rückgabe einer *AuthorizationResponse*-Instanz. Nach Abschluss der Autorisierungsanfrage wertet der Client die Autorisierungsantwort aus. Der Programmverlauf wird im Folgenden ohne weiteren Einfluss des verteilten Autorisierungssystems vom Client gesteuert.

3.4.2 Autorisierungsanfragen an den Autorisierungsmanager

Der *AuthorizationManager* übernimmt die Abwicklung der Autorisierung auf der Client-Seite. Er wird vom Client initialisiert, wie im *Abschnitt 3.4.1* beschrieben wurde. Danach kann der Client über die *CheckAccess*-Methode Autorisierungsanfragen starten. Der Ablauf einer Autorisierungsanfrage innerhalb des *AuthorizationManager* wird in *Abbildung 49* dargestellt.

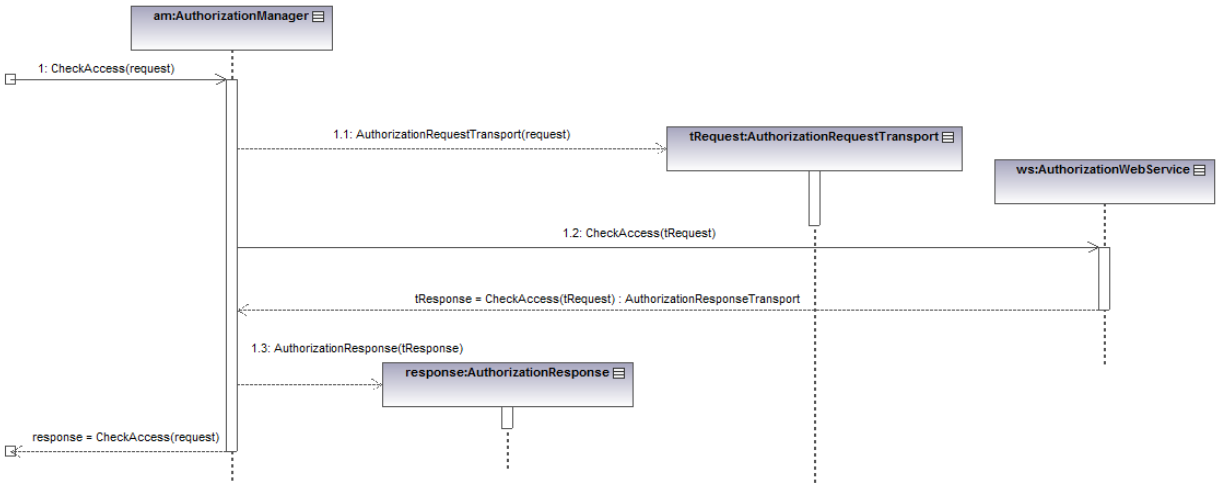

Abbildung 49 Sequenzdiagramm: AuthorizationManager bei einer Autorisierungsanfrage

1. Ein Client sendet eine Autorisierungsanfrage an den *AuthorizationManager*. Dazu wird ein Parameter *request* vom Typ *AuthorizationRequest* übergeben, der notwendige Informationen für die Autorisierung enthält.

   1.1. Der *AuthorizationManager* erstellt aus *request* eine neue Instanz *tRequest* vom Typ *AuthorizationRequestTransport*. *AuthorizationRequestTransport* ist eine spezielle Klasse für die Kommunikation mit Web Services. Auf die Transportklassen wird in der Realisierung im *Abschnitt 4.3 Kommunikation* genauer eingegangen.

   1.2. Der *AuthorizationManager* ruft die Methode *CheckAccess* auf dem Web Service *AuthorizationWebService* auf und übergibt dazu die Instanz *tRequest*, die in Schritt 1.2 erstellt wurde. Der Web Service beantwortet die Anfrage mit einer Instanz *tResponse* vom Typ *AuthorizationResponseTransport*. Diese Instanz enthält das Ergebnis der Autorisierungsanfrage.

   1.3. Im letzten Schritt erstellt der *AuthorizationManager* eine neue Instanz *response* vom Typ *AuthorizationResponse* unter Verwendung von *tResponse* aus dem vorherigen Schritt. *response* wird schließlich vom *AuthorizationManager* an den Client zurückgegeben.

3.4.3 Initialisierung des Web Services und Autorisierungsanfragen

Unabhängig von der Art, wie der Web Service betrieben wird, durchläuft der Web Service bei seinem Start eine Initialisierung, in der die Anwendungskonfigurationen geladen und die darin gespeicherten Privilegien für eine schnelle Abfrage im Speicher optimiert werden.

1. Der Host erstellt den Web Service und ruft den Konstruktor auf. Der Host kann zum Beispiel der IIS oder ein Windows-Dienst sein. Generell ist aber jede Anwendung denkbar, die in der Lage ist Web Services zu erstellen und zu betreiben.

   1.1. Der Web Service lädt die Anwendungskonfigurationen, für die er konfiguriert wurde. In *Abbildung 50* wird zur Einfachheit nur das Laden für eine einzelne Anwendungskonfiguration dargestellt.

   1.2. Die Anwendungskonfiguration wird als Parameter bei der Erstellung einer Instanz des *AuthorizationEvaluator* verwendet. Dieser verarbeitet die in der Anwendungskonfiguration gespeicherten Privilegien und optimiert diese im Speicher, um Abfragen schnell zu beantworten. Die Prozesse des *AuthorizationEvaluator* werden im *Abschnitt 3.4.4* genauer beschrieben.

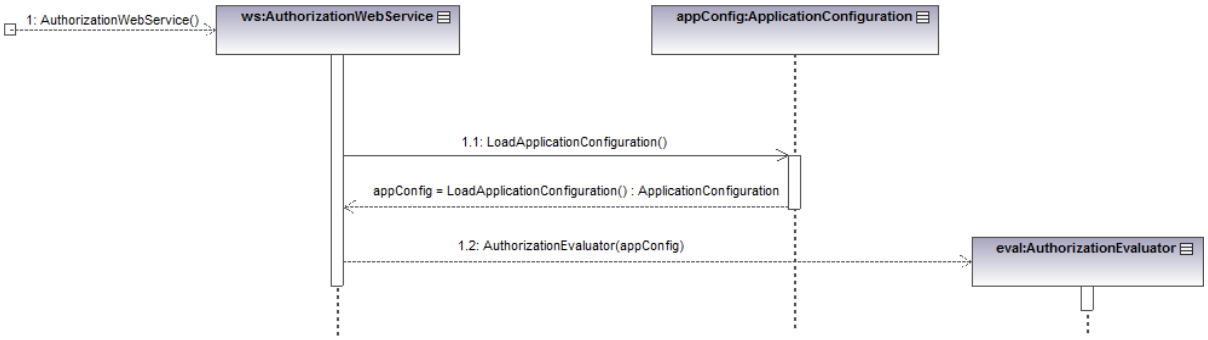

Abbildung 50 Sequenzdiagramm: Initialisierung des Web Services

Nach der Initialisierung ist der Web Service einsatzbereit und kann Autorisierungsanfragen entgegennehmen und verarbeiten. Der Prozess einer solchen Autorisierungsanfrage wird in den folgenden Schritten und *Abbildung 51* näher beschreiben.

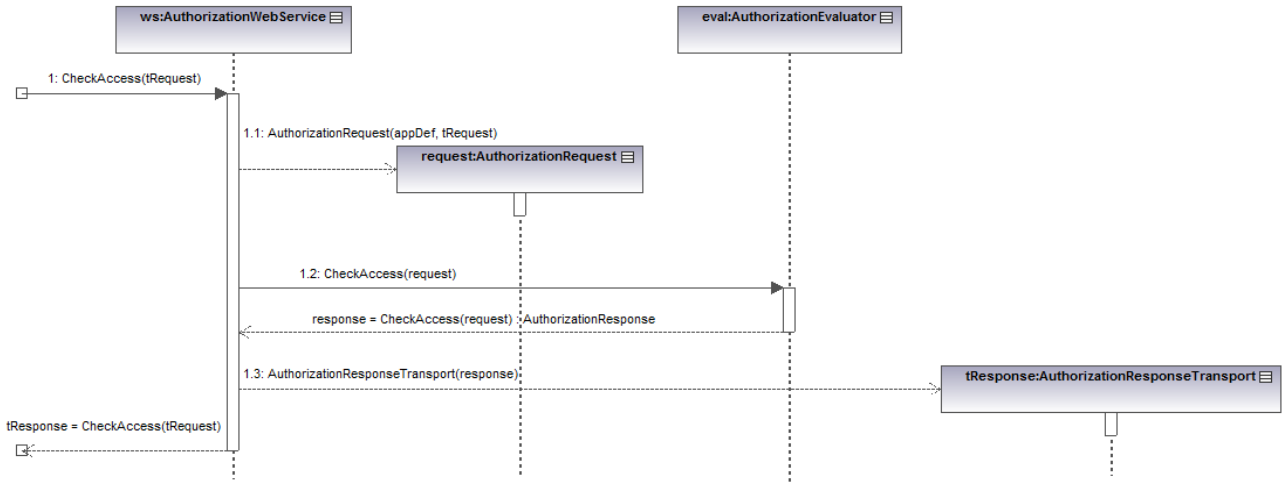

Abbildung 51 Sequenzdiagramm: Autorisierungsanfrage an den Web Service

1. Ein fremder Prozess, der in der Regel auf einem entfernten System läuft, sendet eine Autorisierungsanfrage an den Web Service, indem dieser die Methode *CheckAccess* aufruft und als Parameter eine Instanz vom Typ *AuthorizationRequestTransport* übergibt.

   1.1. Der Web Service erstellt aus der Transportklasse *tRequest* unter Verwendung der entsprechenden Anwendungsdefinition eine Instanz vom Typ *AuthorizationRequest*.

1.2. Der Web Service ruft die *CheckAccess*-Methode vom *AuthorizationEvaluator* auf, der bereits während der Initialisierung des Web Service erstellt wurde. Als Antwort wird eine Instanz *response* vom Typ *AuthorizationResponse* zurückgegeben, die das Ergebnis der Autorisierungsanfrage beinhaltet.

1.3. Schließlich erstellt der Web Service unter Verwendung von *response* eine Instanz *tResponse* vom Typ *AuthorizationResponseTransport*. Das heißt, die Daten der Autorisierungsantwort *response* werden in die Instanz *tResponse* kopiert, damit sie vom Web Service übertragen werden können.

3.4.4 Prozesse des AuthorizationEvaluator

Die Klasse *AuthorizationEvaluator* übernimmt die Überprüfung der Privilegien von Benutzern. Es gibt zwei wichtige Phasen, die in diesem Abschnitt vorgestellt werden. Zu Beginn wird die Klasse mit einer Anwendungskonfiguration initialisiert. Dabei werden alle Privilegien gegliedert nach Operation im Speicher optimiert gespeichert, so dass eine Überprüfung der Berechtigungen für einen bestimmten Benutzer erfolgen kann. Die Anfangsphase wird in *Abbildung 52* und *Abbildung 53* gezeigt, die Überprüfung der Privilegien in *Abbildung 54*.

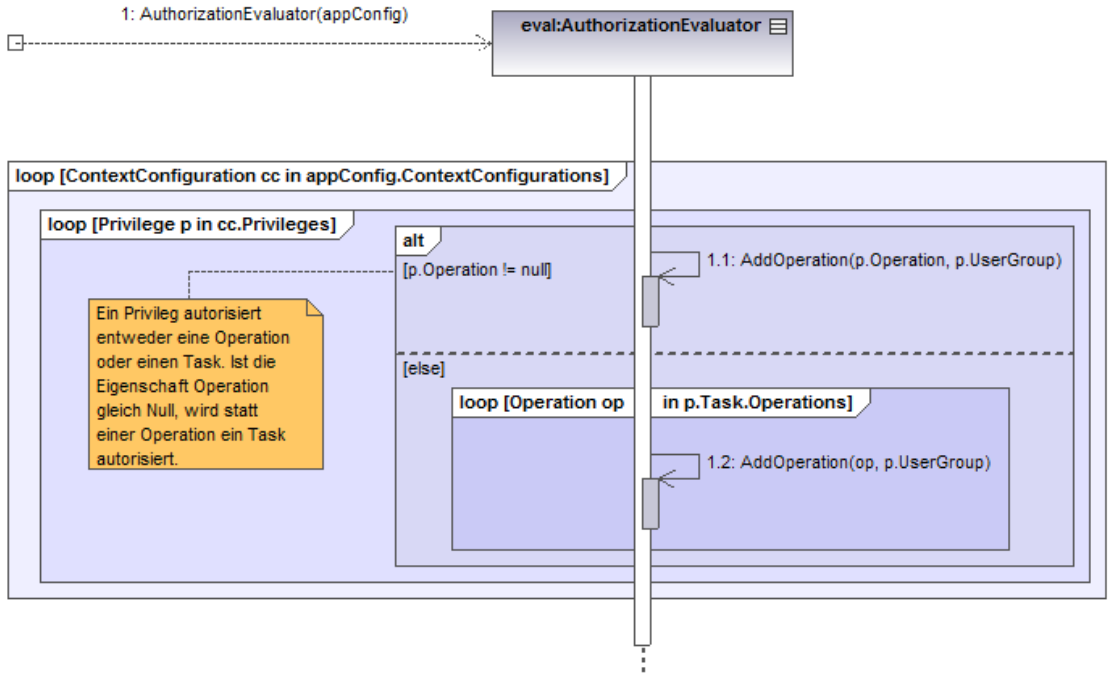

Abbildung 52 Sequenzdiagramm: Initialisierung des AuthorizationEvaluator

1. Der Konstruktor des *AuthorizationEvaluator*, dargestellt in *Abbildung 52*, wird aufgerufen. Da der *AuthorizationEvaluator* eine interne Klasse ist, ist der Web Service momentan die einzige Klasse, die den *AuthorizationEvaluator* verwendet. Eine anderweitige Verwendung in folgenden Versionen des verteilten Autorisierungssystems ist allerdings durchaus vorstellbar, wenn die Kommunikation über andere Technologien als Web Services erfolgt.

   1.1. Für jeden konfigurierten Kontext (*ContextConfiguration*) innerhalb der Anwendungskonfiguration werden alle Privilegien iteriert. Jedes Privileg enthält entweder eine Operation oder mehrere Operationen, je nachdem ob das Privileg eine Operation oder einen Task konfiguriert. Konfiguriert ein Privileg nur eine Operation wird diese eine Operation dem *AuthorizationEvaluator* mittels der Methode *AddOperation* hinzugefügt. Diese Methode wird weiter unten in diesem Abschnitt beschrieben.

1.2. Konfiguriert ein Privileg einen Task wird nochmals durch die Operationen dieses Tasks iteriert. Die einzelnen Operationen werden wiederum mittels der Methode *AddOperation* der lokalen Instanz hinzugefügt.

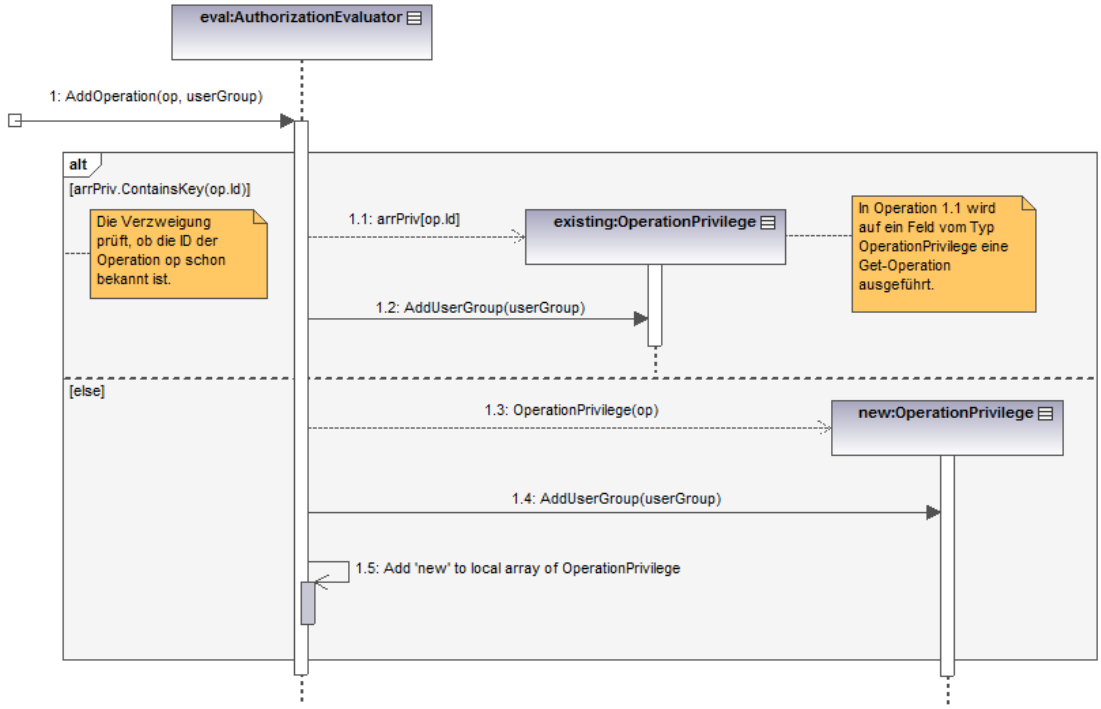

Abbildung 53 Sequenzdiagramm: AuthorizationEvaluator: Hinzufügen einer Operation

1. Die Methode *AddOperation*, dargestellt in *Abbildung 53*, wird während der Initialisierung der Klasse *AuthorizationEvaluator* für jede Operation innerhalb der Privilegien aufgerufen. Als Parameter werden die Operation und die Benutzergruppe, die entweder für diese Operation explizit autorisiert oder deren Autorisierung explizit verweigert ist, übergeben.

   1.1. Sofern die Operation bereits im *AuthorizationEvaluator* bekannt ist, wird der entsprechenden Instanz vom Typ *OperationPrivilege* nur die Benutzergruppe hinzugefügt.

   1.2. Wurden für die Operation bisher keine Privilegien verarbeitet, wird eine neue Instanz *new* vom Typ *OperationPrivilege* für die neue Operation erstellt, indem diese als Parameter beim Konstruktor übergeben wird.

   1.3. Der neuen *OperationPrivilege*-Instanz wird die Benutzergruppe *userGroup* hinzugefügt.

   1.4. Schließlich wird *new* einem lokalen Array im *AuthorizationEvaluator* hinzugefügt. Werden nochmals Privilegien für diese Operation konfiguriert, wird die soeben erstellte *OperationPrivilege*-Instanz verwendet.

Die eigentliche Überprüfung von Privilegien erfolgt in *OperationPrivilege*, dessen Instanzen für jede Operation im *AuthorizationEvaluator* in einem Array vorliegen.

1. Der Web Service ruft die Methode *CheckAccess* des *AuthorizationEvaluator* auf, dargestellt in *Abbildung 54*. Dieser hält ein Array von *OperationPrivilege*-Instanzen.

   1.1. Da für jede konfigurierte Operation genau eine *OperationPrivilege*-Instanz vorliegt, kann diese eindeutig über die ID der Operation gesucht werden.

   1.2. Die Autorisierungsanfrage *request* wird an die im Schritt 1.1 ermittelte *OperationPrivilege*-Instanz weitergeleitet.

1.2.1. Als erstes wird überprüft, ob der Benutzer, für den die Autorisierung durchgeführt wird, in einer der Negativlisten verzeichnet ist. Verbote haben eine höhere Priorität als Erlaubnisse, daher werden die Negativlisten zuerst überprüft.

1.2.2. Wurde der Benutzer in einer Negativliste gefunden, wird eine negative Autorisierungsantwort erstellt und an den Aufrufer *eval* zurückgegeben.

1.2.3. Wurde der Benutzer in keiner Negativliste gefunden, werden die Positivlisten überprüft.

1.2.4. Wurde der Benutzer in einer Positivliste gefunden, wird eine positive Autorisierungsantwort erstellt.

1.2.5. Wurde der Benutzer in keiner Positivliste gefunden, ist keine Autorisierung für diesen Benutzer konfiguriert worden. In diesem Fall ist die Autorisierung nicht definiert und es wird eine negative Autorisierungsantwort erstellt.

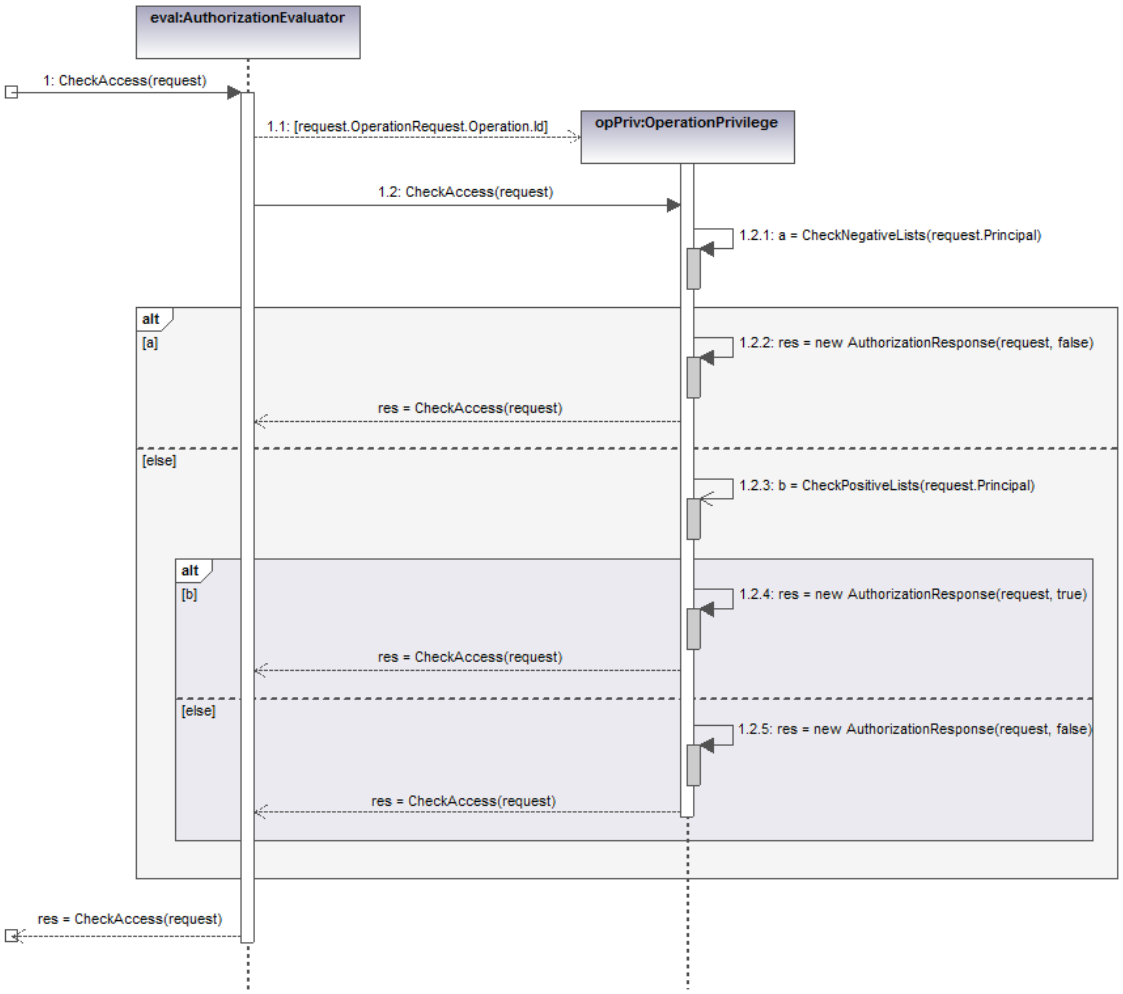

Abbildung 54 Sequenzdiagramm: Überprüfung von Privilegien

3.5 Schemata der XML-Daten

3.5.1 Anwendungsdefinitionen

Das XML-Schema für die Anwendungsdefinition gibt die Struktur einer Anwendung an, die für die Zugriffskontrolle konfiguriert ist. Wurzel der Anwendungsdefinition ist das Element *application*, das genau ein Element des Typs *context* enthält, der den initialen Kontext darstellt. Siehe dazu auch *Abbildung 55*.

Das Wurzelelement *application* enthält die Attribute *id*, *name* und *version*, die in der Attributgruppe *definitionMetadata* zusammengefasst sind. Die Verwendung einer Attributgruppe ist deswegen sinnvoll, da die gleichen Attribute auch von anderen Elementen verwendet werden. Insbesondere während des Entwurfs der Schemata ändern sich die Anforderungen an die Metadaten der Elemente. Bei der Verwendung von Attributgruppen ist die Änderung nur bei der Definition der Attributgruppe notwendig. Alle globalen Typen verwenden die Attributgruppe *definitionMetadata*.

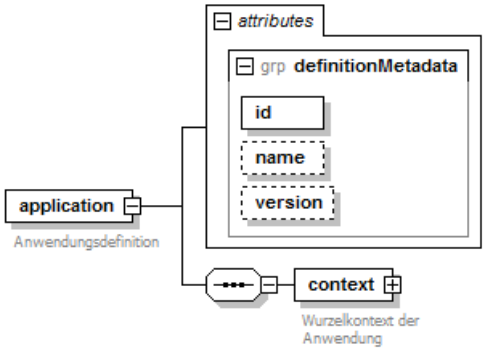

Abbildung 55 Schema für die Anwendungsdefinition

Der Kontext ist vom Typ *ContextType* (*Abbildung 56*), der wiederum weitere Kontexte enthalten kann. Wesentlich für einen Kontext sind die eingebetteten Kontexte, Operationen und Tasks. Die beiden letztgenannten werden im Folgenden separat beschrieben.

Kontexte, Operationen und Tasks sind in Listen eingebettet, die jeweils optional sind. Sind sie allerdings vorhanden müssen sie mindestens ein Element enthalten und können beliebig viele enthalten.

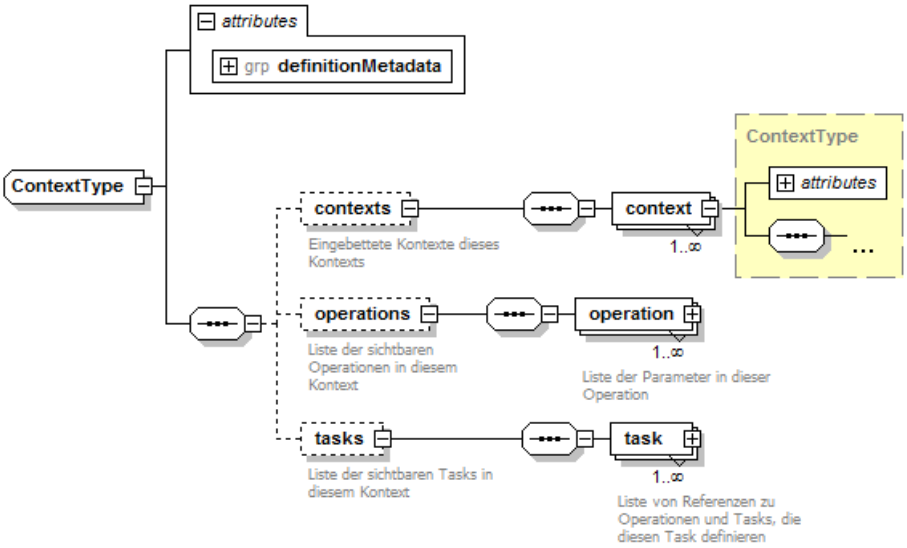

Abbildung 56 Schema eines Kontexts innerhalb einer Anwendungsdefinition

Operationen sind Elemente vom Typ *OperationType* (*Abbildung 57*). Operationen enthalten Parameter von beliebiger Anzahl. Parameter sind lokal definiert, also direkt innerhalb der Definition von *OperationType* und enthalten ebenfalls die Attributgruppe *definitionMetadata* und den Typ eines Parameters als String.

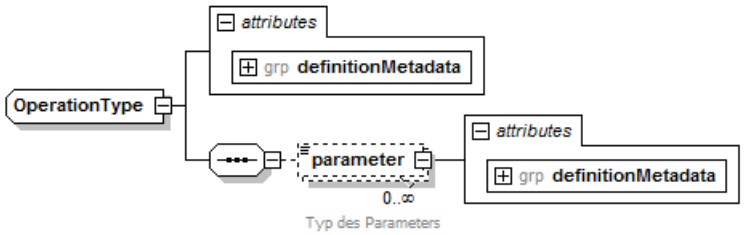

Abbildung 57 Schema einer Operation innerhalb einer Anwendungsdefinition

Tasks, dargestellt in *Abbildung 58*, sind Gruppierungen von Operationen und Tasks. Normalerweise werden Tasks erst in der Anwendungskonfiguration definiert, allerdings können in der Anwendungsdefinition Vorlagen für Tasks mitgegeben werden. Operationen können in mehreren Tasks verwendet werden. Als Referenz wird die ID der Operationen und Tasks verwendet.

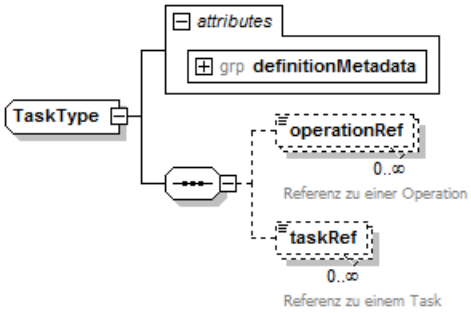

Abbildung 58 Schema eines Tasks innerhalb einer Anwendungsdefinition

3.5.2 Anwendungskonfigurationen

Eine Anwendungskonfiguration basiert auf einer Anwendungsdefinition und definiert Autorisierungen für Benutzergruppen auf Operationen und Tasks. Das Wurzelelement der Anwendungskonfiguration ist *configuration*. Es enthält Attribute und Elemente für die Konfiguration von Kontexten, Benutzergruppen und der Referenz zur Anwendungsdefinition. Die Elemente *context* und *userGroup* verwenden globale Typen und werden weiter unten in diesem Abschnitt beschrieben. In *Abbildung 59* wird das Element *configuration* mit seinen untergeordneten Elementen dargestellt.

Ähnlich dem Schema für Anwendungsdefinitionen gibt es auch hier eine Attributgruppe *definitionMetadata*, die häufig gebrauchte Attribute zusammenfasst. Sie ist nicht identisch mit der gleichnamigen Attributgruppe für Anwendungsdefinitionen.

Zusätzlich enthält das Element *configuration* das Attribut *revision*. Dieses Attribut ist vom Typ Ganzzahl und wird bei jeder gespeicherten Änderung inkrementiert. Die beiden Listen *contexts* und *userGroups* enthalten Elemente der globalen Typen *ContextType* und *UserGroupType* und müssen jeweils mindestens ein Element enthalten.

Das Element *applicationDefinition* ist ein lokaler Typ innerhalb von *configuration* und enthält Angaben, die die verwendete Anwendungsdefinition referenzieren. Dazu gehören die ID und die Revision der Anwendungsdefinition und der Pfad zu ihr. Wird der Pfad nicht angegeben, muss die Anwendungsdefinition anderweitig lokalisierbar sein.

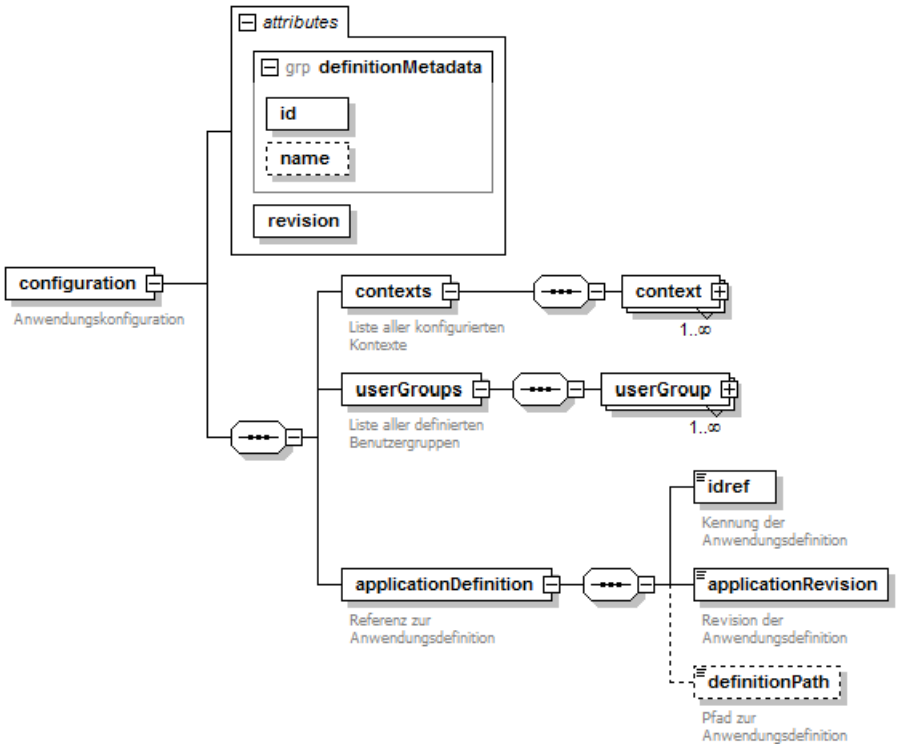

Abbildung 59 Schema für die Anwendungskonfiguration

Der globale Typ *ContextType*, dargestellt in *Abbildung 60*, enthält die Konfiguration für einen Kontext. Das Attribut *contextId* ist die Referenz zu einem Kontext, der innerhalb einer Anwendungsdefinition definiert wurde. Das Attribut *revision* ist ein Revisionszähler, der bei jeder gespeicherten Änderung eines Kontextes inkrementiert wird.

Die beiden Listen *tasks* und *privileges* enthalten Elemente der globalen Typen *TaskType* und *PrivilegeType*. Die Listen sind optional. Werden sie allerdings verwendet, müssen sie jeweils mindestens ein Element enthalten. Die definierten Tasks für einen Kontexts setzen sich aus den Tasks zusammen, die in der Anwendungsdefinition und in der Anwendungskonfiguration definiert wurden.

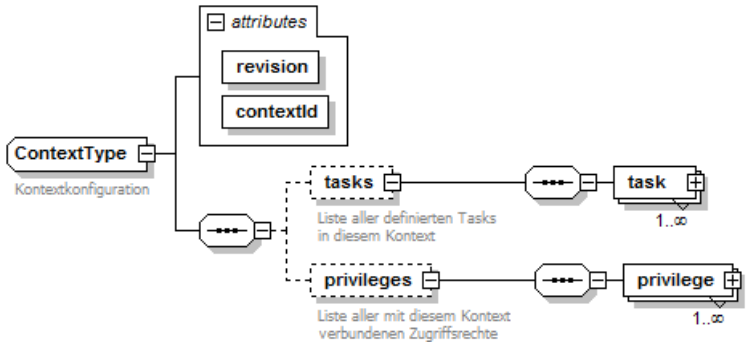

Abbildung 60 Schema eines Kontexts innerhalb einer Anwendungskonfiguration

Der globale Typ *TaskType*, dargestellt in *Abbildung 61*, ist mit dem gleichnamigen Typ aus der Anwendungsdefinition mit der Ausnahme identisch, dass der Typ *TaskType* in der Anwendungskonfiguration das Attribut *version* nicht aufweist.

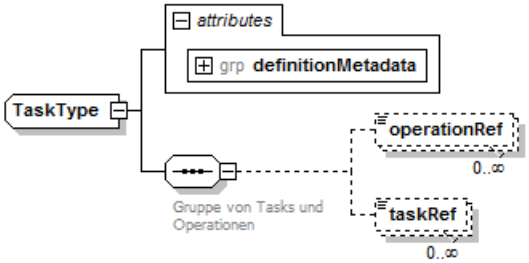

Abbildung 61 Schema eines Tasks innerhalb einer Anwendungskonfiguration

Der globale Typ *PrivilegeType* (*Abbildung 62*) speichert die Referenz zu entweder einer Operation oder einem Task und eine Referenz zu einer Benutzergruppe. Damit konfiguriert dieser Typ eine Autorisierung, da für eine Benutzergruppe festgelegt wird, auf welche Operation oder auf welchen Task diese Benutzergruppe zugreifen darf.

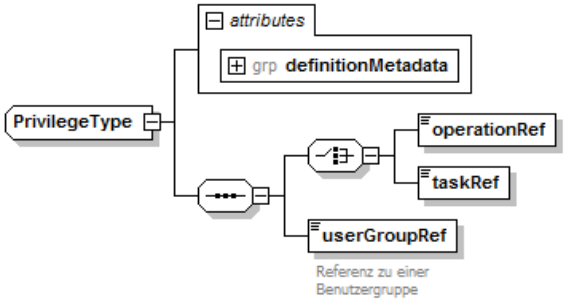

Abbildung 62 Schema eines Privilegs innerhalb einer Anwendungskonfiguration

Der globale Typ *UserGroupType*, dargestellt in *Abbildung 63*, dient der Definition von Benutzergruppen. Eine Benutzergruppe kann durch Identitäten, Rollen und LDAP-Anfragen definiert werden. Dazu enthält *UserGroupType* die optionalen Listen *identities*, *roles* und *queries*, die bei Verwendung jeweils mindestens ein Element enthalten müssen.

Innerhalb von *UserGroupType* gibt es den lokalen Typ *query*, der der Definition einer LDAP-Anfrage dient. Das Element *query* selbst enthält dabei die Abfrage. Über den Parameter *ldapServer* wird der LDAP-Server definiert, der abgefragt wird.

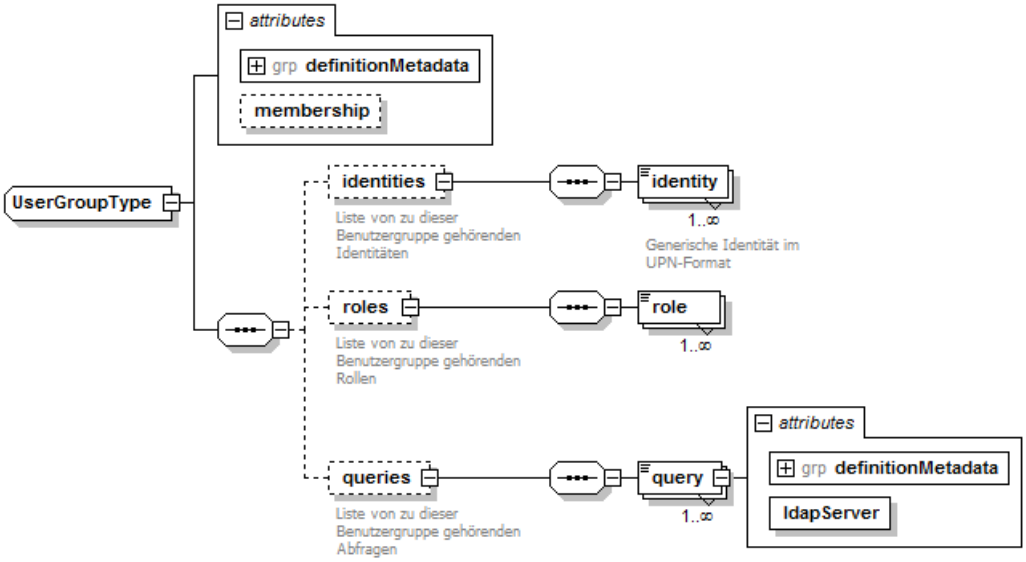

Abbildung 63 Schema einer Benutzergruppe innerhalb einer Anwendungskonfiguration

4 Realisierung

Dieses Kapitel beschreibt alle wesentlichen Punkte der Realisierung des verteilten Autorisierungssystems. Dazu gehören die Entwicklungsumgebung, Hilfsprogramme und Besonderheiten, die bei der Implementierung zu beachten waren.

4.1 Plattform

Das verteilte Autorisierungssystem verwendet als Ausführungsumgebung das Microsoft Framework .NET 2.0. Dieses Framework besteht aus umfangreichen Klassenbibliotheken (Base Class Library, BCL) und ist gleichzeitig auch die Ausführungsumgebung (Runtime) für Programme, die auf .NET basieren. .NET-Programme liegen in der Zwischensprache Microsoft Intermediate Language (MSIL) vor. Zur Ausführungszeit wird diese Zwischensprache Methode für Methode in Maschinencode kompiliert und für das ausführende Computersystem optimiert. Die kompilierten Methoden werden im Arbeitsspeicher bewahrt, bis die Anwendung beendet wird.

Für Systeme, die bereits ab der ersten Ausführung höchst performant sein müssen, ist es möglich, die Kompilierung in Maschinencode während der Installation des Programms durchzuführen. Da es sich bei dem verteilten Autorisierungssystem um eine prototypische Implementation handelt und Leistung in dieser Arbeit eine untergeordnete Rolle spielt, wurde von dieser Möglichkeit kein Gebrauch gemacht.

4.2 Persistierung

Das verteilte Autorisierungssystem nutzt XML-Dateien, um die Definitionen und Konfigurationen von Anwendungen zu speichern. Die XML-Daten sind durch XML-Schemata strukturiert. Das heißt, die Gültigkeit einer XML-Datei kann programmatisch durch den Verweis auf ein Schema geprüft werden.

Die Erstellung von XML-Schemata selbst ist aufwändig, da hierfür ein fundiertes Verständnis von XML notwendig ist. An dieser Stelle sei das Programm Altova XMLSpy 2007 zu erwähnen, dass während dieser Arbeit bei der Erstellung von XML-Schemata gute Dienste geleistet hat. Mit diesem Programm ist es möglich, XML-Schemata grafisch zu modellieren und Anforderungen an die Schemata durch das Setzen von Eigenschaften zu erfüllen. Eine genaue Kenntnis der Syntax und Semantik von XML-Schemata ist dabei nicht notwendig. Beispiele für die grafische Modellierung sind im *Abschnitt 3.5 Schemata der XML-Daten* zu finden. Die Abbildungen in diesem Abschnitt entsprechen der Ansicht bei der Modellierung von XML-Schemata.

Nachdem die XML-Schemata erstellt worden sind, ist es empfehlenswert Klassen zu erstellen, die genau den XML-Schemata entsprechen. Das .NET Framework bietet zwar auch die Möglichkeit unbekannte XML-Dateien auszulesen, allerdings ist die Programmierung wesentlich einfacher und zuverlässiger, wenn entsprechende Klassen verwendet werden. Denn unbekannte XML-Dateien müssen Element für Element mit den richtigen Bezeichnern für Elemente und Attribute ausgelesen werden. Diese Vorgehensweise ist durch den größeren Implementierungsaufwand aufwändig und zudem fehleranfälliger, da Elemente und Attribute von Hand angesprochen werden müssen.

Die Erstellung von Klassen, die XML-Schemata entsprechen, wäre selbst auch sehr aufwändig, wenn dafür nicht entsprechende Hilfsprogramme existieren würden. XMLSpy bietet die Möglichkeit aus XML-Schemata Quellcode zu erzeugen. Ich habe mich allerdings für das Hilfsprogramm XSD (Xml Schemas/DataTypes support utility) entschieden, das im .NET Framework mitgeliefert wird, das bei der Erstellung von Klassen aus XML-Schemata behilflich

ist. Bei diesem Hilfsprogramm handelt es sich um eine Konsolenanwendung. Für die Erstellung von Klassen für das XML-Schema für die Anwendungsdefinition wird beispielsweise der folgende Aufruf verwendet.

xsd "Application Definition.xsd" /c /f
/n:Thesis.Authorization.XmlStorage.ApplicationDefinition

Ein Beispiel zeigt der kurze Ausschnitt im *Quellcode 1*. Der Parameter *c* gibt an, dass Klassen für das XML-Schema erstellt werden sollen. Alternativ wäre es auch möglich, DataSets zu erstellen. Dies wäre dann von Vorteil, wenn auf die XML-Daten in einer Datenbank-ähnlichen Art und Weise zugegriffen werden sollte. In diesem Fall bietet sich der Zugriff in einer objektorientierten Weise besser an.

Der Parameter *f* im Aufruf sorgt dafür, dass XSD alle Elemente und Attribute als öffentliche Felder anstatt als Eigenschaften anlegt. Generell ist in der objektorientierten Programmierung zu Eigenschaften anstatt zu öffentlichen Feldern zu raten. Allerdings bieten Eigenschaften in diesem Kontext keinerlei Mehrwert und blähen die Klassen nur unnötig auf und verlangsamen die Ausführung beim Zugriff auf die Daten.

```
[System.CodeDom.Compiler.GeneratedCodeAttribute("xsd", "2.0.50727.42")]
[System.SerializableAttribute()]
[System.Diagnostics.DebuggerStepThroughAttribute()]
[System.ComponentModel.DesignerCategoryAttribute("code")]
[System.Xml.Serialization.XmlTypeAttribute(AnonymousType=true)]
[System.Xml.Serialization.XmlRootAttribute(Namespace="", IsNullable=false)]
public partial class application {

    /// <remarks/>
    public ContextType context;

    /// <remarks/>
    [System.Xml.Serialization.XmlAttributeAttribute()]
    public string id;

    /// <remarks/>
    [System.Xml.Serialization.XmlAttributeAttribute()]
    public string name;

    /// <remarks/>
    [System.Xml.Serialization.XmlAttributeAttribute()]
    public int version;

    /// <remarks/>
    [System.Xml.Serialization.XmlIgnoreAttribute()]
    public bool versionSpecified;
}
```

Quellcode 1 XSD-generierter Quellcode

Die Struktur, wie die Daten in XML gespeichert werden, ist zwar logisch für die Darstellung in persistierter Form, allerdings nicht optimal für den programmatischen Zugriff. Vor allem ist auch eine Entkopplung von den Klassen für die XML-Daten und den Klassen für die programminterne Repräsentation sinnvoll, da die Klassen, wie sie im Entwurf modelliert wurden, erweiterte Funktionalitäten gegenüber den XML-Klassen aufweisen. Zudem kann auf diese Weise entweder der Programm-interne Ablauf oder die Darstellung in den XML-Daten geändert werden, ohne alle abhängigen Klassen ebenfalls ändern zu müssen.

Für die Überführung der Daten von den XML-Dateien in die Programmklassen habe ich Konverter geschrieben, die diese Aufgabe erledigen. In *Abbildung 64* sind diese Klassen dargestellt. *Abbildung 64* unterscheidet sich in der Visualisierung von den anderen Klassendiagrammen, da es sich hier um ein Klassendiagramm handelt, dass direkt in Visual Studio .NET

erstellt wurde. Diese Klassen wurden nicht vormodelliert wie all die Klassen in *Abschnitt 3.3 Modellierung der Klassen*.

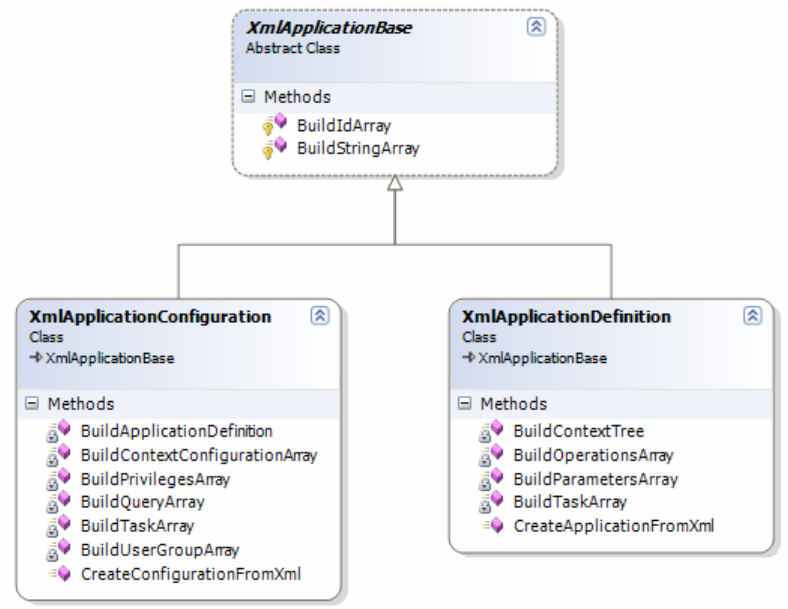

Abbildung 64 Klassendiagramm der XML-Konverterklassen

Die XML-Dateien werden sequenziell ausgelesen. Daher kann es vorkommen, dass nicht alle Referenzziele im Moment des Auslesens bekannt sind. Für diesen Fall werden die Klassen erstellt und es wird nur ein Platzhalter für die entsprechenden Referenzziele gespeichert. Nach dem Auslesen einer XML-Datei, wenn also alle Referenzziele bekannt sein sollten, werden alle Platzhalter aufgelöst und stattdessen echte Objektverweise gesetzt.

Die *Find*-Methoden in den Klassen *ApplicationConfiguration*, *Context* und *Operation* werden während des Auflösungsprozesses verwendet, um die jeweiligen Kontexte, Operationen, Tasks, Parameter und Benutzergruppen zu finden.

Kann während des Auflösungsprozesses eine Referenz nicht gefunden werden, wird der gesamte Prozess als Fehlschlag abgebrochen. Dieser Fall kann eintreten, wenn Bezeichner (in der Regel GUIDs) in der XML-Datei fehlerhaft sind.

Abbildung 65 zeigt die Auflösung der Referenzen für die Anwendungsdefinition.

1. In *Context* wird die Methode *Resolve* von *XmlApplicationDefinition* oder durch einen rekursiven Aufruf von *Context* aufgerufen.

   1.1. Es werden alle vordefinierten Tasks in *context* iteriert und für jeden die Methode *Resolve* aufgerufen. Jeder Task sucht seine referenzierten Operationen und untergeordnete Tasks und speichert die jeweiligen Objektreferenzen.

   1.2. Es werden alle untergeordneten Kontexte iteriert und für jeden die Methode *Resolve* aufgerufen. Jeder Kontext verfährt nach gleichem Schema für die Auflösung weiterer vordefinierter Tasks und untergeordneter Kontexte.

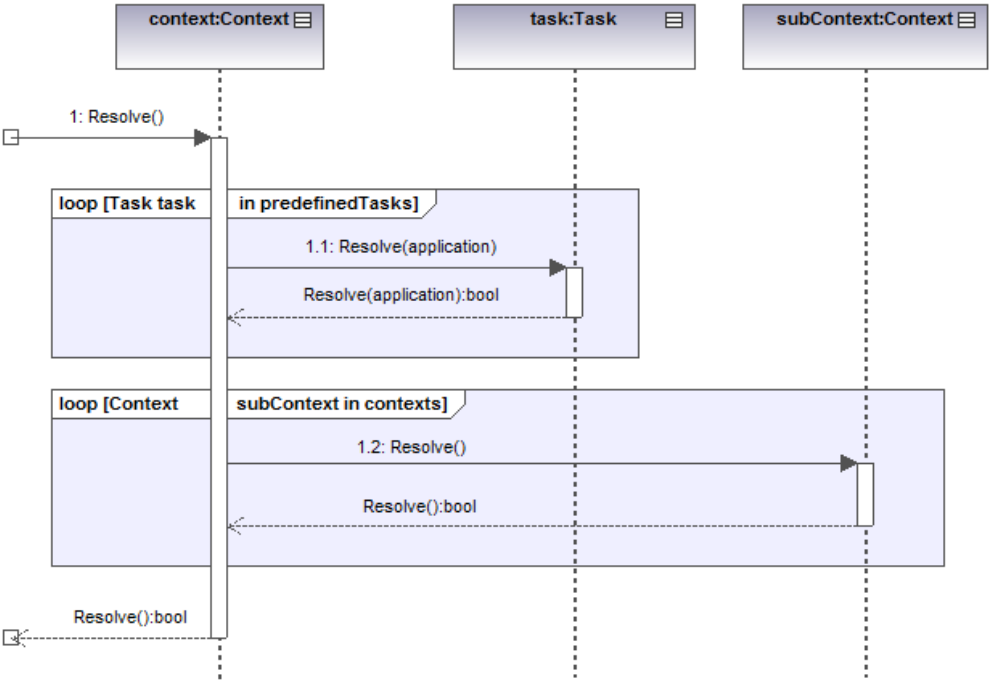

Abbildung 65 Auflösung von Referenzen von Anwendungsdefinitionen

Der Auflösungsprozess für Anwendungskonfigurationen ist dem für Anwendungsdefinitionen ähnlich. Auf eine detaillierte Beschreibung für den Prozess in *Abbildung 66* wird hier daher verzichtet. Die wesentlichen Unterschiede sind, dass die Kontexte in einem flachen Array gespeichert sind und daher keine rekursiven Aufrufe nötig sind. Jede *ContextConfiguration* löst seine Tasks und Privilegien auf.

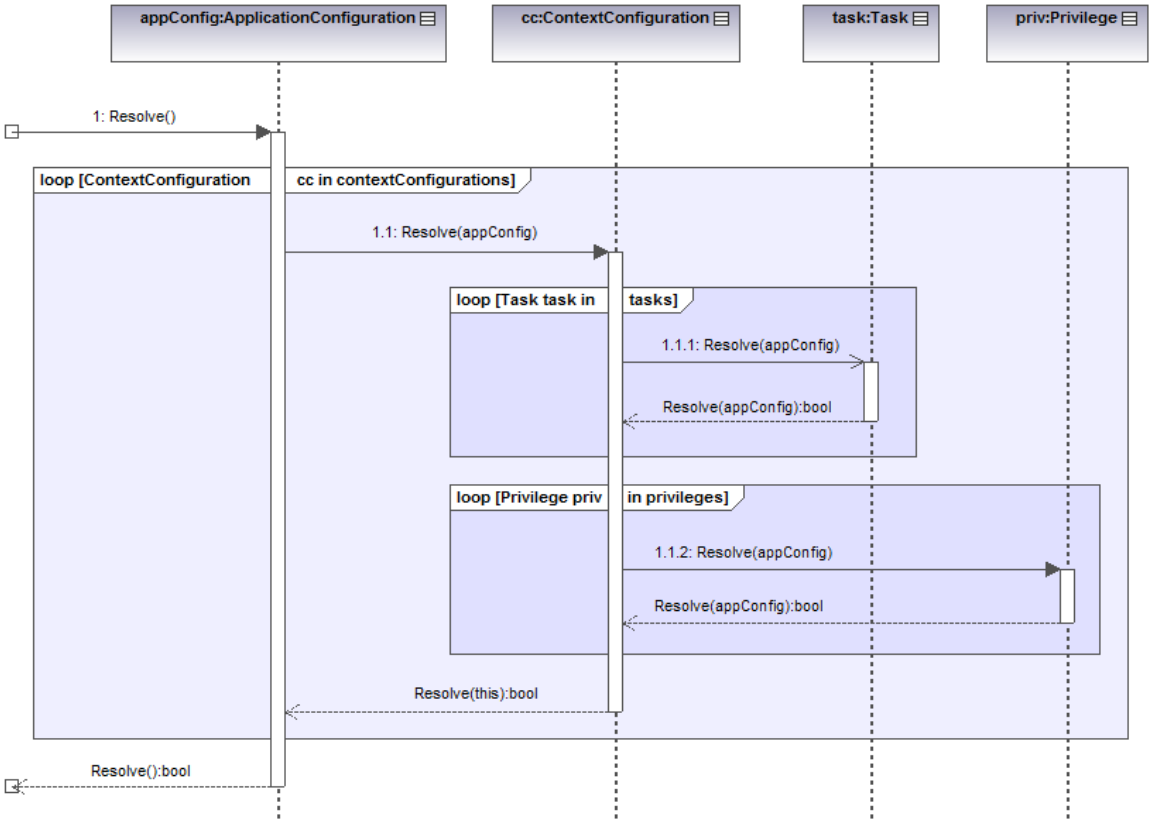

Abbildung 66 Auflösung von Referenzen von Anwendungskonfigurationen

4.3 Kommunikation

Die Kommunikation geschieht im verteilten Autorisierungssystem über Web Services. In .NET können aus Klassen einfach Web Services erstellt werden, indem Klassen von der Basisklasse *System.Web.Services.WebService* und mit dem Attribut *WebService(...)* und Methoden mit *WebMethod* deklariert werden. *Quellcode 2* zeigt die Verwendung bei der Klasse *AuthorizationWebService*.

Deklarative Programmierung ist ein Funktionsmerkmal von .NET und ein Sprachmerkmal von einigen .NET-Sprachen wie C#, VB.NET und J#. Das bedeutet, das Konzept der deklarativen Programmierung ist direkt in .NET vorgesehen, muss allerdings von den einzelnen Sprachen unterstützt werden.

```
[WebService(Namespace = "http://thesis.peterkirchner.de")]
[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
public class AuthorizationWebService : System.Web.Services.WebService {
        // Class implementation omitted here.
}
```

Quellcode 2 Web Service: Deklaration

Wie in *3.3.4 Namensraum Server* bereits geplant wurde, besitzt der Web Service zwei Methoden: *CheckAccess* und *GetApplicationDefinition*. Der Ablauf der Methode *CheckAccess* wurde bereits im *Abschnitt 3.4.3 Initialisierung des Web Services und Autorisierungsanfragen* vorgestellt. In *Quellcode 3* ist die Implementierung zu sehen, die durch den Ablauf, wie er in *Abbildung 51* dargestellt wurde, direkt nachzuvollziehen ist.

```
[WebMethod]
public AuthorizationResponseTransport CheckAccess(AuthorizationRequestTransport
request) {
        // Get true authorization request from the transport class.
        AuthorizationRequest nativeRequest =
                new AuthorizationRequest(appConfig.Application, request);

        // Forward access check.
        AuthorizationResponse nativeResponse =
                eval.CheckAccess(nativeRequest);

        // Pack true authorization response into transport class.
        AuthorizationResponseTransport response =
                new AuthorizationResponseTransport(nativeResponse);

        return response;
}
```

Quellcode 3 Web Service: CheckAccess

Die Klassen, wie sie in *3.3.1 Namensraum Common* modelliert wurden, können nicht direkt für den Web Service eingesetzt werden, da das Framework von .NET für Web Services nur öffentliche Felder von Klassen serialisieren kann. Zudem müssen diese Klassen einen Standardkonstruktor besitzen.

Es gab zwei Lösungsmöglichkeiten für dieses Problem: entweder die Klassen aus dem Namensraum *Common* entsprechend den Bedingungen für Web Services in .NET anpassen oder neue Transportklassen erstellen, die nur dem Zweck dienen, für den Web Service eingesetzt zu werden. Ich habe mich für die zweite Alternative entschieden, da die Bedingungen für Web Services in .NET keinen Einfluss auf die Modellierung der Klassen für das verteilte Autorisierungssystem haben sollten.

Vor der Kommunikation mittels des Web Service werden die Daten in die Transportklassen kopiert und nach der Kommunikation wieder daraus ausgelesen und in die eigentlichen Klas-

sen des Frameworks des verteilten Autorisierungssystem gespeichert. Die Transportklassen sind denen aus dem Namensraum *Common* sehr ähnlich. Alle Eigenschaften sind öffentliche Felder. Jede Klasse besitzt einen Standardkonstruktor und einen Kopierkonstruktor, dem eine Instanz einer *Common*-Klasse übergeben werden kann. Die Transportklassen sind in *Abbildung 67* dargestellt.

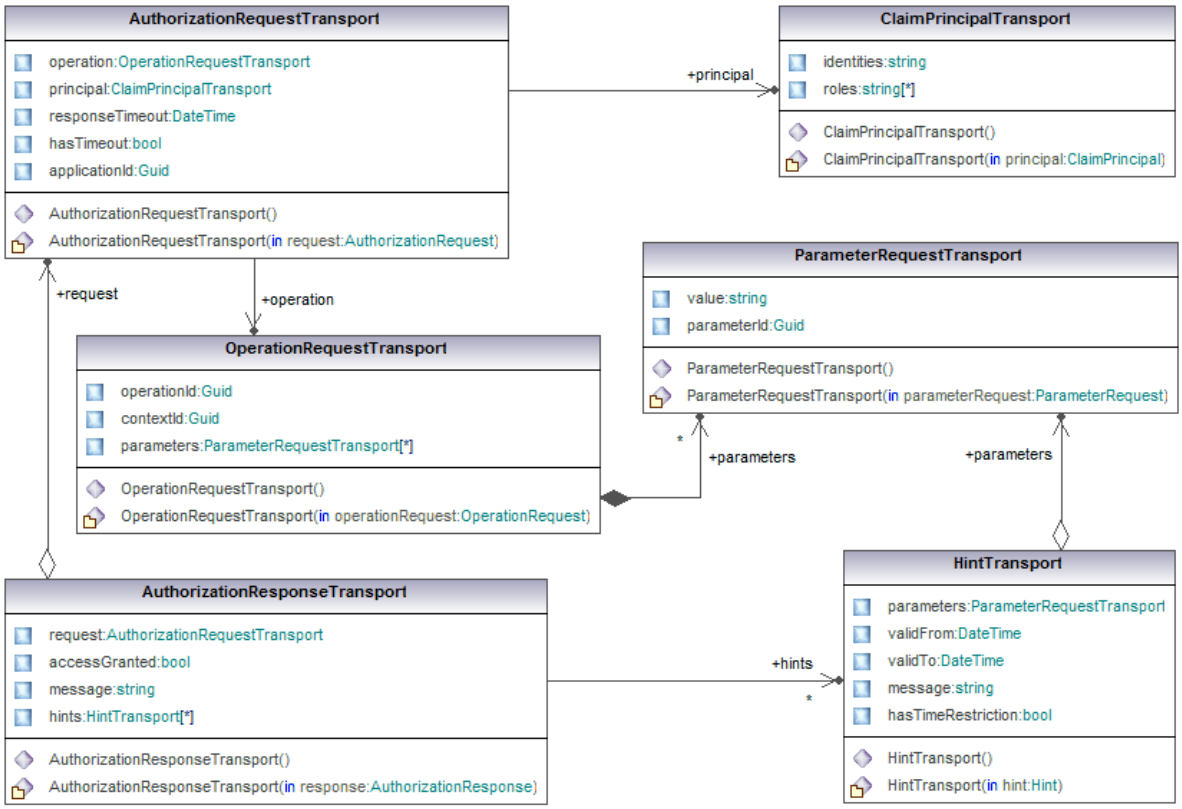

Abbildung 67 Transportklassen des Web Service

Quellcode 4 zeigt die Verwendung des Web Service. Der Ablauf entspricht dem Sequenzdiagramm in *Abbildung 49, Seite 63*. Es ist sehr anschaulich zu sehen, wie die Transportklassen eingesetzt werden und wie der Aufruf zum Web Service erfolgt.

```
public AuthorizationResponse CheckAccess(AuthorizationRequest request) {
        // Pre check the application id of the request.
        if (request.OperationRequest.Context.Application.Id != application.Id)
                throw new InvalidOperationException("The application ID of the request must
match the application ID of the authorization manager.");

        // Create transport data structure.
        AuthorizationRequestTransport transportRequest =
                new AuthorizationRequestTransport(request);

        // Forward the access check to the authorization server.
        AuthorizationResponseTransport transportResponse =
                webService.CheckAccess(transportRequest);

        // Create true Response instance.
        AuthorizationResponse response =
                new AuthorizationResponse(application, transportResponse);

        return response;
}
```

Quellcode 4 Nutzung des Web Service

Der Aufruf *webService.CheckAccess(transportRequest)* startet die Kommunikation mit dem Web Service. Die eigentliche Kommunikation mit dem Web Service wird vom .NET Framework übernommen.

4.4 Authentifizierung

Die Authentifizierung ist Aufgabe der Infrastruktur und des SSO-Systems, in dem sich eine Anwendung befindet. Das verteilte Autorisierungssystem überprüft lediglich, ob ein bestimmter Nutzer eine bestimmte Operation ausführen darf. Es gibt drei Varianten, wie in einer ADFS-Umgebung mit .NET ein Benutzerobjekt erstellt werden kann.

1. Im Code wird ein Benutzerobjekt mit vorgegeben Benutzerdaten erstellt, ADFS kommt dabei nicht zum Einsatz. Diese Vorgehensweise ist sinnvoll, wenn Aufgaben losgelöst von tatsächlichen Benutzerkonten ausgeführt werden müssen. Diese Vorgehensweise wird auch als Impersonation bezeichnet.
2. Die Benutzerdaten werden vom Windows-Sicherheitssystem gelesen und verwendet. Jeder Benutzer besitzt unter Windows ein Sicherheitstoken, das Auskunft über den Benutzernamen, die Domäne und die Sicherheitsgruppen gibt.
3. Die Benutzerdaten werden vom ADFS-System in so genannten Claims (siehe auch *Abschnitt 1.4.2 Active Directory Federation Services*) bereitgestellt. Diese Variante ist die einzige, die die Verwendung von Benutzerkonten über Domänengrenzen hinweg ermöglicht.

In .NET kann der aktuelle Benutzer in Web-Anwendungen über die Eigenschaft *User.Identity* ermittelt werden, die eine Instanz mit dem Interface *IIdentity* zurückgibt. Sofern die Web-Anwendung ADFS für die Authentifizierung verwendet, kann diese Instanz auf den Typ *SingleSignOnIdentity* konvertiert werden und bietet damit Zugriff auf Benutzerdaten, die über ADFS bereitgestellt wurden. Welche Informationen in diesen Klassen über den Benutzer gespeichert sind, kann im *Abschnitt 5.4 Test der Authentifizierung* nachgelesen werden.

4.4.1 Konfiguration für ADFS

Damit ADFS in der Web-Anwendung verfügbar ist, müssen einige Konfigurationen vorgenommen werden, die im Folgenden in diesem Abschnitt beschrieben werden. ADFS benötigt einen eigenen Konfigurationsabschnitt innerhalb der Anwendungskonfiguration. Hierbei darf diese Anwendungskonfiguration, die Bestandteil des Konzepts von .NET ist, nicht mit den Anwendungskonfigurationen verwechselt werden, die vom verteilten Autorisierungssystem verwendet werden.

Quellcode 5 zeigt die Definition eines zusätzlichen Konfigurationsabschnitts. Es ist zu sehen, dass ein neuer Abschnitt *websso* innerhalb von *system.web* erstellt wird. Die Klasse *WebSsoConfigurationHandler* ist für die Auswertung der Einstellungen in diesem Abschnitt verantwortlich.

```
<configSections>
  <sectionGroup name="system.web">
    <section name="websso"
             type="System.Web.Security.SingleSignOn.WebSsoConfigurationHandler,
                   System.Web.Security.SingleSignOn, Version=1.0.0.0,
                   Culture=neutral, PublicKeyToken=31bf3856ad364e35,
                   Custom=null" />
  </sectionGroup>
</configSections>
```

Quellcode 5 Konfiguration für ADFS: Zusätzlicher Konfigurationsabschnitt

Der Konfigurationsabschnitt *websso*, in *Quellcode 6* dargestellt, enthält Einstellungen die vom ADFS-Framework verwendet werden. Die beiden wichtigsten Einstellungen sind *returnurl* und *fs*. *returnurl* enthält den öffentlichen Pfad zu der Web-Anwendung selbst. Dieser Pfad ist notwendig, da ADFS nach der Authentifizierung des Benutzers eine Weiterleitung zu dieser Adresse durchführt. Es ist Voraussetzung, dass es sich hier um eine HTTPS-Adresse handelt. *fs* enthält die Adresse zum ADFS-Dienst. Der ADFS-Dienst ist selbst als Web-Anwendung implementiert, daher ist die vollständige Adresse zu dem Dienst notwendig – die Angabe der Serveradresse reicht nicht aus.

Darüber hinaus wird die Web-Anwendung von ADFS anhand der *returnurl* identifiziert. Bei der Einstellung der Applikation für ADFS muss dieselbe Adresse angegeben werden. Stimmen die Adressen in der Konfiguration der Web-Anwendung und in ADFS nicht überein, wird die Web-Anwendung nicht erkannt. Dies hat zur Folge, dass die Web-Anwendung nicht die konfigurierten Claims des Benutzers erhält.

Dieser Fall war auch bei meinen Tests bei der Entwicklung des verteilten Autorisierungssystems aufgetreten. In der Anwendungskonfiguration hat der Port gefehlt und die Web-Anwendung hat keine korrekten Benutzerdaten von ADFS erhalten.

```
<websso>
  <authenticationrequired />
  <eventloglevel>55</eventloglevel>
  <auditsuccess>2</auditsuccess>
  <urls>
    <returnurl>https://adfsweb.treyresearch.net:8001/</returnurl>
  </urls>
  <cookies writecookies="true">
    <path>/</path>
    <lifetime>240</lifetime>
  </cookies>
  <fs>https://adfsresource.treyresearch.net/
      adfs/fs/federationserverservice.asmx</fs>
</websso>
```

Quellcode 6 Konfiguration für ADFS: Konfiguration der Web-Anwendung

Die Authentifizierung von ADFS erfolgt vor dem eigentlichen Zugriff auf die Web-Anwendung. Damit der Zugriff auf die Web-Anwendung abgefangen werden kann, wird ein zusätzliches HTTP-Modul für die Web-Anwendung eingetragen, wie in *Quellcode 7* abgebildet. Die Klasse *WebSsoAuthenticationModule* übernimmt diese Funktion.

Ohne dieses HTTP-Modul werden Benutzer nicht von ADFS authentifiziert und es stehen innerhalb der Web-Anwendung keine Benutzerdaten von ADFS zur Verfügung. Dieser Umstand war für die Entwicklung und die Testläufe hilfreich, da durch das Auskommentieren dieses HTTP-Moduls, die Web-Anwendung ohne die ADFS-Infrastruktur getestet werden konnte. Wie im *Abschnitt 5.1* zu lesen ist, ist die Einrichtung der ADFS-Infrastruktur relativ aufwändig und ist deswegen nur in der Testumgebung eingerichtet worden – allerdings nicht auf dem Entwicklungssystem.

```
<httpModules>
  <add
      name="Identity Federation Services Application Authentication Module"
      type="System.Web.Security.SingleSignOn.WebSsoAuthenticationModule,
            System.Web.Security.SingleSignOn, Version=1.0.0.0,
            Culture=neutral, PublicKeyToken=31bf3856ad364e35,
            Custom=null" />
</httpModules>
```

Quellcode 7 Konfiguration für ADFS: Einbinden eines zusätzlichen HTTP-Moduls

4.4.2 Programmatischer Zugriff auf ADFS

In der Web-Anwendung des Beispielszenarios wird eine zentrale Funktion *CreateClaimPrincipal* verwendet, die die Benutzerdaten von ADFS auswertet und eine Instanz vom Typ *ClaimPrincipal* zurückliefert, die für eine Autorisierungsanfrage notwendig ist. Die Methode nimmt als Parameter ein Interface vom Typ *IPrincipal* an. Jede Webseite enthält die Eigenschaft *User*, die genau dieses Interface aufweist.

Im ersten Schritt wird versucht die Identität von *user* in *SingleSignOnIdentity* zu konvertieren, die von ADFS stammt und zusätzliche Benutzerdaten zur Verfügung stellen kann, die von ADFS übermittelt wurden. *SingleSignOnIdentity* besitzt eine Auflistung von Sicherheitseigenschaften, die mit dem aktuellen Benutzer verbunden sind. In diesem Fall interessieren uns nur die Sicherheitseigenschaften, die Auskunft über die Gruppenzugehörigkeit des Benutzers geben. Mithilfe dieser Gruppen wird eine neue *ClaimPrincipal*-Instanz erstellt.

```
private ClaimPrincipal CreateClaimPrincipal(IPrincipal user) {
        // Get SSO identity if possible.
        SingleSignOnIdentity ssoId = user.Identity as SingleSignOnIdentity;

        if (ssoId != null) {
                // Build array of groups.
                StringCollection sc = new StringCollection();

                foreach (SecurityProperty secProp
                                in ssoId.SecurityPropertyCollection)
                        if (secProp.ClaimType == WebSsoClaimType.Group)
                                sc.Add(secProp.Value);

                string[] roles = new string[sc.Count];
                sc.CopyTo(roles, 0);

                // Create claim principal.
                ClaimPrincipal cp = new ClaimPrincipal(ssoId, roles);

                return cp;
```

Quellcode 8 Auswertung einer SSO-Identität von ADFS

Steht ADFS nicht zur Verfügung, kann *user* nicht in *SingleSignOnIdentity* konvertiert werden. In einer Produktivumgebung sollte nun eine entsprechende Fehlermeldung angezeigt werden. Allerdings bietet sich dieser Umstand, wie bereits im vorherigen Abschnitt beschrieben, an, die Anwendung gezielt zu testen (siehe *Quellcode 9*).

```
        } else {        // SSO not available.

                // In real environment raise error.
                // For purpose of testing create static claim principal.
                ClaimPrincipal cp = new ClaimPrincipal(
                        new GenericIdentity(@"kirchner\peter"),
                        new string[] { "Thesis Students Claim" });

                return cp;
        }
}
```

Quellcode 9 Fortsetzung von Quellcode 8: Statischer ClaimPrincipal für Testzwecke

4.5 Autorisierung

Die Autorisierung wird anhand der Implementation des Beispielszenarios beschrieben. *Quellcode 10* zeigt die Methode, die bei der ersten Verwendung der Web-Anwendung aufgerufen wird. Die Adresse zum Autorisierungsserver und die ID der verwendeten Konfiguration

werden aus einer Konfigurationsdatei geladen. *Quellcode 11* zeigt einen Auszug aus einer solchen Konfigurationsdatei.

Die Web-Anwendung erstellt beim Start einmalig eine Instanz des *AuthorizationManager*, an den später die Autorisierungsanfragen gerichtet werden. Bei der Instanziierung des *AuthorizationManager* werden die Adresse des Autorisierungsservers und die ID der Anwendungskonfiguration übergeben.

```
void Application_Start(object sender, EventArgs e)
{
    // Code that runs on application startup

    // Get authorization server.
    Uri server = new Uri(System.Web.Configuration.WebConfigurationManager
        .AppSettings["authorizationServer"]);

    // Get application ID;
    Guid applicationId =
        new Guid(System.Web.Configuration.WebConfigurationManager
                .AppSettings["applicationId"]);

    // Create an AuthorizationManager instance for all sessions.
    Thesis.Authorization.Client.AuthorizationManager am =
        new Thesis.Authorization.Client.AuthorizationManager(
             server, applicationId);

    // Put this instance into the application state collection.
    Application.Add("azMan", am);

}
```

Quellcode 10 Instanziierung des AuthorizationManager

Die Adresse zum Autorisierungsserver besteht aus dem Protokoll, dem Host, einem Port und dem Pfad zum Web Service. Statt HTTP sollte in einer Produktivumgebung das Protokoll HTTPS verwendet werden, da ansonsten vertrauliche Daten des Autorisierungsprozesses abgefangen oder manipuliert werden könnten. Die verwendeten Web Services des .NET Framework besitzen keinen Schutz für die Vertraulichkeit und Integrität der Kommunikation.

```
<appSettings>
  <add key="database" value="database.xml"/>
  <add key="authorizationServer"
       value ="http://localhost:2083/Server/Service.asmx"/>
  <add key="applicationId"
       value="{F089120A-DEA0-4a5e-B1E1-9B62FED5DC8C}" />
</appSettings>
```

Quellcode 11 Auszug aus der Konfigurationsdatei der Web-Anwendung

Quellcode 12 zeigt die Erstellung, Übermittlung und Auswertung einer Autorisierungsanfrage. Das Testanwendung verwendet eine zentrale Funktion *CheckAccess*, die einen booleschen Wert zurückgibt, ob der aktuelle Benutzer berechtigt ist, die spezifizierte Operation auszuführen.

Im ersten Schritt wird die Instanz des *AuthorizationManager* geladen, die beim Start der Web-Anwendung erstellt wurde. Danach werden Instanzen für die Operation und den Benutzer erstellt. Die Methode *CreateClaimPrincipal* wurde im vorherigen Abschnitt bereits vorgestellt. Mit den beiden Angaben über Operation und Benutzer kann eine Autorisierungsanfrage erstellt werden, die mittels des *AuthorizationManager* zum Autorisierungsdienst übermittelt wird. Die Eigenschaft *AccessGranted* der Autorisierungsantwort ist schließlich das Ergebnis der Funktion.

```
public static bool CheckAccess(IPrincipal user, Guid opId) {
        // 1. Get authorization manager.
        AuthorizationManager am = (AuthorizationManager)ApplicationState["azMan"];

        // 2. Get operation that has to be authorized.
        OperationRequest or = am.Application.RootContext
                .CreateOperationRequest(opId);

        // 3. Create a claim principal.
        ClaimPrincipal cp = CreateClaimPrincipal(user);

        // 4. Create authorization request.
        AuthorizationRequest ar = new AuthorizationRequest(cp, or);

        // 5. Transmit authorization requst to authorization manager.
        AuthorizationResponse response = am.CheckAccess(ar);

        return response.AccessGranted;
}
```

Quellcode 12 Erstellen einer Autorisierungsanfrage

Die Methode *CheckAccess* wird nicht direkt bei der Zugriffskontrolle aufgerufen, sondern über aussagekräftigere Methoden. Dieser Ansatz bietet sich an, da die Verwendung der GUIDs leicht zu Fehlern führen kann, da man ihnen die Operation nicht ansehen kann, für die sie stehen. Ebenso wie die Methode *CheckAccessViewCourses* in *Quellcode 13* existieren die Methoden *CheckAccessAttendCourse* und *CheckAccessCreateCourse*.

```
public static bool CheckAccessViewCourses(IPrincipal user) {
        // operation view courses
        return CheckAccess(user,
                new Guid("{18C9F0E3-AA77-4e19-ABF5-FEF1AFB9F705}"));
}
```

Quellcode 13 Verwendung der zentralen Autorisierungsmethode aus Quellcode 12

Quellcode 14 zeigt beispielhaft wie die *CheckAccess*-Methoden eingesetzt werden. *CheckAccessViewCourses* prüft, ob dem Benutzer Kurse angezeigt werden dürfen. Ist dies nicht der Fall, wird stattdessen eine Fehlermeldung angezeigt. *CheckAccessCreateCourse* prüft, ob der Benutzer Kurse erstellen darf. Hier ist gut zu sehen, dass das verteilte Autorisierungssystem derzeit nicht für Auditing-Zwecke eingesetzt werden kann, da Prüfungen auch im Vorfeld geschehen können.

```
protected void Page_Load(object sender, EventArgs e) {

        // Check authorization to view courses.
        if (Authorization.CheckAccessViewCourses(User)) {
                GridView1.DataSource = TestDatabase.DataSet.Courses;
                GridView1.DataBind();
        } else
                lblPermissionDenied.Visible = true;

        // View identity of the current user.
        lblAccount.Text = User.Identity.Name;

        // Disable create button if the user
        // has not the appropriate privileges.
        btnCreateCourse.Enabled = Authorization.CheckAccessCreateCourse(User);

}
```

Quellcode 14 Absicherung von kritischem Code

Obwohl im Vorfeld verhindert wurde, dass ein Benutzer Aktionen starten kann, für die er nicht autorisiert ist, ist es notwendig, auch die entsprechenden Funktionen auf Autorisierung zu überprüfen. Dies ist sinnvoll, da zum einen Fehler in der Vorüberprüfung nicht auszuschließen sind und zum anderen Angriffe auf die Webseite möglich sind. Es ist vorstellbar, dass ein Angreifer die Methode der Schaltfläche *btnCreateCourse* über einen manipulierten Aufruf versucht zu starten. *Quellcode 15* zeigt ein Beispiel für die erneute serverseitige Überprüfung der Autorisierung.

```
protected void btnCreateCourse_Click(object sender, EventArgs e) {
        // Double check if the user has the appropriate
        // privileges to execute this operation.
        if (Authorization.CheckAccessCreateCourse(User)) {
                // Create new data row for new course.
                TestDatabase.DataSet.Courses.AddCoursesRow(
                        txtName.Text, txtDescription.Text,
                        int.Parse(txtPoints.Text));

                // Save changes.
                TestDatabase.SaveDatabase();

                // Update data grid to display the new course.
                GridView1.DataBind();
        }
}
```

Quellcode 15 Serverseitige Prüfung von kritischem Code

5 Test & Bewertung

Dieser Abschnitt widmet sich der Überprüfung der Funktionalität des verteilten Autorisierungssystems. Das Autorisierungssystem kann nur innerhalb einer ADFS-Infrastruktur getestet werden. Daher wird zu Beginn beschrieben, wie eine solche Infrastruktur aufgebaut werden kann und was dabei zu beachten ist. Schließlich wird festgelegt, welche Bereiche der Software getestet werden und wie festgestellt werden kann, ob die Software erwartungsgemäß funktioniert.

5.1 Einrichtung einer Testumgebung für ADFS

Zweck dieser Testumgebung ist die Überprüfung der Funktionalität des Frameworks, das in dieser Arbeit erstellt wurde. Die gesamte Software, die zum Betrieb der Testumgebung notwendig ist, konnte ich kostenfrei aus der MSDN Academic Alliance des Hasso-Plattner-Instituts nutzen.

Für die Testsysteme wird Microsoft Virtual PC mit SP1 (Version 6.0.156.0) eingesetzt. Die Verwendung virtueller Maschinen hat mehrere Vorteile. Es wird nur ein leistungsfähiger PC benötigt, der so genannte Host, auf dem später die virtuellen Systeme, auch Gast-Betriebssysteme genannt, ausgeführt werden. Wichtigste Eigenschaften für den Host sind der Arbeitsspeicher und die Festplatten für die virtuellen Maschinen. Die CPU des Host-Systems ist weniger ausschlaggebend. Jedes Gastbetriebssystem besitzt virtuelle Festplatten. Jede virtuelle Festplatte wird in einer einzelnen Datei gespeichert, einem so genannten Image. Während des Testens sind bei den virtuellen Systemen die Undo Disks aktiviert. Undo Disks ermöglichen das Festhalten aller Änderungen seit dem letzten Starten einer virtuellen Maschine. Beim Herunterfahren einer virtuellen Maschine kann man entscheiden, ob die Änderungen beibehalten, gelöscht oder übernommen werden sollen.

Der Aufbau der Testumgebung folgt dem Aufbau wie er in [Pierson 2006] beschrieben ist. In diesem Szenario gibt es zwei fiktive Firmen *A. Datum* und *Trey Research*. Nutzer von *A. Datum* sollen auf Ressourcen (Web-Anwendungen) von *Trey Research* zugreifen können. Die individuellen, technischen Daten können der *Tabelle 7 auf Seite 104 im Anhang* entnommen werden.

Die Systeme *adfsweb*, *adfsaccount* und *adfsresource* benötigen die Internet Information Services (IIS). *adfsweb* benötigt den IIS für diverse Web-Anwendungen. Beim *adfsaccount* und *adfsresource* dagegen wird der IIS vom Federation Server von ADFS benötigt.

Zur Einrichtung des Federation Server ist das IIS 6.0 Resource Kit notwendig. Es enthält u.a. ein Tool zum Ausstellen und Zuweisen von selbst signierten SSL-Zertifikaten, die für den Betrieb von ADFS absolut notwendig sind.

Wie in *Tabelle 7* zu sehen ist, gibt es in dieser Testumgebung zwei Domänen, *adatum.com* und *treyresearch.net*. *adfsaccount* und *adfsresource* sind Domänencontroller und betreiben somit auch den Verzeichnisdienst Active Directory, der alle Benutzer- und Ressourcenkonten ihrer jeweiligen Domänen enthält.

5.2 Konfiguration des Beispielszenarios

Die Konfiguration von ADFS für die Testanwendung baut auf der Testumgebung auf, die im *vorherigen Abschnitt* beschrieben wurde. Die Webseite des Beispielszenarios verwendet zwei Sicherheitsgruppen: *Thesis Students* und *Thesis Teachers*. Diese entsprechen den Rollen *Studenten* und *Mitarbeiter*, die im *Abschnitt 1.6 Beispielszenario Prüfungsverwaltung* als die Rollen des Beispielszenarios ausgemacht wurden.

Die beiden Beispielkonten *Adamcar* und *Alansh* aus dem Aufbau der Testumgebung wurden auch als Benutzer für das Beispielszenario verwendet. *Adamcar* ist dabei in der Rolle Student in der Sicherheitsgruppe *Thesis Students* und *Alansh* als Mitarbeiter in der Gruppe *Thesis Teachers*. Es wurde auch ein drittes Benutzerkonto *Thomasv* erstellt, das in keiner dieser Gruppen Mitglied ist und somit weder in der Rolle Student noch Mitarbeiter ist. *Abbildung 68* zeigt die neu angelegten Sicherheitsgruppen im Active Directory.

Abbildung 68 Sicherheitsgruppen des Beispielszenarios im Active Directory

Tabelle 3 zeigt eine Übersicht der Benutzerkonten und der zugeordneten Sicherheitsgruppen. Die Kennwörter dienen hier als Referenz zur Überprüfung des Beispielsszenarios, das auf den beiliegenden DVDs enthalten ist. Siehe hierzu *Abschnitt 10.1 Beiliegende Datenträger*.

Benutzername	**Kennwort**	**Sicherheitsgruppen**	**Rolle**
Adamcar	P@ssw0rd	Domain Users; Thesis Students	Student
Alansh	P@ssw0rd	Domain Users; Thesis Teachers	Mitarbeiter
Thomasv	P@ssw0rd	Domain Users	Allgemeiner Benutzer

Tabelle 3 Übersicht der Benutzerkonten des Beispielszenarios

Der nächste Schritt besteht darin, die Claims in ADFS zu definieren, die von der Testanwendung benötigt werden. Für die beiden Rollen wurden entsprechend die Claims *Thesis Students Claim* und *Thesis Teachers Claim* angelegt. Diese Claims müssen jeweils beim Account Partner und beim Resource Partner erstellt werden. Sie müssen nicht identisch sein. In diesem Beispiel haben sie der Einfachheit halber dieselben Bezeichnungen. *Abbildung 69* zeigt die neu definierten Claims auf Seiten des Account Partner. Entsprechend ist die Darstellung beim Resource Partner.

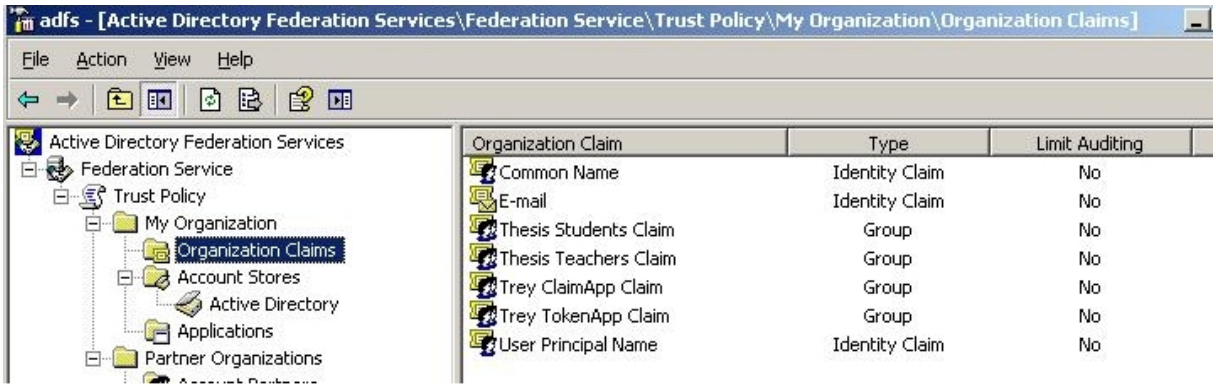

Abbildung 69 Claims des Beispielszenarios in ADFS

Die Claims müssen mit Sicherheitsgruppen aus dem Active Directory assoziiert werden. Die Sicherheitsgruppe *Thesis Students* bildet auf den Claim *Thesis Students Claim* und *Thesis Teachers* auf *Thesis Teachers Claim* ab. *Abbildung 70* zeigt diese Assoziation.

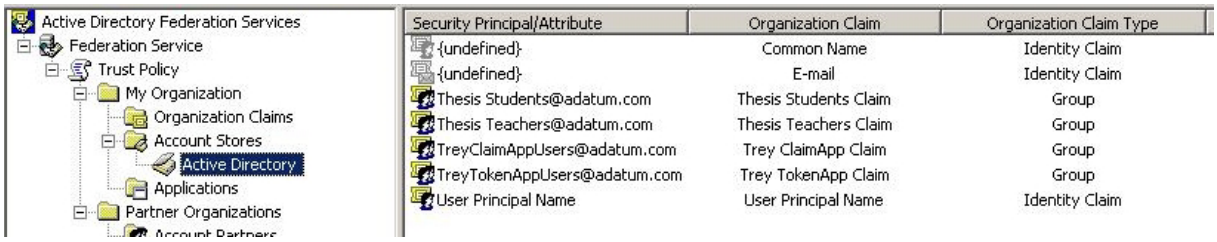

Abbildung 70 Mapping zwischen AD und ADFS

Der letzte Schritt auf Seiten des Account Partner ist die Transformation von internen Claims zu den herausgehenden Claims. Die herausgehenden Claims sind die einzige Vereinbarung die zwischen Account Partner und Resource Partner getroffen werden muss. In diesem Beispielszenario wurde die Vereinbarung auf *ThesisStudentsMapping* und *ThesisTeachersMapping* festgelegt, die respektive für die Claims *Thesis Students Claim* und *Thesis Teachers Claim* stehen. In *Abbildung 71* ist diese Vereinbarung zu sehen.

Umgekehrt wird auf Seiten des Resource Partner die Transformation für hereinkommende Claims definiert. Dieses Verfahren ist dem des Account Partner sehr ähnlich.

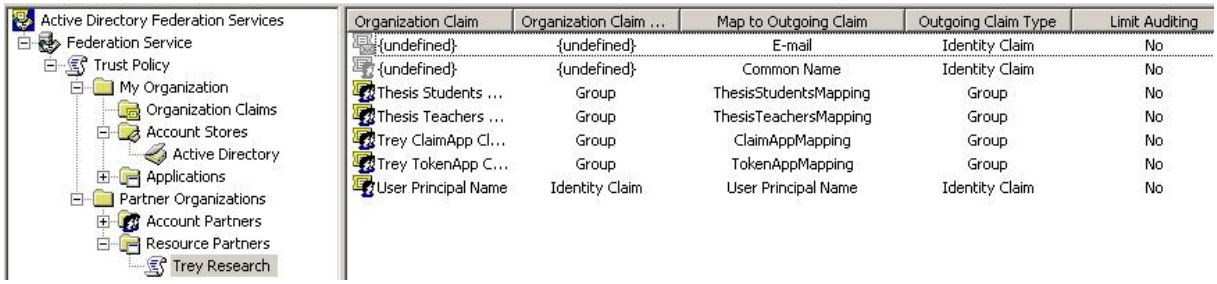

Abbildung 71 Herausgehende Claims des Beispielszenarios in ADFS

Der Resource Partner muss außerdem noch die Testanwendung in ADFS konfigurieren. Dazu wird eine neue Anwendung definiert, die über die Adresse der Anwendung identifiziert wird, wie im *Abschnitt 4.4.1 Konfiguration für ADFS* beschrieben wurde. Letztlich wird für die Anwendung festgelegt, welche der verfügbaren Claims der Anwendung übermittelt werden. In *Abbildung 72* ist zu sehen, dass die Claims *Thesis Students Claim*, *Thesis Teachers Claim* und der *User Principal Name* übermittelt werden.

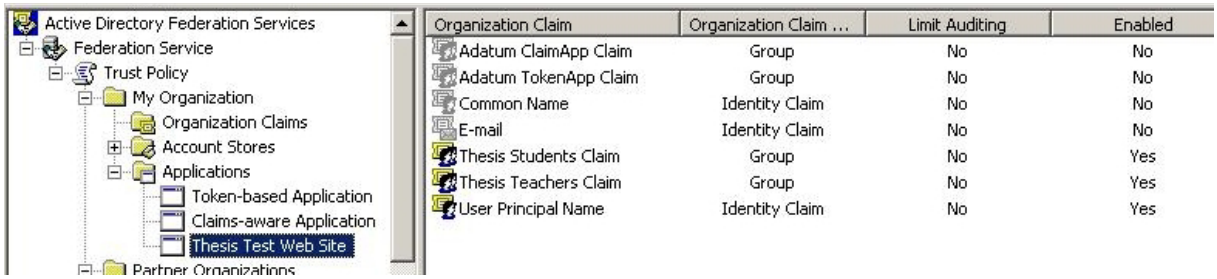

Abbildung 72 Definition einer Anwendung in ADFS

5.3 Aufbau der Testanwendung

Die Testanwendung ist eine Web-Anwendung, die einige der Workflows des Beispielszenarios, das im *Abschnitt 1.6* vorgestellt wurde, aufweist. Es gibt drei Webseiten: die Startseite, die Verwaltungsseite für die Studenten und die Verwaltungsseite für die Mitarbeiter.

Die Startseite (*Abbildung 73*), ist lediglich ein Verteiler zu den beiden anderen Seiten. Zusätzlich enthält sie einen Link zu einer Testseite für die Authentifizierung des aktuellen Benutzers. Diese Seite wird im nächsten *Abschnitt 5.4* genauer vorgestellt.

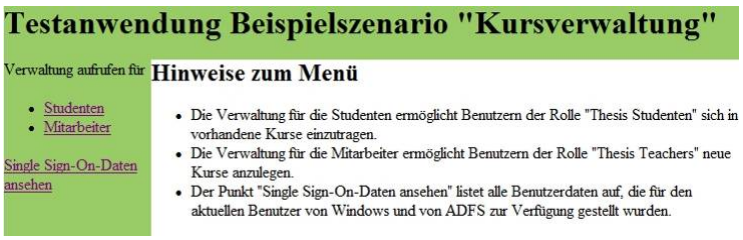

Abbildung 73 Startseite der Testanwendung

In der Verwaltung für Studenten (*Abbildung 74*) werden zwei Listen angezeigt. Die erste Liste führt alle verfügbaren Kurse auf und die zweite Liste die Kurse, in die sich der aktuelle Benutzer eingetragen hat. Weiterhin ist es dem Studenten möglich, sich auf dieser Seite in weitere Kurse einzutragen.

Abbildung 74 Verwaltungsseite für Studenten der Testanwendung

Die Verwaltung für die Mitarbeiter (*Abbildung 75*) enthält eine Liste, die alle verfügbaren Kurse enthält. Neue Kurse können über die Eingabemaske unterhalb der Liste hinzugefügt werden.

Abbildung 75 Verwaltungsseite für Mitarbeiter der Testanwendung

5.4 Test der Authentifizierung

Die Testanwendung enthält eine Webseite, die Auskunft über die Benutzerdaten gibt, die der Anwendung übermittelt wurden. Mithilfe dieser Seite kann genau festgestellt werden, welche Art der Authentifizierung stattgefunden hat und welche Informationen über den Benutzer zur Verfügung stehen. Diese Testseite wurde in fast unveränderter Form aus dem Step-by-Step-Guide [Pierson 2006] übernommen, der im *Abschnitt 5.1* zum Aufbau der Testumgebung diente.

In *Abbildung 76* ist der generelle Aufbau der Testseite zu sehen. Es gibt fünf Sektionen: *PageInformationen* gibt die Adresse an, die für den Aufruf der aktuellen Seite verwendet wurde. *User.Identity* gibt Information darüber, welche Art von Identität zur Verfügung steht. *(IIdentity)User.Identity* ist ein Interface unabhängig von der konkreten Instanz, die *User.Identity* zugrunde liegt. Ist *User.Identity* vom Typ *SingleSignOn* gibt die Sektion *(SingleSignOn)User.Identity* umfangreich Auskunft über etliche Parameter, die im Zusammenhang mit der Authentifizierung über ADFS stehen. Zusätzlich ist in der Sektion *SingleSignOnIdentity.SecurityPropertyCollection* die Auflistung aller Sicherheitseigenschaften zu finden, die der aktuelle Benutzer mit sich führt.

Abbildung 76 zeigt den Zugriff auf die Testanwendung als anonymer Nutzer. Weder ADFS noch NTLM wurden verwendet und es hat daher keine Authentifizierung des Benutzers stattgefunden.

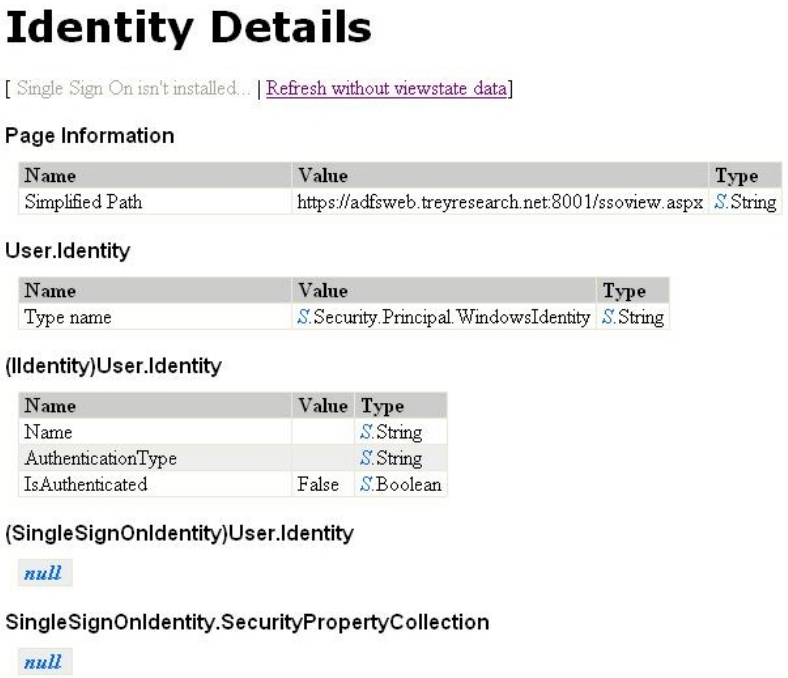

Abbildung 76 Benutzeridentität ohne SSO anonym

Abbildung 77 zeigt die Informationen, die für einen Benutzer zur Verfügung stehen, der über ADFS authentifiziert wurde. Die meisten Parameter sind selbsterklärend und für die Implementation des verteilten Autorisierungssystem nicht von Bedeutung. Besonders wichtig ist allerdings die Auflistung *SecurityPropertyCollection*, die die Gruppen enthält, in der der Benutzer Mitglied ist. Diese Angaben werden direkt, wie in *Abschnitt 4.4.2 Programmatischer Zugriff auf ADFS* beschrieben, für die Erstellung eines *ClaimPrincipal* verwendet.

User.Identity

Name	Value	Type
Type name	*SSO*.SingleSignOnIdentity	*S*.String

(IIdentity)User.Identity

Name	Value	Type
Name	Adamcar@adatum.com	*S*.String
AuthenticationType	WebSSO	*S*.String
IsAuthenticated	True	*S*.Boolean

(SingleSignOnIdentity)User.Identity

Name	Value
Name	Adamcar@adatum.com
NameType	http://schemas.xmlsoap.org/claims/UPN
SecurityPropertyCollection	*SSO.Auth*.SecurityPropertyCollection
AuthenticatingAuthority	urn:federation:adatum
AuthenticationMethod	urn:federation:authentication:windows
AuthenticationType	WebSSO
IsAuthenticated	True
SignInUrl	https://adfsresource.treyresearch.net/adfs/ls/?wa=wsignin1.0&v 2fadfsweb.treyresearch.net%3a8001%2f&wct=2007-02-15T 2fadfsweb.treyresearch.net%3a8001%2fssoview.aspx
SignOutUrl	https://adfsresource.treyresearch.net/adfs/ls/?wa=wsignout1.0
WindowsIdentity	*S*.Security.Principal.WindowsIdentity

SingleSignOnIdentity.SecurityPropertyCollection

Uri	Claim Type	Claim Value
http://schemas.xmlsoap.org/claims/UPN	UPN	Adamcar@adatum.com
http://schemas.xmlsoap.org/claims/Group	Group	Thesis Students Claim

Abbildung 77 Benutzeridentität mit SSO authentifiziert

Die Authentifizierung über ADFS wird in *Abbildung 78* der Authentifizierung über NTLM gegenübergestellt. Bei NTLM gibt es keine Auflistung der Gruppen, in denen der Benutzer Mitglied ist. Die Verwendung des verteilten Autorisierungssystems wäre somit etwas aufwändiger, da der Client durch alle lokalen und globalen Sicherheitsgruppen iterieren und auf Mitgliedschaft prüfen müsste, um die notwendigen Gruppen für die Erstellung eines *ClaimPrincipal* zu sammeln.

Page Information

Name	Value	Type
Simplified Path	http://localhost:4259/TestWebSite/ssoview.aspx	*S*.String

User.Identity

Name	Value	Type
Type name	*S*.Security.Principal.WindowsIdentity	*S*.String

(IIdentity)User.Identity

Name	Value	Type
Name	KIRCHNER\peter	*S*.String
AuthenticationType	NTLM	*S*.String
IsAuthenticated	True	*S*.Boolean

(SingleSignOnIdentity)User.Identity

null

SingleSignOnIdentity.SecurityPropertyCollection

null

Abbildung 78 Benutzeridentität mit NTLM authentifiziert

5.5 Testfälle

Das verteilte Autorisierungssystem wurde anhand von drei Workflows für drei Rollen getestet. Dazu wird eine Matrix aufgestellt, die die Erwartungswerte der Autorisierung enthält. Die drei Workflows sind dem Funktionsumfang des Beispielsszenarios entnommen, wie es in *Abschnitt 1.6* beschrieben wurde. Die Matrix ist in *Tabelle 4* dargestellt.

	Studenten	**Mitarbeiter**	**Allgemeine Benutzer**
Kurse anzeigen	Zugriff erlaubt	Zugriff erlaubt	Zugriff verweigert
An Kursen teilnehmen	Zugriff erlaubt	Zugriff verweigert	Zugriff verweigert
Kurse anlegen	Zugriff verweigert	Zugriff erlaubt	Zugriff verweigert

Tabelle 4 Erwartungswerte der Testfälle

Die Tests werden als Black-Box-Tests durchgeführt. Das heißt, mit jeder Rolle wird jede Operation durchgeführt und mit dem Erwartungswert verglichen. Kenntnis der internen Vorgänge ist dabei nicht nötig.

Die verschiedenen Rollen wurden simuliert, indem für jede Rolle die Webseite mit einem separaten Benutzerkonto besucht wurde. Die drei Benutzerkonten lauten *Adamcar*, *Alansh* und *Thomasv* (allgemeiner Benutzer). *Adamcar* ist Mitglied der Sicherheitsgruppe *Thesis Students*, *Alansh* Mitglied von *Thesis Teachers* und *Thomasv* ist Mitglied keiner Gruppe.

Tabelle 5 zeigt das Ergebnis der Testläufe. Alle Erwartungswerte wurden erfüllt und es ist somit von einer korrekten Funktion des verteilten Autorisierungssystems auszugehen. Die ausführliche Bewertung der Testläufe ist im Anhang im *Abschnitt 10.3* zu finden. Dort sind alle Screenshots mit einer Auswertung verzeichnet.

Rolle	**Operation**	**Zugriff gewährt?**	**Entspricht Erwartungswert?**
Student	Kurse anzeigen	☑	☑
	Kursen beitreten	☑	☑
	Kurse erstellen	☒	☑
Mitarbeiter	Kurse anzeigen	☑	☑
	Kursen beitreten	☒	☑
	Kurse erstellen	☑	☑
Allgemeiner Mitarbeiter	Kurse anzeigen	☒	☑
	Kursen beitreten	☒	☑
	Kurse erstellen	☒	☑

Tabelle 5 Ergebnis der Testläufe

6 Zusammenfassung

In dieser Arbeit wurden die Grundlagen der Authentifizierung und Autorisierung besprochen. Die Probleme der lokalen Autorisierung wurden herausgearbeitet und die Diskussion über bisherige Arbeiten zu dem Thema gaben Einblick, welche Lösungsansätze bereits erarbeitet wurden.

Der Schwerpunkt der Authentifizierung lag auf dem bereits existierenden Single Sign-On-System Active Directory Federation Services. Die theoretischen Grundlagen für dieses System wurden erarbeitet und die praktische Verwendung dokumentiert.

Es wurde ein Autorisierungssystem entworfen, das in der Lage ist, sich die Funktionen des SSO-Systems ADFS zunutze zu machen. Das verteilte Autorisierungssystem erfüllt die Zielsetzung, die die Grundlage dieser Arbeit waren. Die herausgearbeiteten Probleme wurden durch die entworfenen Lösungsansätze umgesetzt.

Das verteilte Autorisierungssystem ist, wie im Abstract vorgesehen, eine prototypische Implementation. Das bedeutet, es ist derzeit möglich eine verteilte Autorisierung durchzuführen. Der Entwurf sieht bereits die Konfiguration von Anwendungen und die Verwaltung des verteilten Autorisierungssystems vor. Diese Einstellungen müssen derzeit allerdings noch manuell in den XML-Konfigurationsdateien bearbeitet werden. Hilfsprogramme für die Konfiguration und Verwaltung sind wichtig, insbesondere da der Mensch bei derartigen Datenmengen zu Fehlern neigt. Im Rahmen dieser Arbeit stand jedoch die verteilte Autorisierung innerhalb von ADFS im Vordergrund, weswegen auf die Implementation der Hilfsprogramme verzichtet wurde. Die Verwaltung, wie sie im *Entwurf* geplant wurde, wurde nicht implementiert, da sie keinen wesentlichen Einfluss auf das Kernthema dieser Arbeit gehabt hätte. Die Modellierung der Verwaltung gibt allerdings einen guten Überblick über die Beziehungen zu den verschiedenen Elementen des Autorisierungssystems untereinander und wie diese vom Benutzer wahrgenommen werden.

Die Fehlerkontrolle wurde im Prototyp nur dort implementiert, wo sie für die Funktionskontrolle notwendig war. Fehlerfälle wie die Behandlung von Verbindungsabbrüchen oder Netzausfällen allgemein spielen beim Prototyp keine wichtige Rolle. Die Datenstrukturen an sich sind nicht anfällig für Fehler, da diese während Laufzeit durch das streng typisierte Framework nur die Daten aufnehmen können, für die sie vorgesehen wurden. Auch die Übermittlung durch die Web Services als XML und SOAP lässt kaum Möglichkeiten zu, Daten zu übertragen, die zu Programmfehlern führen können.

Die Protokollierung ist ein wichtiges Thema bei dem Zugriff auf Ressourcen. Die Protokollierung hat innerhalb dieser Arbeit jedoch eine andere Bedeutung erhalten, als sie bei der lokalen Autorisierung vorherrscht. Bei der lokalen Autorisierung geht mit der Protokollierung die Erwartung einher, dass ein Zugriff auf die betreffende Ressource stattgefunden hat. Bei der verteilten Autorisierung kann davon nicht ausgegangen werden. Wie das Beispielszenario veranschaulicht, kann das verteilte Autorisierungssystem auch für die Vorüberprüfung der Autorisierung verwendet werden. In diesem Fall hat noch kein Zugriff auf die Ressource stattgefunden.

Der *Entwurf* sieht ein neues Autorisierungskonzept der *Hints* vor. Ist eine Autorisierung nicht möglich, ist angedacht Vorschläge zurückzuliefern, die den Zugriff auf die entsprechende Ressource erlauben. Das Konzept der *Hints* wurde nicht implementiert, da dieses Konzept sehr aufwändig zu implementieren gewesen wäre und nicht Kernthema der Arbeit ist. Es hätte ein Regelsystem implementiert werden müssen, das in der Lage ist, Autorisierungsanfragen auf Basis der verschiedenen Anwendungskonfigurationen umzuändern, so dass eine positive Autorisierungsantwort gegeben werden kann. Die Datenstrukturen sind allerdings bereits auf die Implementierung der *Hints* vorbereitet. Ähnlich den *Hints* ist auch die Autorisierung auf

Basis von Parametern, eigenen Assemblys und LDAP-Anfragen vorbereitet, aber nicht implementiert worden.

Die Persistierung aller Datenstrukturen in dieser Arbeit wird in XML vorgenommen. Dieser Weg war, aus Erfahrung gelernt, aufwändiger als die binäre Serialisierung, wie in der Beschreibung der Konverterklassen im *Abschnitt 4.2 Persistierung* gut zu erkennen ist. Die Speicherung der Daten in XML ermöglicht allerdings die Verwendung von Hilfsprogrammen wie zum Beispiel Altova XMLSpy oder der programmatischen Weiterverarbeitung, da viele Frameworks die Verarbeitung von XML-Dateien ermöglichen. Auch können XML-Daten vom Menschen gelesen und geschrieben werden, was bei binären Daten ungleich aufwändiger ist. Dieser Umstand hat es ermöglicht, in dieser Arbeit auf die Implementierung der Verwaltung zu verzichten, da es möglich war, die notwendigen Anwendungsdefinitionen und -konfigurationen für die Testläufe manuell zu schreiben.

Die Testanwendung des Beispielszenarios zeigt die Verwendung des verteilten Autorisierungssystems, das in dieser Arbeit erstellt wurde. Die Testumgebung wird auf den beiliegenden DVDs mitgeliefert und kann daher auch getestet werden. Weitere Informationen zu den Ressourcen auf den DVDs sind im Anhang *Abschnitt 10.1* zu finden. Es wurden nicht alle Workflows umgesetzt, die im Beispielszenario aufgestellt wurden, sondern es wurde eine Auswahl getroffen, die unterschiedliche Rollen betreffen. Die Funktionsfähigkeit der verteilten Autorisierung kann daher anhand der Testanwendung und der ausgewählten Workflows überprüft werden.

7 Ausblick

Das Thema dieser Arbeit bietet viel Potential für Folgearbeiten. Die Problemanalyse und der Entwurf enthalten bereits viele Ideen für die verteilte Autorisierung, aber auch der Autorisierung allgemein. Die Zusammenfassung zeigt, dass aufgrund der beschränkten Zeit und Raum in dieser Arbeit nicht alle Ideen umgesetzt werden konnten. Dieser Ausblick soll aufzeigen, welches Potential in diesem Thema und dem verteilten Autorisierungssystem steckt.

Die Konzepte der *Hints*, Autorisierung auf Basis von Parametern und eigenen Assemblys sind Erweiterungen der allgemeinen Autorisierung, sei es nun lokal oder verteilt.

Die Autorisierung kann zusätzlich optimiert werden, indem es ermöglicht wird, eine ganze Sammlung (engl. Batches) an Autorisierungsanfragen zu übermitteln und dadurch den Kommunikationsaufwand zu minimieren. Es ist allerdings vorhersehbar, dass der Zeitaufwand mit einer steigenden Anzahl an Operationen und Benutzergruppen pro Anwendung einen größeren Anteil an der Ausführungszeit benötigt als die Übermittlung der Autorisierungsanfragen selbst – ein stabiles Netzwerk vorausgesetzt.

Im *Abschnitt 4.5 Autorisierung* wurde bereits angedeutet, dass die Verwendung der GUIDs für die Operationen im Quellcode eine Quelle für Fehler sein kann, da GUIDs vom Menschen nicht leicht gemerkt werden können. Hier bietet es sich an, ein Hilfsprogramm zu entwickeln, das aufgrund der Anwendungsdefinition automatisch Quellcode generiert, der ähnlich dem aus *Abschnitt 4.5* ist. Methoden mit aussagekräftigen Namen erleichtern das Lesen des Quellcodes und reduzieren damit die Gefahr, falsche Operationsbezeichner zu verwenden.

Zwei Konzepte, die eng miteinander in Beziehung stehen, sind der Einsatz von Caches auf Seiten des Resource Partner und der Kaskade von mehreren Autorisierungsservern. Das Prinzip ist dabei, die Antworten aller statischen Autorisierungsanfragen auf der Seite des Resource Partner zu speichern und bei der nächsten, gleichen Anfrage die Antwort aus dem Cache zu liefern. Dynamische Autorisierungen, die also auf LDAP-Anfragen, auf Parametern oder eigenen Autorisierungs-Assemblies beruhen, sind davon ausgenommen. Kann die Anfrage nicht beantwortet werden, wird der Autorisierungsserver kontaktiert.

Hierbei ist es nun denkbar, eine Kaskade von Autorisierungsservern einzurichten, die alle von individuellen Administratoren gepflegt werden und somit Spezialwissen halten können. Die Verwaltung von unterschiedlichen Benutzergruppen könnte beispielsweise auf mehrere Autorisierungsserver und mehrere Administratoren und Richtlinien verteilt werden.

In der *Zusammenfassung* wurde schon das Thema der Protokollierung angesprochen. Ergänzend zur der Protokollierung in dem Sinne, wie in der Zusammenfassung diskutiert, ist eine Protokollierung für den eigentlichen Zugriff auf die Ressourcen notwendig. Diese Art der Protokollierung, Auditing genannt, kann entweder als ein separates verteiltes System implementiert werden oder in das verteilte Autorisierungssystem integriert werden. Beide Ansätze haben gewiss ihre Vor- und Nachteile, die hier allerdings nicht weiter erörtert werden sollen.

Eine weitere sinnvolle Ergänzung des Autorisierungssystems wäre die Konfigurierbarkeit der Art und Weise, wie Privilegien ausgewertet werden. Ein denkbares Verfahren ist hier die Priorisierung von Privilegien anstatt der statischen Reihenfolge, erst die Negativ- und schließlich die Positivlisten zu überprüfen.

Die Kommunikation an sich bietet auch Potential für Veränderungen. Die Web Services nutzen derzeit die im .NET Framework mitgelieferte XML-Serialisierung von einfachen Klassen (siehe *Abschnitt 4.3 Kommunikation*). Hier würde sich anbieten stattdessen auf spezialisierte XML-Standards wie WS-Trust, WS-SecureConversation und WS-SecurityPolicy zu setzen. Auch für die Kodierung der Daten gibt es bereits Standards wie XACML, XrML und SAML, wobei der letztgenannte der am weitesten verbreitete ist.

8 Glossar

Dieses Glossar listet Abkürzungen und Begriffe auf, die in dieser Arbeit wiederholt verwendet wurden.

- .NET, gesprochen dot NET, ist eine Plattform, Architektur und Sammlung von Klassenbibliotheken. Die Plattform wird durch die Laufzeitumgebung (Runtime) gestellt, die .NET-Programme ausführen kann. .NET unterstützt weit über 40 Programmiersprachen. Die wichtigsten sind C#, VB.NET, J# und C++.NET.
- ACL, Access Control List ist eine Liste, die Paare von Subjekten und einer Menge von Zugriffsrechten enthält. Siehe *Abschnitt 1.5.1 Zugriffskontroll-Matrix, Seite 20*.
- AD, Active Directory ist ein Verzeichnisdienst in Windows Netzwerken. Siehe *Abschnitt 1.4.1.3Active Directory, Seite 10*.
- AD/AM, Active Directory in Application Mode ist eine bestimmte Ausführungsart des Active Directory, die keine Domäne voraussetzt. Siehe *Abschnitt 1.4.1.4 Active Directory in Application Mode, Seite 13*.
- ADFS, Active Directory Federation Services sind Web-Dienste, die den Zugriff auf das Active Directory über Domänengrenzen hinweg ermöglicht. Siehe *Abschnitt 1.4.2 Active Directory Federation Services, Seite 14*.
- AP, Account Partner ist ein Begriff im Kontext von ADFS und bezeichnet die Partei, die im Besitz der Benutzerkonten ist. Siehe *Abschnitt 1.4.2 Active Directory Federation Services, Seite 14*.
- API, Application Programming Interface bezeichnet eine Programmierschnittstelle für Software. APIs ermöglichen häufig die Ansteuerung oder Erweiterung von Software für Drittanbieter.
- Auditing bezeichnet die Protokollierung der Zugriffe von Benutzern und Prozessen auf Ressourcen. Auditing kann dabei Zugriffsarten wie zum Beispiel Öffnen, Schreiben und Lesen unterscheiden.
- C#, gesprochen C Sharp, ist eine objektorientierte Programmiersprache in .NET. Sie hat Ähnlichkeit mit C++ und Java. Beispiele von C#-Quellcode sind im *Kapitel 4 Realisierung, Seite 72,* zu finden.
- DAC, Discretionary Access Control bezeichnet ein Verfahren, bei dem die Eigentümer von Ressourcen die Zugriffsberechtigungen selbst bestimmen können. Siehe *Abschnitt 1.5.2 Discretionary & Mandatory Access Control, Seite 21*.
- DN, Distinguished Name bezeichnet in Verzeichnisdiensten einen absoluten, eindeutigen Namen, um Einträge zu identifizieren und zu finden. Siehe *Abschnitt 1.4.1.1 X.500, Seite 8*.
- DNS, Domain Name System bezeichnet einen Dienst und ein Protokoll, die vornehmlich zur Auflösung von Domainnamen zu IP-Adressen verwendet wird. DNS wird abgesehen von der Auflösung von Domainnamen auch für andere Nachschlagefunktionen verwendet.
- Framework bezeichnet in der Regel eine Klassenbibliothek, die der Softwareentwicklung dient.
- Fx ist Kurzform für Framework.

- GUID, Globally Unique Identifier bezeichnet eine 128 Bit lange, global eindeutige Zahl. Ein Beispiel einer GUID ist {E2B4742D-D744-4b2d-A9C2-CB55EC42DD46}.
- IIS, Internet Information Services ist der Name des integrierten Webservers in Microsoft Windows.
- J#, gesprochen J Sharp, ist eine objektorientierte Programmiersprache in .NET. Sie ist das .NET-Pendant zu Java.
- Kerberos ist ein verteiltes Authentifizierungsprotokoll, das Netzwerkdienste und die Kommunikation zwischen diesen definiert. Siehe *Abschnitt 1.4.3 Kerberos, Seite 17*.
- LDAP, Lightweight Directory Access Protocol bezeichnet ein Protokoll, das den einfachen Zugriff auf Verzeichnisdienste ermöglicht. Siehe *Abschnitt 1.4.1.2 Lightweight Directory Access Protocol, Seite 10*.
- MAC, Mandatory Access Control bezeichnet ein Verfahren, bei dem das System regelbasiert die Zugriffsberechtigungen auf Ressourcen festlegt. Siehe *Abschnitt 1.5.2 Discretionary & Mandatory Access Control, Seite 21*.
- NTLM, NT LAN Manager bezeichnet die Standardauthentifizierung in Microsoft Netzwerken.
- Objekt ist ein Begriff aus der Informationssicherheit und bezeichnet alle Arten von Ressourcen in einem Computersystem. Es wird dabei zwischen passiven Objekten (z.B. Dateien und Datenbankeinträge) und aktiven Objekten (z.B. Prozesse, Dienste und Drucker) unterschieden. Siehe auch *Abschnitt 1.5 Autorisierung, Seite 20*.
- RBAC, Role-Based Access Control bezeichnet ein Sicherheitsmodell, das Zugriffsberechtigungen auf Basis von Rollen definiert. Siehe *Abschnitt 1.5.3 Role-Based Access Control (RBAC), Seite 22*.
- RDN, Relative Distinguished Name bezeichnet in Verzeichnisdiensten einen relativen, eindeutigen Namen, um Einträge zu identifizieren und zu finden. Siehe *Abschnitt 1.4.1.1 X.500, Seite 8*.
- RP, Resource Partner ist ein Begriff im Kontext von ADFS und bezeichnet die Partei, die im Besitz der Ressourcen ist, auf die Benutzer des Account Partner zugreifen möchten. Siehe *Abschnitt 1.4.2 Active Directory Federation Services, Seite 14*.
- Sandbox bezeichnet eine von einer Laufzeitumgebung erstellte, von dem Betriebsystem isolierte Umgebung, in der Programmcode ausgeführt werden kann, dessen Zugriffe auf die Ressourcen des Betriebssystems von der Laufzeitumgebung beschränkt oder verweigert werden. Beispiele für Sandboxen sind die virtuelle Maschine von Java und die Laufzeitumgebung von .NET. Siehe auch *2.3.2 Security Agent Based Distributed Authorization: An Approach, Seite 38*.
- SDK, Software Development Kit bezeichnet eine Sammlung von Bibliotheken, Hilfsprogrammen, Dokumentationen und Beispielen für die Entwicklung von Software. SDKs sind häufig auf bestimmte Arten von Software spezialisiert.
- SOA, Services Oriented Architecture bezeichnet eine Systemarchitektur, die auf modulare Dienste zur Erbringung von Geschäftsprozessen setzt. Insbesondere werden mit SOA Web Services verbunden, die diese Dienste anbieten, mit dem Ziel eine möglichst hohe Kompatibilität mit anderen Diensten zu erreichen.
- SOAP, ursprünglich die Abkürzung für Simple Object Access Protocol, stellt ein Protokoll für den Datenaustausch zwischen Systemen per XML dar. Mittlerweile ist SO-

AP jedoch keine Abkürzung mehr, sondern der Name ist SOAP, da das Protokoll in mehreren Versionen immer wieder erweitert wurde.

- SQL, Structured Query Language bezeichnet eine Abfragesprache für Datenbanken.
- SSO, Single Sign-On bezeichnet ein Verfahren der Authentifizierung in Netzwerken, die es dem Benutzer ermöglicht, auf mehrere Dienste mit einer einmaligen Anmeldungen zuzugreifen. Siehe *Abschnitt 1.4.4 Single Sign-On, Seite 19*.
- STS, Security Token Service bezeichnet einen Dienst, der für die Erstellung von Sicherheitstoken verantwortlich ist. Sicherheitstoken sind Datenstrukturen, die die Identität oder gewisse Eigenschaften von Benutzern und Prozessen bescheinigen. Siehe *Abschnitt 1.4.2.2 Server Rollen & Komponenten, Seite 15, in 1.4.2 Active Directory Federation Services*.
- Subjekt ist ein Begriff aus der Informationssicherheit und bezeichnet Benutzer und vom Benutzer beauftragte Objekte, die ausführungsfähig sind, wie zum Beispiel Prozesse und Dienste. Siehe auch *Abschnitt 1.5 Autorisierung, Seite 20*.
- TGS, Ticket Granting Service ist ein Dienst in Kerberos, der Tickets zur Bescheinigung der Identität von Benutzern und Prozessen ausstellt. Ähnlich STS. Siehe *Abschnitt 1.4.3 Kerberos, Seite 17*.
- UDDI, Universal Description, Discovery and Integration bezeichnet einen Verzeichnisdienst für Web Services.
- UML, Unified Modeling Language bezeichnet eine vielseitige Beschreibungssprache zur Modellierung von Software, Prozessen und anderen Beziehungen. Siehe *Abschnitt 1.2.2 Diagramme, Seite 3*.
- UPN, User Principal Name bezeichnet den Benutzernamen eines Benutzers innerhalb einer Domäne in der typischen Form einer E-Mail-Adresse. Siehe *Abschnitt 1.4.2.3 Authentifizierung, Seite 16, in 1.4.2 Active Directory Federation Services*.
- VB.NET, gesprochen VB dot NET oder Visual Basic dot NET, ist eine objektorientierte Programmiersprache in .NET. Sie ist eine Weiterentwicklung von Visual Basic.
- WS, Web Service oder Webdienst ist eine Webanwendung, die Dienste für Anwendungen per XML bereitstellt. Dienstaufrufe werden dabei meist per SOAP übertragen.
- XML, Extensible Markup Language bezeichnet eine Auszeichnungssprache von strukturierten, hierarchischen Daten. Siehe auch *Abschnitt 3.5 Schemata der XML-Daten, Seite 67*.

9 Verzeichnisse

9.1 Literaturverzeichnis

Coulouris 2005 COULOURIS, George ; DOLLIMORE, Jean ; KINDBERG, Tim: *Verteilte Systeme : Konzepte und Design.* 3., überarbeitete Auflage München : Pearson Studium, 2005 (Informatik). – ISBN 3-8273-7022-1

Daigle 2004 DAIGLE, L.: *WHOIS Protocol Specification* URL http://tools.ietf.org/html/rfc3912. – VeriSign, Inc.

Eckert 2002 ECKERT, Claudia: *IT-Sicherheit : Konzepte - Verfahren - Protokolle.* 2. Auflage Oldenbourg, R, 2002 – ISBN 3486272055

Emig 2006 EMIG, Christian ; SCHANDUA, Heiko ; ABECK, Sebastian: *SOA-aware Authorization Control.* International Conference Software Engineering Advances ICSEA'06, Tahiti / French Polynesia, November 2006

Ferraiolo 2003 FERRAIOLO, David F. ; KUHN, D. Richard ; CHANDRAMOULI, Ramaswamy: *Role-based access control.* Boston Mass. : Artech House, 2003 (Artech House computer security series). – ISBN 1-58053-370-1

Fuhrberg 2001 FUHRBERG, Kai ; HÄGER, Dirk ; WOLF, Stefan: *Internet-Sicherheit : Browser, Firewalls und Verschlüsselung.* 3., aktualisierte und erweiterte Auflage München : Hanser, 2001 – ISBN 3-446-21725-8

ICANN URL http://www.icann.org/. – Internet Corporation For Assigned Names and Numbers (ICANN)

Karp 2006 KARP, Alan H.: *Authorization-Based Access Control for the Services Oriented Architecture.* 2006

Klement 2003 KLEMENT, Peter ; MICHELA, Franco ; PALME, Markus: *Active Directory : Ein konzeptioneller Überblick über den Verzeichnisdienst von Microsoft Windows Server 2003.* Unterschleißheim : Microsoft Press, 2003 (Fachbibliothek). – ISBN 3-86063-676-6

Kristol 2000 KRISTOL, D. ; MONTULLI, L.: *HTTP State Management Mechanism* URL ftp://ftp.rfc-editor.org/in-notes/rfc2965.txt. – Bell Laboratories, Lucent Technologies; Epinions.com, Inc.

Microsoft 2003 MICROSOFT CORPORATION: *Description of Cookies* URL http://support.microsoft.com/kb/260971/en-us. – Aktualisierungsdatum: 2003-11-25. – Überprüfungsdatum 2006-11-22

Microsoft Svr: *Windows Server.* 2003 R2. Microsoft Corporation URL http://www.microsoft.com/windowsserver2003/default.mspx. – Überprüfungsdatum 2006-11-15

Microsoft TechNet 2005 MICROSOFT TECHNET: *Active Directory Federation Services (ADFS)* URL http://technet2.microsoft.com/WindowsServer/en/library/050392bc-c8f5-48b3-b30e-bf310399ff5d1033.mspx

Neuman 2005 NEUMAN, C.: *The Kerberos Network Authentication Service (V5)* URL http://tools.ietf.org/html/rfc4120. – MIT

Pierson 2006 PIERSON, Nick: *Step-by-Step Guide for Active Directory Federation Services.* Version 1.2 Juni 2006

Priebe 2005 PRIEBE, Torsten ; DOBMEIER, Wolfgang ; MUSCHALL, Björn ; PERNUL, Günther: ABAC - Ein Referenzmodell für attributbasierte Zugriffskontrolle. In: FEDERRATH, Hannes (Hrsg.). *Sicherheit 2005: Sicherheit - Schutz und Zuverlässigkeit, Beiträge der 2. Jahrestagung des Fachbereichs Sicherheit der Gesellschaft für Informatik e.v. (GI), 5.-8. April 2005 in Regensburg*. GI, 2005. (LNI, 62). – ISBN 3-88579-391-1, S. 285–296

Sermersheim 2006 SERMERSHEIM, Ed. J.: *Lightweight Directory Access Protocol (LDAP): The Protocol* URL http://www.ietf.org/rfc/rfc4511.txt. – Novell, Inc.

Stephenson 2004 STEPHENSON, David: *XML Schema best practices.* December 2004

Tanenbaum 2003 TANENBAUM, Andrew S. ; VAN STEEN, Maarten ; MUHR, Judith: *Verteilte Systeme : Grundlagen und Paradigmen.* München : Pearson Studium, 2003 – ISBN 3-8273-7057-4

Varadharajan VARADHARAJAN, V. ; KUMAR, N. ; MU, Y.: Security Agent Based Distributed Authorization: An Approach. In: NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY; NATIONAL COMPUTER SECURITY CENTER OF THE NATIONAL SECURITY AGENCY (HRSG.). *Proc. 21st National Information Systems Security Conference,* S. 315–328

Woo 1993 WOO, Thomas Y. C. ; LAM, Simon S.: A framework for distributed authorization. In: ASSOCIATION FOR COMPUTING MACHINERY. (HRSG.). *1st ACM Conference on Computer and Communications Security : Nov. 3 - 5 1993, Fairfax, Virginia*. New York NY : ACM Press, 1993. – ISBN 0-89791-629-8, S. 112–118

9.2 Verzeichnis der Abbildungen

9.3 Verzeichnis der Tabellen

9.4 Verzeichnis der Quellcode-Auszüge

10 Anhang

10.1 Beiliegende Datenträger

Dieser Arbeit liegen zwei DVDs bei, die das Programm Virtual PC 2007, die Testumgebung in Form von vier virtuellen Maschinen, diese Arbeit in elektronischer Form und den Quellcode des verteilten Autorisierungssystems enthalten.

Das Programm Virtual PC 2007 und die virtuellen Maschinen dürfen aus lizenzrechtlichen Gründen der Öffentlichkeit nicht zugänglich gemacht werden. Die Beilage dient lediglich der Überprüfbarkeit des in dieser Arbeit erstellten Prototyps des verteilten Autorisierungssystems.

Für den administrativen Zugriff auf die virtuellen Maschinen ist als Benutzername *Administrator* und als Kennwort *pass* zu verwenden. Die Kennwörter der Benutzerkonten sind in der *Tabelle 3 im Abschnitt 5.2 Konfiguration des Beispielszenarios auf Seite 84* aufgeführt.

Die Verzeichnisstruktur der DVDs ist *Tabelle 6* zu entnehmen.

DVD	Verzeichnis	Inhalt
1	Diplomarbeit	Die vorliegende Diplomarbeit als PDF-Dokument.
	Quellcode	Alle Quellcodes des verteilten Autorisierungssystems und der Testanwendung.
	Virtual PC 2007	Das Programm Microsoft Virtual PC 2007 in der Version 6.0.156.0 mit dem die Testläufe durchgeführt wurden.
	Virtuelle Maschinen\adfsaccount	Die virtuelle Maschine für das System *adfsaccount*, das den Domänencontroller für *A. Datum* enthält und den Account Partner innerhalb der ADFS-Infrastruktur darstellt.
	Virtuelle Maschinen\adfsclient	Die virtuelle Maschine für das System *adfsclient*, das Mitglied der Domäne *A. Datum* ist.
	XML	Die XML-Schemata für die Anwendungsdefinition- und konfiguration und die XML-Beispieldaten der Testanwendung.
2	Virtuelle Maschinen\adfsresource	Die virtuelle Maschine für das System *adfsresource*, das den Domänencontroller für *Trey Research* enthält.
	Virtuelle Maschinen\adfsweb	Die virtuelle Maschine für das System *adfsweb*, das Mitglied der Domäne *Trey Research* ist, den Webserver für die Testanwendung betreibt und innerhalb der ADFS-Infrastruktur den Resource Partner darstellt.

Tabelle 6 Verzeichnisstruktur der beiliegenden DVDs

10.2 XML-Dateien des Beispielszenarios

10.2.1 Anwendungsdefinition

```
<?xml version="1.0" encoding="UTF-8"?>
<application id="{F089120A-DEA0-4a5e-B1E1-9B62FED5DC8C}" name="Thesis Test Web Site" version="0"
        xsi:noNamespaceSchemaLocation="Application%20Definition.xsd"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
        <context id="{463D726D-1AE3-42de-AE5F-EA87B1D99CE3}" name="root" version="0">
                <contexts>
                        <context id="{A24F5EC3-B0AB-48b2-B694-DACC0D31A1CD}" name="attendees"
                                version="0">
                                <operations>
                                        <operation id="{E6CC33F7-862C-4093-B85A-EFB0D7004435}"
                                                name="AttendCourse" version="0"/>
                                </operations>
                        </context>
                        <context id="{5127A3A4-66DF-4dcb-A7A8-EDE5466FC081}"
                                name="management" version="0">
                                <operations>
                                        <operation id="{DBF730E2-5A31-41b8-B52E-263201065B51}"
                                                name="CreateCourse" version="0"/>
                                </operations>
                        </context>
                </contexts>
                <operations>
                        <operation id="{18C9F0E3-AA77-4e19-ABF5-FEF1AFB9F705}"
                                name="ViewCourses" version="0"/>
                </operations>
                <tasks>
                        <task id="{C626FFEC-901D-429e-9EF4-B21455F7EA84}" name="AttendCourse"
                                version="0">
                                <operationRef>{E6CC33F7-862C-4093-B85A-EFB0D7004435}
                                </operationRef>
                        </task>
                        <task id="{F5FA49B0-2C74-4524-9FEA-EEA556F3E88E}" name="CreateCourse"
                                version="0">
                                <operationRef>{DBF730E2-5A31-41b8-B52E-263201065B51}
                                </operationRef>
                        </task>
                        <task id="{7A7D47EF-D15A-422c-B4EC-73DC3A9B5813}" name="ViewCourse"
                                version="0">
                                <operationRef>{18C9F0E3-AA77-4e19-ABF5-FEF1AFB9F705}
                                </operationRef>
                        </task>
                </tasks>
        </context>
</application>
```

10.2.2 Anwendungskonfiguration

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration id="{EEA5EF28-F6E2-41ba-9E87-5FC91BA175DE}" revision="0" name="Test Configuration"
        xsi:noNamespaceSchemaLocation="Application%20Configuration.xsd"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
        <contexts>
                <context contextId="{93657960-9B3D-492f-8E08-E9A79B336706}" revision="0">
                        <privileges>
                                <privilege id="{2B251E4D-560C-4731-8DE8-7ECCFFAA6CA6}"
                                        name="ViewCoursesByStudents">
                                        <taskRef>{7A7D47EF-D15A-422c-B4EC-73DC3A9B5813}
                                        </taskRef>
```

```
                                        <userGroupRef>{77104441-979B-4b6e-99F8-FB1A1002D43E}
                                        </userGroupRef>
                                </privilege>
                                <privilege id="{E90F7058-CF68-4261-A559-2FC5412893DD}"
                                        name="ViewCoursesByTeachers">
                                        <taskRef>{7A7D47EF-D15A-422c-B4EC-73DC3A9B5813}
                                        </taskRef>
                                        <userGroupRef>{0207A102-472B-44b6-8ED1-9A1AFF36D894}
                                        </userGroupRef>
                                </privilege>
                                <privilege id="{C7687816-9696-4ff2-9617-FE67B49C3125}"
                                        name="AttendCourses">
                                        <taskRef>{C626FFEC-901D-429e-9EF4-B21455F7EA84}
                                        </taskRef>
                                        <userGroupRef>{77104441-979B-4b6e-99F8-FB1A1002D43E}
                                        </userGroupRef>
                                </privilege>
                                <privilege id="{BE0E7375-7EC3-4224-8183-AAD4D52485EB}"
                                        name="CreateCourses">
                                        <taskRef>{F5FA49B0-2C74-4524-9FEA-EEA556F3E88E}
                                        </taskRef>
                                        <userGroupRef>{0207A102-472B-44b6-8ED1-9A1AFF36D894}
                                        </userGroupRef>
                                </privilege>
                        </privileges>
                </context>
        </contexts>
        <userGroups>
                <userGroup id="{77104441-979B-4b6e-99F8-FB1A1002D43E}" membership="true"
                        name="Students">
                        <roles>
                                <role>Thesis Students Claim</role>
                        </roles>
                </userGroup>
                <userGroup id="{0207A102-472B-44b6-8ED1-9A1AFF36D894}" membership="true"
                        name="Teachers">
                        <roles>
                                <role>Thesis Teachers Claim</role>
                        </roles>
                </userGroup>
        </userGroups>
        <applicationDefinition>
                <idref>{F089120A-DEA0-4a5e-B1E1-9B62FED5DC8C}</idref>
                <applicationRevision>0</applicationRevision>
                <definitionPath>C:\Documents and Settings\peter.KIRCHNER\
                        My Documents\Studium\DA\Implementierung\Schema\
                        AppDef Thesis TestWebSite.xml</definitionPath>
        </applicationDefinition>
</configuration>
```

10.3 Technische Daten der virtuellen Maschinen

Tabelle 7 listet die technischen Daten der virtuellen Maschinen, die für die Testläufe in *Kapitel 5* verwendet wurden.

Identifikation	**Rolle in der Testumgebung**	**Betriebssystem**	**Virtuelle Hardware**	**Netzwerkeinstellungen**
Hostname: *adfsclient* Domäne: *adatum.com*	Client	Windows XP Professional (SP2)	420 MB Arbeitsspeicher	IP-Adresse: 192.168.1.1 Subnet-Maske: 255.255.255.0 DNS (primär): 192.168.1.3 DNS (sekundär): 192.168.1.4
Name: *adfsweb* Domäne: *treyresearch.net*	Web-Server	Windows Server 2003 R2, Enterprise Edition	420 MB Arbeitsspeicher	IP-Adresse: 192.168.1.2 Subnet-Maske: 255.255.255.0 DNS (primär): 192.168.1.4
Name: *adfsaccount* Domäne: *adatum.com*	Federation Server und Domain Controller	Windows Server 2003 R2, Enterprise Edition	420 MB Arbeitsspeicher	IP-Adresse: 192.168.1.3 Subnet-Maske: 255.255.255.0 DNS (primär): 192.168.1.3
Name: *adfsresource* Domäne: *treyresearch.net*	Federation Server und Domain Controller	Windows Server 2003 R2, Enterprise Edition	420 MB Arbeitsspeicher	IP-Adresse: 192.168.1.4 Subnet-Maske: 255.255.255.0 DNS (primär): 192.168.1.4

Tabelle 7 Technische Daten der virtuellen Systeme

10.4 Screenshots der Testläufe

Dieser Abschnitt enthält die Screenshots der Testfälle, wie sie im *Abschnitt 5.5* beschrieben wurden. Für jeden Testfall werden die entsprechenden Screenshots bewertet.

Die Operation „Anzeigen von Kursen“ wurde gewährt, wenn die Listen auf den beiden Webseiten angezeigt wurden. Der Zugriff wurde verweigert, wenn nicht die Liste sondern stattdes-

sen die Fehlermeldung „Sie haben keine Berechtigung, die Kurse einzusehen.“ angezeigt wurde.

Die Operation „Kursen beitreten“ wurde gewährt, wenn auf der Seite „Student - Veranstaltungen verwalten“ die Schaltfläche „Kurs beitreten“ aktiviert ist. Ist diese Schaltfläche deaktiviert, wurde der Zugriff verweigert.

Die Operation „Kurse erstellen“ wurde gewährt, wenn auf der Seite „Mitarbeiter - Veranstaltungen erwalten“ die Schaltfläche „Veranstaltung erstellen“ aktiviert ist. Ist diese Schaltfläche deaktiviert, wurde der Zugriff verweigert.

Abbildung 79 und *Abbildung 80* zeigen den bestandenen Testfall für die Rolle der Studenten. Die Kurse werden angezeigt und der Student hat die Möglichkeit, Kursen beizutreten. Er darf allerdings keine Kurse erstellen.

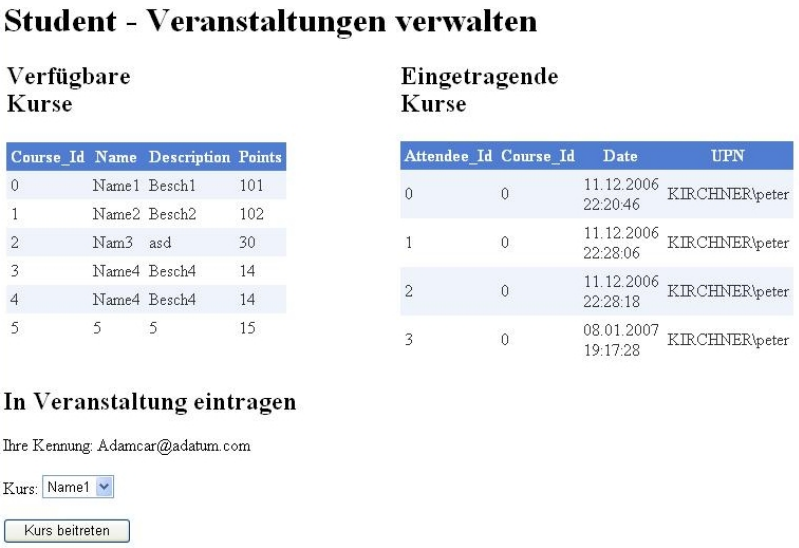

Abbildung 79 Testfall Anzeigen und Beitreten für Studenten

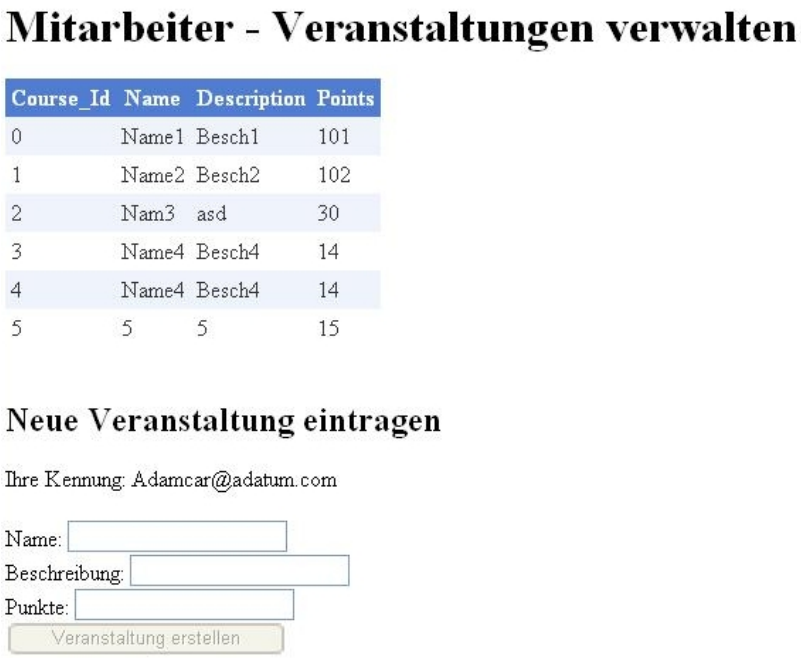

Abbildung 80 Testfall Anzeigen und Erstellen für Studenten

Abbildung 81 und *Abbildung 82* zeigen den bestandenen Testfall für die Rolle der Mitarbeiter. Die Kurse werden angezeigt und der Mitarbeiter hat keine Möglichkeit, Kursen beizutreten. Er darf allerdings Kurse erstellen.

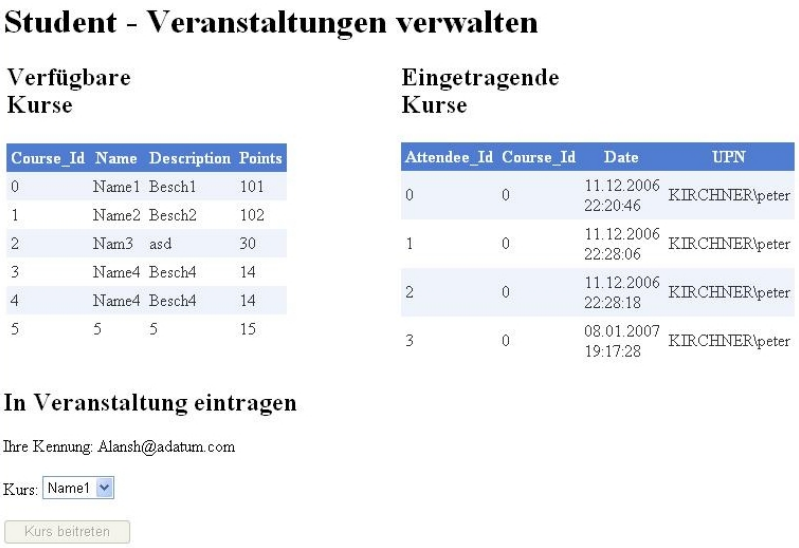

Abbildung 81 Testfall Anzeigen und Beitreten für Mitarbeiter

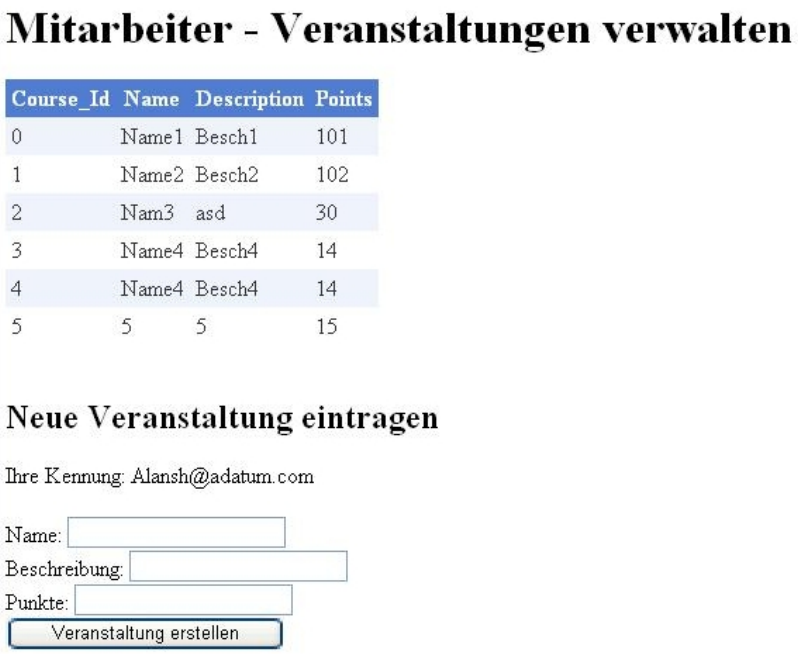

Abbildung 82 Testfall Anzeigen und Erstellen für Mitarbeiter

Abbildung 83 und *Abbildung 84* zeigen den bestandenen Testfall für die Rolle der allgemeinen Benutzer, also denjenigen, die weder der Rolle Student noch Mitarbeiter angehören. Die Kurse werden nicht angezeigt, stattdessen erscheint eine Fehlermeldung, die auf die fehlenden Rechte hinweist. Der Benutzer hat keine Möglichkeit Kursen beizutreten oder diese zu erstellen.

Student - Veranstaltungen verwalten

Verfügbare Kurse

Sie haben keine Berechtigung, die Kurse einzusehen.

Eingetragende Kurse

Attendee_Id	Course_Id	Date	UPN
0	0	11.12.2006 22:20:46	KIRCHNER\peter
1	0	11.12.2006 22:28:06	KIRCHNER\peter
2	0	11.12.2006 22:28:18	KIRCHNER\peter
3	0	08.01.2007 19:17:28	KIRCHNER\peter

In Veranstaltung eintragen

Ihre Kennung: thomasv@adatum.com

Kurs:

Kurs beitreten

Abbildung 83 Testfall Anzeigen und Beitreten für allgemeine Benutzer

Mitarbeiter - Veranstaltungen verwalten

Sie haben keine Berechtigung, die Kurse einzusehen.

Neue Veranstaltung eintragen

Ihre Kennung: thomasv@adatum.com

Name:
Beschreibung:
Punkte:
Veranstaltung erstellen

Abbildung 84 Testfall Anzeigen und Erstellen für allgemeine Benutzer