Zugriffskontrolle in dienstorientierten Architekturen

zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften

von der Fakultät für Informatik
der Universität Fridericiana zu Karlsruhe (TH)

genehmigte

Dissertation

von

Christian Emig

aus Karlsruhe

Tag der mündlichen Prüfung:	25. Juni 2008
Erster Gutachter:	Prof. Dr. Sebastian Abeck
Zweiter Gutachter:	Prof. Dr. Hannes Hartenstein

Inhaltsverzeichnis

1 Einleitung

1.1 Einführung in das Themengebiet

In der heutigen Zeit der Globalisierung und des ständig zunehmenden Wettbewerbs ist eine Rechnerunterstützung im Sinne des Einsatzes von Informationstechnologie (IT) zur Steigerung der Effizienz im Unternehmensumfeld nicht mehr wegzudenken. Im Rahmen der Rechnerunterstützung wird üblicherweise eine Vielzahl von Software-Systemen unterschiedlicher Hersteller eingesetzt mit dem Ziel, Geschäftsprozesse oder Teile davon zu unterstützen. Diese Software-Systeme bilden eine heterogene, zum Teil über Jahrzehnte gewachsene und individuell geartete Anwendungs- und Systemlandschaft.

In den Anfängen der Rechnerunterstützung wurden große, oft monolithische Software-Systeme als individuelle Lösungen entwickelt. Zur Entwicklung solcher Individual-Software wird nach der Erfassung der relevanten Geschäftsprozesse und der Festlegung der gewünschten, zugehörigen Rechnerunterstützung eine passende Software-Architektur definiert und darauf aufbauend ein entsprechendes Software-System implementiert [Ba00]. Die Entwicklung von Individual-Software ist langwierig und teuer, ermöglicht jedoch eine maßgeschneiderte Rechnerunterstützung [St06a]. Den vorgenannten Problemen wird durch die Entwicklung von Standard-Software entgegengewirkt, die, ausgehend von branchenüblichen Geschäftsprozessen, für einen großen potenziellen Kundenkreis entwickelt wird. Oftmals erweist sich die Anpassung von Standard-Software an die spezifischen Erfordernisse eines Unternehmens jedoch nicht minder aufwendig und damit einhergehend kostenintensiv [Li03].

Unter anderem sind es zwei große Problemfelder, die zu einem Umdenken in der Software-Entwicklung führten. Ein zentrales Problem aktueller Anwendungssysteme ist die geringe Flexibilität zur Anpassung, wenn ein Unternehmen seine Geschäftsprozesse neu gestaltet. Individualität stellt einen Erfolgsfaktor für ein Unternehmen dar. Um im Wettbewerb bestehen zu können, muss es effizientere und damit bessere Geschäftsprozesse einsetzen als seine Mitbewerber. Zudem muss es dem Unternehmen möglich sein, Prozessanpassungen schnell umsetzen zu können. Dabei soll es von seinen Anwendungssystemen unterstützt und nicht behindert werden. Hierzu ist es insbesondere erforderlich, dass sich Software-Lösungen nach den Unternehmensanforderungen richten und nicht umgekehrt.

Der Erfolg einer Rechnerunterstützung kann also daran gemessen werden, ob und wie schnell sie sich an das Unternehmen und seine Geschäftsprozesse anpassen lässt. Diese Herausforderung kann mit herkömmlichen Software-Systemen nicht schnell genug erreicht werden [AS04].

Ein zweites Problemfeld stellt die fehlende Integrationsfähigkeit bestehender Anwendungssysteme dar, die in einer technischen Heterogenität begründet liegt. Ursachen sind unterschiedliche technische Plattformen, aber auch nicht existierende beziehungsweise nicht standardisierte Schnittstellen zur Verbindung der Software-Systeme untereinander. In der Vergangenheit wurde zur Lösung von Integrationsfragestellungen *Middleware*-Technologie eingesetzt, die als Bindeglied zwischen den vorhandenen Anwendungssystemen eingefügt wird und die Kommunikation und den Datenaustausch untereinander ermöglicht. Die meisten dieser *Middleware*-Ansätze haben sich jedoch nicht am Markt durchsetzen können. Bei der von der *Object Management Group* (OMG) spezifizierten *Common Object Request Broker Architecture* (CORBA) [OMG-CORBA] haben sich die Komplexität und der damit einhergehende Umsetzungsaufwand als zu hoch erwiesen [He06, Re06], andere Lösungen wie das von Microsoft spezifizierte *Distributed Component Object Model* (DCOM) [MS-DCOM] oder das von Sun spezifizierte Jini-Framework [SUN-Jini] konnten sich ebenfalls nicht nachhaltig etablieren. All diese Ansätze haben gemeinsam, dass die Integrationsfragestellung auf einer hauptsächlich technologischen Ebene betrachtet wird [Ke02].

Dienstorientierte Architekturen (engl. *Service-Oriented Architecture*, SOA) stellen eine Weiterentwicklung bestehender Software-Architekturen für verteilte Anwendungen dar, die das Ziel verfolgt, die vorgenannten Defizite zu überwinden. Als wesentlicher Unterschied zu *Middleware*-basierten Ansätzen der Anwendungsintegration stehen hier nicht länger die Anwendungssysteme und damit die Technologien im Vordergrund, sondern die zu unterstützenden Geschäftsprozesse [Ar04]. Ziel ist es, eine unmittelbare Abbildung von formalisierten Geschäftsprozessen auf Rechnerstützung durchzuführen, sodass einerseits eine größtmögliche Flexibilität auf Software-Ebene hinsichtlich Änderungen in den Geschäftsprozessen besteht [MP04] und andererseits die Komplexität der heterogenen Anwendungs- und Systemlandschaft verschattet wird. Neben der Prozessorientierung ist ein weiterer zentraler Beitrag dienstorientierter Architekturen die Vereinfachung bei der Integration der Anwendungen. Dies wird erreicht durch die Kapselung von Anwendungslogik in geschäftsrelevanten Diensten mit

standardisierten Dienstschnittstellen. Die dienstorientierte Bereitstellung ermöglicht eine bessere Wiederverwendungsmöglichkeit von Fachfunktionalität in verschiedenen Kontexten [DJ+05].

Mehr noch als die bestehenden, in sich geschlossenen Software-Systeme erfordern die verteilten und über Schnittstellen lose gekoppelten, dienstorientierten Architekturen den integrierten Einsatz von Sicherheitsmaßnahmen zur Absicherung von schützenswerten Ressourcen [Os05]. Der Begriff „Sicherheit“ ist sehr weit gefasst und beinhaltet zahlreiche Aspekte, unter anderem das Identitätsmanagement (IdM) und die Zugriffskontrolle [SUN-NI]. Aufgabe des Identitätsmanagements ist es, die digitalen Identitäten der Angehörigen eines Unternehmens zur Verfügung zu stellen, die als Grundlage für Zugriffskontrolle dienen. Die Umsetzung effektiver Zugriffskontrolle ist für sichere Geschäftsprozesse unverzichtbar, da diese voraussetzen, dass genau geregelt werden kann, welche Benutzer Aktionen in einem Prozess ausführen dürfen [Ku05]. Wichtig in diesem Zusammenhang ist, dass der Einsatz von Sicherheitsmaßnahmen für die Benutzer nicht als lästig empfunden wird. Für die Einfachheit der Nutzung von heterogenen Anwendungslandschaften steht der Begriff des *Single Sign-On* (SSO) [GK+03]. SSO impliziert, dass aus Sicht des Benutzers nur ein einziger Anmeldevorgang erforderlich ist, um alle angebotenen Funktionalitäten, für die der Benutzer über entsprechende Berechtigungen verfügt, in Anspruch nehmen zu können. Dies stellt insbesondere durch die Einbindung verschiedener bestehender Anwendungssysteme mit jeweils integrierten, aber individuellen Zugriffskontroll-Verfahren eine Herausforderung dar.

Durch die Bereitstellung von Fachfunktionalität an (geschäftsrelevanten) Dienstschnittstellen verwischen die bisherigen „harten“ Systemgrenzen. Für den Kontext der dienstorientierten Architekturen muss die Zugriffskontrolle auf Ebene der bereitgestellten (Einzel-)Dienste und Dienstkompositionen explizit berücksichtigt werden. Die wesentliche Aufgabe von Zugriffskontrolle ist es zu entscheiden, ob ein Benutzer einen angefragten Dienst nutzen darf oder nicht. In den klassischen Software-Systemen wird die Zugriffskontrolle innerhalb der Systemgrenze durchgeführt. Diese vergleichsweise „harten“ Systemgrenzen werden jedoch durch die Einbindung von klassischen Anwendungssystemen in dienstorientierte Architekturen über rein fachfunktional ausgeprägte Dienstschnittstellen aufgebrochen. Dies ist bei der Einführung dienstorientierter Architekturen zu beachten, denn oft werden in diesem Rahmen Sicherheitsaspekte bis zur Inbetriebnahme ausgeblendet. Im schlechtesten Fall existieren damit

nach wie vor für jede Gruppe von Diensten eines Altsystems eine eigene Benutzerverwaltung und eine isolierte Zugriffskontrolle. Dies würde eine ungünstige Insellösung darstellen, die wiederum integriert werden müsste. Wesentlich sinnvoller ist es, die Sicherheitsanforderungen bereits von Anfang an zu berücksichtigen, um dadurch das Ziel eines architekturweiten, integrierten Sicherheitskonzeptes zu verfolgen [Ku05]. Hierzu ist eine Zugriffskontroll-Architektur notwendig, die über die bestehenden Anwendungssysteme hinweg nicht nur, wie heute oft üblich, ein systemübergreifendes Benutzerverzeichnis bereitstellt, sondern alle zur effektiven Durchführung von Zugriffskontrolle notwendigen Daten und Algorithmen kapselt. Kernbausteine der Architektur sind einerseits ihre internen Komponenten und Schnittstellen. Das Anbieten von Zugriffskontrolle als gekapselter Dienst (engl. *Identity as a Service*) [Ni04, Ni06] macht es andererseits erforderlich, auch die entsprechenden Autorisierungsdaten in Verbindung mit den digitalen Identitäten der Benutzer zu spezifizieren und vorzuhalten.

Zusammenfassend lässt sich konstatieren, dass eine durchdachte Zugriffskontroll-Architektur, definiert durch ihre internen Komponenten, Schnittstellen, Datenstrukturen und Algorithmen, eine essenzielle Voraussetzung für die Einführung von dienstorientierten Architekturen ist [Bl03, Gr05, Ba07].

1.2 Betrachtetes Szenario

Dienstorientierte Architekturen existieren in unterschiedlichen Entwicklungsstufen, die hinsichtlich ihrer Mächtigkeit aufeinander aufbauen. In der einfachsten Umsetzungsform werden individuelle und proprietäre Kommunikationsbeziehungen zwischen Anwendungssystemen auf einheitliche (Dienst-)Schnittstellen und Kommunikationsprotokolle angehoben. Dies entspricht den Herausforderungen der klassischen Anwendungsintegration (engl. *Enterprise Application Integration*, EAI). In einer weiteren Stufe ermöglicht die Bereitstellung einer hinreichend großen Menge an Diensten aus verschiedenen Anwendungssystemen die explizite Komposition von Fachfunktionalität über die Grenzen von Anwendungssystemen hinweg. Dieser Aspekt kann wiederum von einer technisch motivierten Integration erweitert werden zu einer unmittelbaren Abbildung von (allgemeinen) Geschäftsprozessen auf Dienstkompositionen. Hierbei sind zusätzlich Anforderungen wie langlaufende Prozesse und Benutzerinteraktionen zu berücksichtigen. Auch der organisatorische Rahmen, in dem eine dienstorientierte Architektur betrieben wird, spielt eine Rolle. Sinnvollerweise beginnt die

Umsetzung der Architektur zunächst unternehmensintern. Obwohl auch innerhalb eines Unternehmens verschiedene Organisations- und Verantwortungsbereiche existieren, bestehen dennoch ein gemeinsamer Schutzbereich und einheitliche, unternehmensweise Sicherheitsrichtlinien. Dienstorientierte Architekturen müssen jedoch an Unternehmensgrenzen nicht Halt machen. Sie sollen in einer weiteren Ausbaustufe auch die Geschäftsprozessabwicklung zwischen verschiedenen Unternehmen (engl. *Business to Business*, B2B) abdecken. Gerade aus dem Blickwinkel der Zugriffskontrolle auf die in diesem Kontext genutzten Dienste und des dafür notwendigen, föderierten Identitätsmanagements stellen sich jedoch weitergehende Anforderungen [Ho07].

Um die im nachfolgenden Kapitel aufgezeigte Problemstellung einordnen zu können, wird in diesem Kapitel in das betrachtete Szenario eingeführt. Dadurch findet insbesondere eine Eingrenzung des betrachteten, fachlichen Rahmens der dienstorientierten Architektur statt.

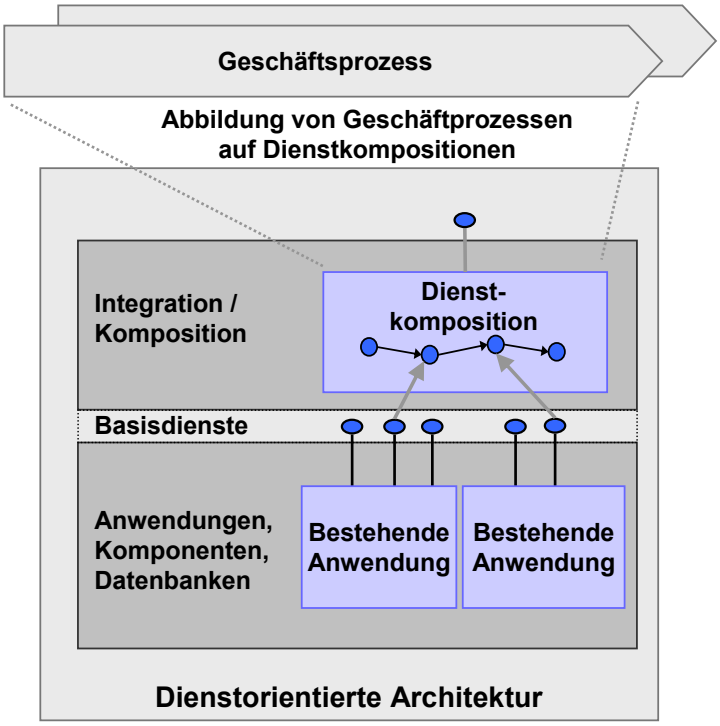


Abbildung 1: Kern einer dienstorientierten Architektur

Abbildung 1 gibt einen Überblick über den Aufbau dienstorientierter Architekturen, der in dieser Arbeit als Ausgangspunkt für die Erweiterung der Facharchitektur um Zugriffskontrolle genutzt wird. Eine dienstorientierte Architektur lässt sich durch eine Aufteilung in logische Schichten darstellen. Die unterste Schicht

fasst dabei die bereits bestehenden Software-Systemen (auch: Altsysteme, engl. *Legacy System*) zusammen, die weiter betrieben werden können und sollen. Die vollständige und zeitgleiche Umstellung (engl. *Big Bang*) aller bestehenden Anwendungen auf die neuen Herangehensweisen und technologischen Rahmenbedingungen sind in dienstorientierten Architekturen ausdrücklich weder gefordert noch notwendig [KB+05]. Stattdessen wird zur weiteren Nutzung der bestehenden Anwendungssysteme die existierende technische Heterogenität in einer virtuellen Grenzschicht vereinheitlicht. In dieser logischen Schicht der Basisdienste findet die dienstorientierte Bereitstellung von Fachfunktionalität aus den bestehenden Anwendungssystemen statt. Diese wird in geschäftsrelevanten Basisdiensten an technologisch einheitlichen Schnittstellen zur Verfügung gestellt [AC+04].

Aus technischer Sicht wird im betrachteten Szenario davon ausgegangen, dass die bestehende Funktionalität der Altsysteme über sogenannte *Application Server* als Webservices zur Verfügung gestellt wird (engl. *Wrapping to Web Services*). Es wird jedoch nicht die gesamte Anwendungsfunktionalität der bestehenden Systeme durch Webservices bereitgestellt, sondern bedarfsorientiert und schrittweise vorgegangen. Sukzessive wird die Funktionalität mit Wiederverwendungscharakter oder Integrationspotenzial über Stellvertreterobjekte (engl. *Proxy*) in einem *Application Server* bereitgestellt. Die *Application Server* sind dabei lediglich ein Mittel zum operativen Betrieb von Webservices und sollen im Grundsatz austauschbar sein. Während dies auf fachlicher Seite ermöglicht, verschiedene *Application Server* für den Betrieb der entwickelte Webservices zu nutzen, stellt dies jedoch auch Anforderungen hinsichtlich der Ankopplung einer übergreifenden Zugriffskontroll-Architektur.

Diese durch die *Application Server* bereitgestellten Basisdienste bilden an sich zwar noch keine Geschäftsprozesse ab, ermöglichen jedoch durch die Standardisierung und Herstellerunabhängigkeit der Webservice-Schnittstellen eine vereinfachte Integration auf Software-Ebene, da die Komposition dieser Dienste einheitlich und außerhalb der Anwendungssysteme durchgeführt werden kann. Dadurch entsteht eine Unabhängigkeit von den Details der Implementierung des jeweiligen Software-Systems [ZT+05]. Die Integrations- und Kompositionsschicht bildet den Kern von dienstorientierten Architekturen [EL+06]. Sie stellt das Bindeglied zwischen Geschäftsprozessen einerseits und Rechnerunterstützung andererseits dar und bietet zwei Möglichkeiten: Einerseits vereinfacht sie die technische Integration durch standardisierte Schnittstellen und einheitliche

Kommunikationsprotokolle. Andererseits bietet sie aus geschäftsorientierter Sicht den Ausgangspunkt, um durch eine formale Beschreibung der Geschäftsprozesse ein „Programmieren im Großen“ zu erreichen [Le03]. Programmieren im Großen impliziert eine Transformation von formalisierten Geschäftsprozessen zu Prozessartefakten in Form von ausführbaren Dienstkompositionen.

Im Rahmen dieser Arbeit wird ein unternehmensinternes Migrationsszenario analog der vorangegangenen Beschreibung betrachtet. Fachliche Funktionalität aus bestehenden Anwendungen soll an Dienstschnittstellen zur Verfügung gestellt werden, um eine anwendungsübergreifende Integration anhand formalisierter Geschäftsprozesse zu ermöglichen. Der Fokus liegt dabei auf den vollständig automatisierbaren Prozessen, die von ihrer Charakteristik weder langlaufend sind noch über eine über den Start des Prozesses hinausgehende Benutzerinteraktionen verfügen. Diese Integration fachlicher Funktionalität aus heterogenen Anwendungen soll durch explizite Dienstkompositionen außerhalb der jeweiligen Anwendungssysteme umgesetzt werden. Für die Umsetzung des fachlichen Teils ohne Erweiterung um Aspekte wie Zugriffskontrolle findet dieser Ansatz bereits technologische Unterstützung, wie der Autor in [EM+05] und [EW+06] exemplarisch gezeigt hat. Bewusst ausgeblendet werden unternehmensübergreifende, dienstorientierte Architekturen, die gerade hinsichtlich der Realisierung von Zugriffskontrolle andere Schwerpunktfragestellungen beinhalten wie das Aufbauen und Umsetzen von Vertrauensbeziehungen als Grundlage für föderiertes Identitätsmanagement. Als konkrete Ausprägung des Szenarios wird exemplarisch die Migration der bestehenden Anwendungslandschaft der Universität Karlsruhe in eine dienstorientierte Architektur betrachtet, die durch das Projekt „Karlsruher integriertes InformationsManagement“ (KIM) [UKA-KIM] vorangetrieben wird.

1.3 Problemstellung und Zielsetzung

Eine Migration bestehender Anwendungssysteme in dienstorientierte Architekturen hat bereits begonnen und wird in den nächsten Jahren sukzessive weitergeführt werden. Die Entwicklung neuer Software-Systeme wird mehr und mehr auf Basis von Diensten und Dienstkompositionen durchgeführt, und auch bestehende Anwendungen müssen hinsichtlich dieser Anforderungen angepasst werden. Die Einführung dienstorientierter Architekturen in Unternehmen setzt zwar keine vollständige Adaption aller bestehenden Anwendungssysteme voraus. Bei der Bereitstellung von Fachfunktionalität von bestehenden Anwendungssystemen an

Dienstschnittstellen wird jedoch davon ausgegangen, dass die in den Systemen integrierte Querschnittsfunktionalität wie beispielsweise Zugriffskontroll-Komponenten in den jeweiligen Systemen verbleiben und aufgrund ihrer Isoliertheit und Heterogenität nicht ohne Weiteres anwendungsübergreifend weiter genutzt werden können. Zur Einführung einer dienstorientierten Architektur ist es daher von Vorteil, wenn initial eine geeignete Infrastruktur entworfen und zur Verfügung gestellt wird. Solch einer Infrastruktur wird üblicherweise ein Minimum an Querschnittsdiensten wie ein Dienstverzeichnis und eine Zugriffskontroll-Architektur zugeordnet [Ar04]. Idealerweise ist die Zugriffskontroll-Architektur selbst dienstorientiert aufgebaut. Dies bedeutet, dass sie ihre Funktionalität an klar definierten, stabilen und herstellerunabhängigen Schnittstellen anbietet. Dadurch wird eine Verschattung der umsetzenden Produkte und Technologien erreicht, was eine lose Kopplung mit der fachfunktionalen Seite der dienstorientierten Architektur ermöglicht, damit künstliche Abhängigkeiten vermeidet sowie die Austauschbarkeit der implementierenden Komponenten und Produkte erlaubt.

Das übergreifende Ziel und damit der Leitgedanke dieser Arbeit ist die Ermöglichung von Zugriffskontrolle in dienstorientierten Architekturen, die aus einer Vielzahl elementarer Webservices und Webservice-Kompositionen besteht, wobei unterschiedliche *Application Server* genutzt werden, um diese zu betreiben. Fachliche Webservices sollen mit einem Zugriffskontroll-Dienst komponiert werden. Die dadurch entstehende Verbindung eines fachlichen Webservices mit einem Zugriffskontroll-Dienst soll zu einer betriebenen, fachlichen Einheit führen, die unmittelbar um Zugriffskontrolle ergänzt wurde. Zur Erreichung dieses Ziels werden nachfolgend zwei Teilziele mit jeweils einer Zielerweiterung motiviert.

Ziel Z1: Entwurf einer unternehmensweiten, integrierten Zugriffskontroll-Architektur

Webservices und Webservice-Kompositionen werden bereits prototypisch eingesetzt. Es fehlen jedoch noch Komponenten, die diese rein fachlich geprägten Dienste um Zugriffskontrolle ergänzen. Um die Prinzipien der dienstorientierten Facharchitektur analog auf die querschnittlich genutzte Zugriffskontroll-Architektur abzubilden, ist eine Verschattung der Zugriffskontroll-Funktionalität hinter klar spezifizierten Dienstschnittstellen anzustreben. Neben den Schnittstellen als statischem Teil sind die Interaktionsbeziehungen zwischen Facharchitek-

tur und Zugriffskontroll-Architektur zu definieren. Unter Berücksichtigung des definierten Szenarios, in dem bewusst keine Festlegung auf einen festen *Application Server* getroffen wurde, ist es sicherzustellen, dass die verknüpfenden Komponenten auf Seiten der Facharchitektur mit einem größtmöglichen, plattformunabhängigen Wiederverwendungsgrad (Portabilität) ausgestattet sind.

Zielerweiterung ZE1.1: Integration verteilt vorliegender Benutzerdaten

Das Ziel Z1 umfasst neben den Kernkomponenten auch die Einbindung der Benutzerdaten, die zur Durchführung von Zugriffskontroll-Entscheidungen benötigt werden. Eine integrierte Sicht auf die Benutzerdaten ist eine wesentliche Voraussetzung zur Entwicklung einer übergreifenden Zugriffskontroll-Architektur. Benutzerdaten liegen üblicherweise verteilt und oft inkonsistent in verschiedenen Benutzerverzeichnissen und Anwendungssystemen vor. Für die Integration von Benutzerverzeichnissen aus technischer Sicht gibt es bereits verschiedene Ansätze von Meta-Verzeichnissen über virtuelle Verzeichnisse bis hin zu proprietären Entwicklungen. Diese Ansätze nutzen feste Abbildungsregeln von einem Modell in ein anderes. Insbesondere die explizite Formulierung der semantischen Zusammenhänge der Repräsentation der digitalen Identitäten in den verschiedenen Datenquellen kommt dabei zu kurz. Ein Integrationsansatz, der die Semantik explizit formalisiert, ist anzustreben.

Ziel Z2: Formalisierung von Autorisierungsdaten für den Kontext Webservice-basierter, dienstorientierter Architekturen

Autorisierungsdaten stellen einen wesentlichen Baustein einer dienstorientierten Zugriffskontroll-Architektur dar; sie bilden die Grundlage einer Zugriffskontroll-Entscheidung. Zur Realisierung von Zugriffskontrolle sind daher neben der Entwicklung der notwendigen Komponenten ein geeignetes Zugriffskontroll-Modell sowie eine zugehörige Sprache zu spezifizieren, die die Formulierung von Zugriffskontroll-Policies ermöglicht. Diese Sprache darf nur in soweit plattformspezifisch sein, als dass sie sich auf dienstorientierte Architekturen auf Basis von Webservices als zu schützende Objekte bezieht. Sie sollte die Komplexität der einzusetzenden Sicherheitsprodukte abstrahieren und es damit einem Entwickler von fachlichen Webservices ermöglichen, Zugriffsbeschränkungen hinsichtlich der Webservices formal zu spezifizieren. Die Sprache selbst soll

jedoch keine weitere Spezifik hinsichtlich einer konkreten Fachdomäne aufweisen.

Zielerweiterung ZE2.1: Modellgetriebene Entwicklung der Zugriffskontroll-Policies

Das Ziel Z2 stellt ein Zugriffskontroll-Modell und eine zugehörige Policy-Sprache in den Vordergrund. Die hier vorgestellte Erweiterung von Z2 verfolgt das Ziel, die Policy-Erstellung in den Software-Entwicklungsprozess einzubinden. Die frühzeitige Behandlung von Sicherheitsaspekten in Software-Entwicklungsprozessen wird in aktuellen wissenschaftlichen Veröffentlichungen regelmäßig angemahnt. Unter anderem widmet sich mit [Lo03] eine ganze Dissertation diesem Thema. Modellgetriebene Ansätze zur Spezifikation von Zugriffskontroll-Policies ermöglichen durch die Abstraktion produktspezifischer Details und die damit einhergehende Reduktion technischer Komplexität, dass sich auch die Entwickler der fachlichen Webservices mit dem Themenkomplex formal auseinandersetzen können. Die Zielerweiterung lässt sich in zwei Teilaspekte aufteilen: Einerseits sollen Zugriffskontroll-Policies nicht erst bei Auslieferung beziehungsweise Inbetriebnahme (engl. *Deployment*) definiert werden, sondern bereits während des Software-Entwicklungsprozesses parallel zur Entwicklung der Fachfunktionalität. Andererseits soll es möglich sein, die Policies unabhängig von der eingesetzten Zugriffskontroll-Architektur und damit zunächst plattformunabhängig zu formalisieren und dann durch Transformation spezifische Ausprägungen einer konkreten Zugriffskontroll-Architektur zu erhalten.

1.4 Zusammenfassung der Ergebnisse

Abbildung 2 stellt grafisch die Ergebnisse dar, die zur Erreichung der vorgenannten Ziele geschaffen wurden. Das übergreifende Ergebnis, in das sich auch die nachfolgenden Arbeiten einordnen lassen, stellt eine dienstorientierte Zugriffskontroll-Architektur als Erweiterung des fachlichen Teils einer Webservice-orientierten Architektur dar. Der erste der beiden Hauptbeiträge dieser Arbeit liegt in der **Spezifikation einer dienstorientierten Zugriffskontroll-Architektur**, die die dienstorientierte Architektur erweitert. Sie ist maßgeblich durch den **dienstorientierten *Policy Decision Point* (PDP)** nach außen repräsentiert, der die Schnittstelle der Zugriffskontroll-Architektur gegenüber der Facharchitektur darstellt. Als unmittelbares Gegenstück zum PDP wird ein universell

wiederverwendbarer, also **portabler *Policy Enforcement Point* (PEP)** als Bindeglied zwischen Facharchitektur und Zugriffskontroll-Architektur entwickelt. Der interne Aufbau der entworfenen Zugriffskontroll-Architektur versteht sich als Muster, das es ermöglichen soll, bestehende Lösungen und kommerzielle Sicherheitsprodukte zu integrieren. Der PEP ist in zwei Richtungen lose gekoppelt: auf der einen Seite in Richtung der *Application Server*, die als austauschbare *Container*-Elemente für die Bereitstellung der Webservices auf fachfunktionaler Seite der dienstorientierten Architektur benötigt werden. Auf diese Weise wird eine Herstellerunabhängigkeit erreicht, insbesondere hinsichtlich der *Application Server*. Andererseits sollte die Verbindung zwischen der fachfunktionalen Seite und der Zugriffskontroll-Architektur ebenfalls lose gekoppelt werden. Dies wird durch die Spezifikation von einheitlichen Dienstschnittstellen erreicht, welche zusätzlich die Komplexität innerhalb der Zugriffskontroll-Architektur für die Entwickler auf der fachfunktionalen Seite verschatten. Es muss lediglich gegen diese definierten und stabil gehaltenen Schnittstellen entwickelt werden.

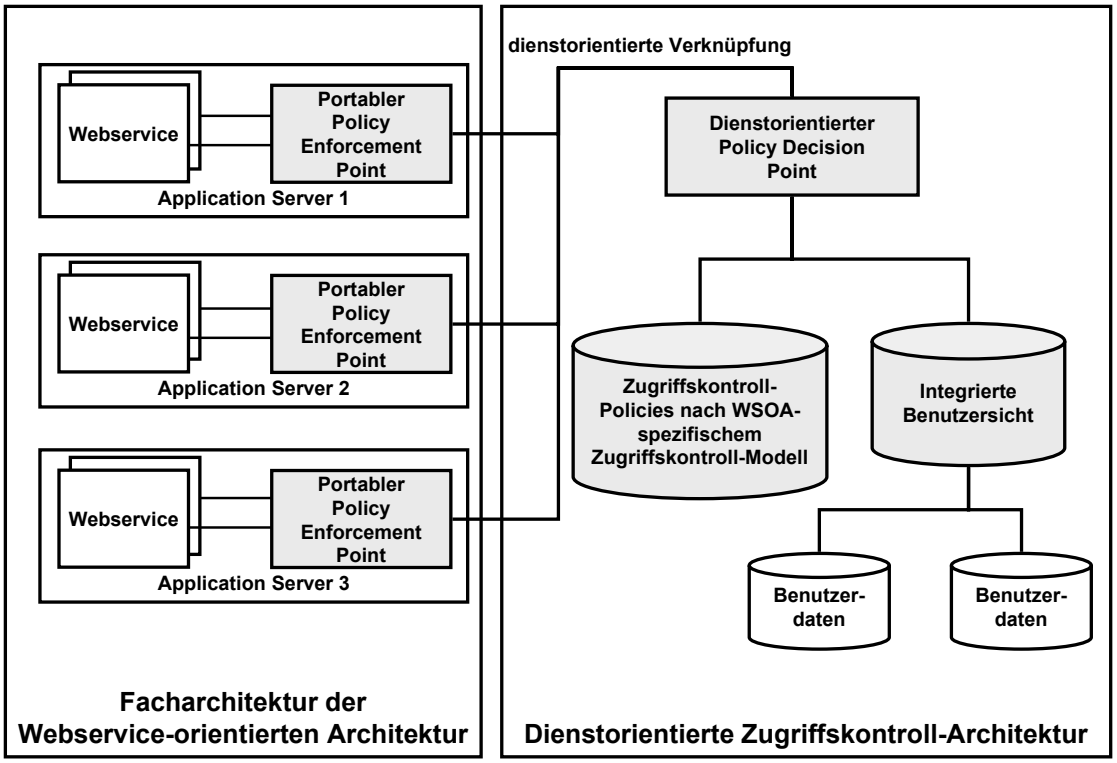


Abbildung 2: Zusammenfassung der Ergebnisse

Der zweite Hauptbeitrag liegt in der Spezifikation eines **Zugriffskontroll-Modells** und einer **zugehörigen Sprache zur Darstellung von Zugriffskontroll-Policies für Webservice-orientierte Architekturen**. Policies können als Konfigurationsdaten verstanden werden, die nicht hart im fachfunkti-

onalen Programmcode fixiert sind. Analysiert man bestehende Zugriffskontroll-Modelle hinsichtlich ihrer Einsetzbarkeit in dienstorientierten Architekturen, so zeigt sich, dass bestehende Zugriffskontroll-Modelle nicht unmittelbar übertragbar sind [YT05]. Daher wird zunächst ein Zugriffskontroll-Modell entwickelt, das die Besonderheiten von Webservice-orientierten Architekturen berücksichtigt, insbesondere die Struktur und die Granularität der zu schützenden Objekte, aber auch die explizite Dienstkomposition innerhalb der dienstorientierten Architektur. Aufbauend auf dem Zugriffskontroll-Modell wird eine Sprache zur Formulierung entsprechender Zugriffskontroll-Policies für den Webservice-Kontext entwickelt. Diese sind unabhängig in Hinblick auf die tatsächlich eingesetzten Komponenten in der Zugriffskontroll-Architektur.

Die beiden vorgenannten Hauptbeiträge werden anschließend durch zwei Erweiterungen ergänzt:

Die erste Erweiterung befasst sich mit den Zugriffskontroll-Policies, für die ein Vorgehensmodell zur **modellgetriebenen Entwicklung von Zugriffskontroll-Policies** entworfen wurde. Ziel dabei ist, im Rahmen des Software-Entwicklungsprozesses bereits frühzeitig neben der Fachfunktionalität die zugehörigen Nutzungsbedingungen in Form von Zugriffskontroll-Policies formulieren zu können. Diese sollen während der einzelnen Phasen der Software-Entwicklung aufgegriffen werden und schrittweise modellgetrieben verfeinert und konkretisiert werden. Hierbei werden die Grundlagen der modellgetriebenen Architektur [OMG-MDA] eingesetzt, wobei die Artefakte in den Phasen plattformunabhängiges Modell (engl. *Platform-Independent Model*, PIM), plattformabhängiges Modell (engl. *Platform-Specific Model*, PSM) und der effektiv auswertbare Code (engl. *Platform-Specific Code*, PSC) betrachtet werden. Die Plattform stellt in diesem Kontext die konkrete Zugriffskontroll-Architektur dar. Dieses Vorgehen ermöglicht es, Zugriffskontroll-Policies zunächst unabhängig von den eingesetzten Produkten zur Realisierung von Zugriffskontrolle zu modellieren, sie dann in produktspezifische Policies zu transformieren, die unmittelbare Modelle des produktspezifischen Codes darstellen.

Die zweite Erweiterung befasst sich mit der Integration verteilt vorliegender Benutzerdaten. Trotz der fachlichen Heterogenität der einzelnen Anwendungssysteme nutzen sie Benutzerverzeichnisse, um Zugriffskontroll-Informationen zu speichern. Um diese „Identitätsinseln“ zu integrieren, wird in der Arbeit eine **semantische Integration der Benutzerdaten mittels Ontologien** vorgenom-

men. Dadurch werden die unterschiedlichen syntaktischen Repräsentationen in den Benutzerverzeichnissen homogenisiert und eine Synchronisation über die Verzeichnisse hinweg wird vereinfacht.

1.5 Prämissen der Arbeit

Die Forschungsgebiete der Zugriffskontrolle einerseits und von dienstorientierten Architekturen andererseits stellen bereits einzeln betrachtet umfangreiche Forschungsgegenstände dar. Für die Erarbeitung der Ergebnisse werden daher nachfolgende Einschränkungen gemacht und Annahmen getroffen.

Prämisse P1: Zugriffskontrolle und Identitätsmanagement

Sowohl die Zugriffskontrolle als auch das Identitätsmanagement werden dem Bereich der IT-Sicherheit zugeordnet. Das Identitätsmanagement umfasst die Verwaltung der personenbezogenen Daten der Mitarbeiter eines Unternehmens und bildet damit die Grundlage zur Durchführung von Zugriffskontrolle. Der Begriff des Identitätsmanagements wird in Literatur und Praxis oft unterschiedlich definiert. Teilweise findet sogar eine Synonymbildung der Begriffe „Zugriffskontrolle“ und „Identitätsmanagement“ statt. Im Rahmen dieser Arbeit umfasst der Begriff des Identitätsmanagements neben der Verwaltung der reinen Identitätsdaten auch alle notwendigen Prozesse und Architekturelemente, die zur Bereitstellung digitaler Identitäten dienen. Neben den Kernprozessen wie der Datenprovisionierung und -synchronisation gibt es auch ergänzende Prozesse wie Auditierungen, die die Einhaltung (engl. *Compliance*) der definierten Prozesse sicherstellen. Im Kontext dieser Arbeit werden die über die Kernprozesse hinausgehenden Aufgaben des Identitätsmanagements nicht betrachtet. Eine klare Trennung der Daten der digitalen Identität in diejenige, die rein identitätsbezogen sind und diejenige, die nur relevant für die Durchführung von Zugriffskontrolle sind, ist heutzutage weder sinnvoll noch förderlich. Es können im Gegenteil alle zur Verfügung stehenden Daten einer digitalen Identität zur Entscheidung einer Zugriffskontroll-Anfrage notwendig sein und müssen daher genutzt werden können. Diese Daten dienen dann als Ausgangspunkt für Zugriffskontrolle, die im Kern sicherstellt, dass nur autorisierte Zugriffe eines Subjekts (repräsentiert durch seine digitale Identität) auf ein Objekt zugelassen werden.

Prämisse P2: Unternehmensinterne Architektur

Für die Realisierung von Zugriffskontrolle wird im Rahmen dieser Arbeit der Kontext einer in sich geschlossenen Unternehmung betrachtet. Aspekte wie föderiertes Identitätsmanagement (engl. *Federated Identity Management*, FIM) in unternehmensübergreifenden Formen der Zusammenarbeit (engl. *Business to Business*, B2B) werden in dieser Arbeit nicht betrachtet. Zwar ist es auch in Unternehmungen üblich, sich in der Aufbauorganisation durch getrennte Organisationseinheiten zu gliedern, die mit eigenen Kompetenzen hinsichtlich des Einsatzes von Rechnerunterstützung ausgestattet sind. Jedoch gelten in solchen Organisationsformen andere Rechtsbestimmungen hinsichtlich der Nutzung von personenbezogenen Daten, die insbesondere für das Identitätsmanagement und die Zugriffskontrolle eine essenzielle Basis darstellen.

Prämisse P3: Ausprägungsstufe der dienstorientierten Architektur

Dienstorientierte Architekturen bestehen in unterschiedlichen Ausprägungsstufen. Im Kontext dieser Arbeit wird analog zur Beschreibung des Szenarios in Kapitel 1.2 eine Facharchitektur betrachtet, die sich auf die Abbildung von voll automatisiert durchführbaren Geschäftsprozessen fokussiert. Langlaufende Prozesse, insbesondere solche mit Benutzerinteraktion, die über das Starten des Prozesses hinausgeht, werden in dieser Arbeit nicht betrachtet.

Prämisse P4: Webservice-orientierte Architektur

Es existieren verschiedene Ausprägungsformen von dienstorientierten Architekturen, wie beispielsweise DCOM, CORBA, Jini oder Webservice-basierte Lösungen. In den letzten Jahren haben sich Webservices als vielversprechende und derzeit auch meist genutzte Implementierungsform ausgezeichnet, unter anderem durch die bewusste Herstellerunabhängigkeit, aber auch durch die Unkompliziertheit in der konkreten Realisierung [DJ+05]. Diese Arbeit geht daher auf fachfunktionaler Seite von einer Webservice-orientierten Architektur (WSOA) aus. Trotzdem lassen sich viele Ergebnisse dieser Arbeit auch auf andere Ausprägungsformen übertragen.

Prämisse P5: Querschnittseigenschaften

Die Bereitstellung von Fachfunktionalität als Dienst impliziert neben der eigentlichen Programmlogik einen qualitätsgesicherten Betrieb. Hierzu gehören weitere Querschnittseigenschaften, die im Rahmen von Dienstleistungsvereinbarungen (engl. *Service Level Agreements*, SLA) vereinbart werden [Ma01]. Von dieser Querschnittsfunktionalität wird in dieser Arbeit nur die Teilmenge der Anforderungen betrachtet, die einen Einfluss auf die Zugriffskontrolle haben.

Prämisse P6: Anwendungssystem

Der Begriff „Anwendungssystem“ wird entsprechend der Begriffsdefinition für (Software-)Anwendungssysteme des Informatik-Begriffsnetzes der Gesellschaft für Informatik [GI-IB] verwendet und im Rahmen dieser Ausarbeitung synonym genutzt zu den Begriffen „Anwendung“ und „Software-System“. Ein Anwendungssystem ist in diesem Kontext definiert als ein System, das Software-Komponenten enthält. Im weiteren Sinne umfasst es eine Menge von inhaltlich zusammengehörigen Aufgaben, die dafür verantwortlichen Menschen als Aufgabenträger und die zu ihrer Erfüllung eingesetzte technische Ausstattung.

1.6 Aufbau der Arbeit

Die vorliegende Arbeit ist in neun Kapitel gegliedert, wie in Abbildung 3 dargestellt.

Kapitel 1 motiviert den Themenbereich der dienstorientierten Architekturen und führt in die Problemstellung ein. Es werden die Prämissen dieser Arbeit definiert und der Lösungsansatz skizziert.

In **Kapitel 2** werden die Begriffe und weiteren Grundlagen für diese Arbeit definiert, soweit sie für das Verständnis der Arbeit erforderlich sind.

Der Stand der Forschung und Technik wird in **Kapitel 3** thematisiert. Ausgehend von einem Anforderungskatalog werden die erarbeiteten Grundlagen aufgegriffen und ihre Anwendung in Forschung und Praxis bewertet. Ausgehend von den herausgearbeiteten Defiziten wird ein Handlungsbedarf motiviert, der durch die nachfolgenden Kapitel bearbeitet wird.

Mit dem Entwurf einer Architektur für dienstorientierte Zugriffskontrolle für Webservice-orientierte Architekturen werden in **Kapitel 4** ihre Komponenten entworfen. Neben dem dienstorientiert aufgebauten *Policy Decision Point* wird ein universell einsetzbarer *Policy Enforcement Point* eingeführt, der ohne große Anpassungen in *Application Server* unterschiedlicher Hersteller einsetzbar ist. Ebenfalls wird die Verknüpfung von fachlichem Dienst, PEP und PDP zur Realisierung der Zugriffskontroll-Funktionalität dargestellt.

Kapitel 5 widmet sich der Erstellung eines Zugriffskontroll-Modells für Webservice-orientierte Architekturen. Es wird mit der *Web Services Access Control Markup Language* (WSACML) eine Sprache vorgestellt, die lediglich die notwendige Abhängigkeit zur Facharchitektur besitzt, nicht jedoch zu einem möglichen Produkt, das zur Implementierung auf Seiten der dienstorientierten Architektur eingesetzt wird. WSACML trägt insbesondere dem Aspekt der Vergröberung der Dienstschnittstellen und der zusammengesetzten Dienste Rechnung.

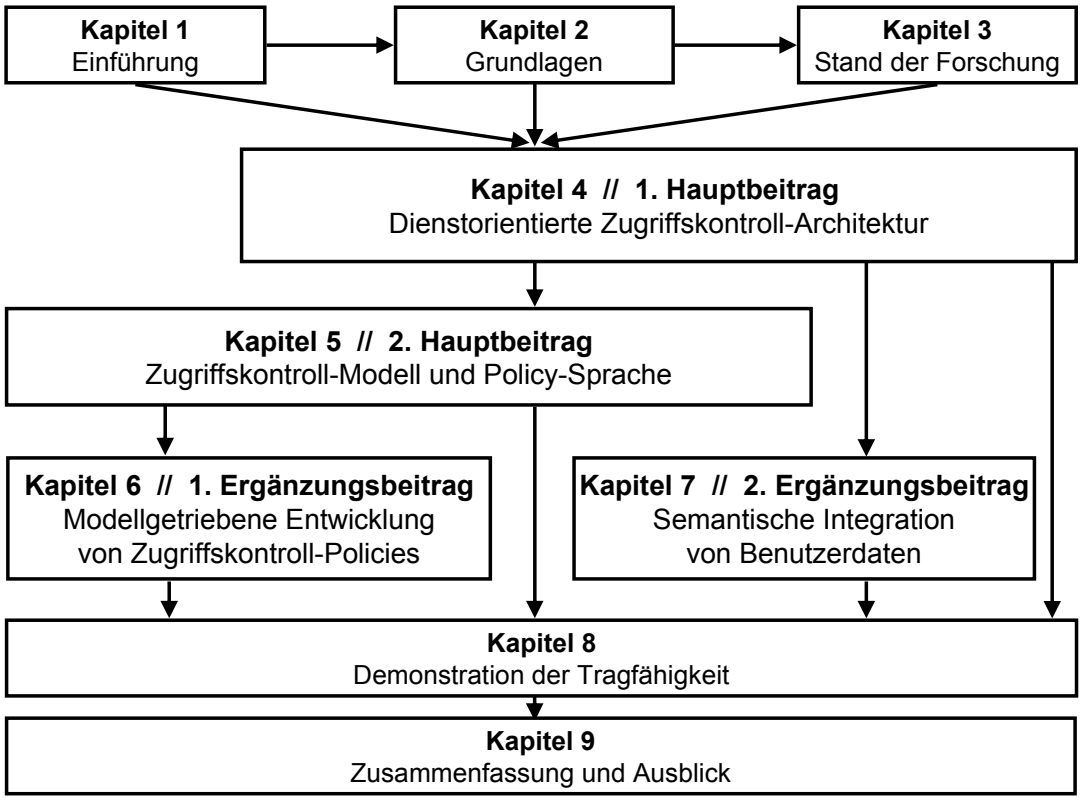


Abbildung 3: Aufbau der Arbeit

Dieses Zugriffskontroll-Modell wird dann in **Kapitel 6** genutzt, um Zugriffskontroll-Policies modellgetrieben entwickeln zu können. Dargestellt wird

ein Vorgehen, wie Policies bereits während des fachfunktionalen Software-Entwicklungsprozesses entwickelt werden können, wobei dies unabhängig von der eingesetzten Technologie der Zugriffskontroll-Architektur stattfindet.

Kapitel 7 wendet sich den Benutzerverzeichnissen zu, die ebenfalls Basiskomponenten einer Zugriffskontroll-Architektur darstellen. Üblicherweise existiert in Unternehmen eine große Zahl dieser Verzeichnisse, und es ist erforderlich, diese konsistent zu halten. In diesem Kapitel wird eine Lösung vorgestellt, die dieses mittels des Einsatzes von Ontologien erzielt.

In **Kapitel 8** erfolgt der Tragfähigkeitsnachweis der entwickelten Konzepte anhand der prototypischen Implementierung im Kontext von Forschungs- und Industrieprojekten.

In **Kapitel 9** werden die erzielten Ergebnisse zusammengefasst und bewertet. Die Arbeit schließt mit einem Ausblick auf offene und weiterführende Fragestellungen.

1.7 Typografische Konventionen und Rechtschreibung

In dieser Arbeit werden folgende Schriftarten verwendet:

- **Fettschrift** zur Hervorhebung wichtiger Sachverhalte,
- *kursive Schrift* bei der Verwendung von Wörtern und Wortgruppen, die im Sinne des Duden als aus einer fremden Sprache zitiert angesehen werden, sowie speziell gekennzeichnete Zitate. Eigennamen und produktspezifische Ausdrücke in fremder Sprache werden in dieser Arbeit ausdrücklich **nicht** kursiv geschrieben,
- `Schreibmaschinenschrift` zur Darstellung und Referenzierung von Quellcode.

Überschriften, Abbildungen und Tabellen wurden von den typografischen Konventionen ausgenommen.

Als Grundlage zur Rechtschreibung wird der zum Zeitpunkt der Erstellung dieser Dissertation gültige Duden [Du06] verwendet. Der Begriff „Policy“, der in dieser

Arbeit aufgrund seiner zentralen Bedeutung regelmäßig auftritt, wird zur Verbesserung des Leseflusses als deutscher Begriff behandelt und entsprechend nicht kursiv gestellt.

2 Grundlagen

In diesem Kapitel wird in die für das Verständnis dieser Arbeit benötigten Themengebiete eingeführt. Zunächst wird der Dienstbegriff definiert und in Bezug zu Webservice-basierten, dienstorientierten Architekturen gestellt. Danach folgt eine Einführung in den Bereich des Identitätsmanagements und der Zugriffskontrolle. Es schließen sich die Grundlagen der Modellierung und modellgetriebenen Entwicklung an. Darauf aufbauend werden Entwurfsmuster als wiederverwendbare Vorlagen in der Software-Entwicklung vorgestellt. Das Kapitel endet mit der Motivation von Ontologien zur formalen Modellierung von Semantik.

2.1 Dienste, Webservices und dienstorientierte Architekturen

Der Dienstbegriff stellt für diese Arbeit eine wichtige Grundlage dar. In diesem Abschnitt werden daher zunächst Definitionen für die Fachbegriffe „Dienst“ und „dienstorientierte Architektur“ gegeben und in die grundlegenden Technologien und Standards für Webservices als Realisierungsform dienstorientierter Rechnerunterstützung eingeführt.

2.1.1 Der Dienstbegriff

Eine allgemein gehaltene und nicht auf Rechnerunterstützung eingeschränkte Definition des Dienstbegriffs wird durch das Deutsche Institut für Normung in DIN EN ISO 8402:1994 beziehungsweise DIN EN ISO 9000:2005 gegeben. Demnach handelt es sich bei einem Dienst (beziehungsweise einer Dienstleistung) um

> *„das Ergebnis mindestens einer Tätigkeit, die notwendigerweise an der Schnittstelle zwischen dem Lieferanten und dem Kunden ausgeführt wird und üblicherweise immateriell ist.“*

Diese Definition zeigt, dass bei dienstorientierter Ausführung einer Arbeitsleistung nicht die Art und Weise der Durchführung, sondern das Ergebnis der Arbeit eines Dienstgebers (Lieferanten) gegenüber einem Dienstnehmer (Kunden) im Zentrum der Betrachtung steht.

Auch im Kontext der verteilten Software-Systeme wurde der Dienstbegriff bereits früh geprägt. Im OSI-Referenzmodell [Zi80] werden die Begriffe „Schicht“ (engl. *Layer*) und „Dienst“ (engl. *Service*) definiert. Zunächst wird zur Reduktion der Komplexität die technische (Kommunikations-)Infrastruktur auf mehrere überschneidungsfrei übereinanderliegende Schichten aufgeteilt. Jede dieser Schichten nutzt Dienste der unmittelbar darunterliegenden Schicht und bietet der jeweils darüberliegenden Schicht Dienste an. Die durch eine Schicht bereitgestellten Dienste werden an definierten Dienstzugangspunkten (engl. *Service Access Point*, SAP) zur Verfügung gestellt. Ziel dieser auf Diensten basierenden Verknüpfung der Schichten zu einem Gesamtsystem ist die Abschirmung der Schichten untereinander vor internen Details wie beispielsweise der konkreten Implementierung. Diese bleiben dadurch in dem Rahmen veränderbar, solange der entsprechende Dienstzugangspunkt wie spezifiziert bedient werden kann.

In einer Grundlagenarbeit zu (IT-)Diensten und Dienstmanagement [Ro03] wird der Dienstbegriff wie folgt verfeinert:

> *“A service is defined as a functionality that is provided with a certain quality and cost at a Service Access Point (SAP). […] The functionality of a service consists of two parts: (i) the functionality of the service itself, and (ii) the functionality of sub-services that are involved in the service provisioning with respect to the service hierarchy.”*

Durch diese Aussage werden der Aspekt der Verschattung von Interna der Dienstrealisierung hinter einem definierten Dienstzugangspunkt und die Möglichkeit der Dienstkomposition als Möglichkeit zur Bereitstellung von Diensten hervorgehoben. Zudem wird der Bezug zwischen Funktionalität und Qualität aufgezeigt.

Die unmittelbare Verknüpfung von Funktionalität mit Qualität wird in [HA+99] weitergehend vertieft. Demnach verknüpft der Dienstbegriff die durch den Dienst realisierte Fachfunktionalität mit einer zu vereinbarenden Dienstgüte (engl. *Quality of Service*, QoS). Dabei handelt es sich bei der Dienstgüte um eine typische Schnittstelleninformation zwischen Dienstgeber und Dienstnehmer. Hierzu können neben der Festlegung von Parametern wie Verfügbarkeit und Performanz auch weitere nicht funktionale Eigenschaften der Dienstschnittstelle

definiert werden. Bei Verwendung von Zugriffsbeschränkungen können dies beispielsweise Nutzungserlaubnisse und -einschränkungen sein. Diese bilden den Ausgangspunkt für Zugriffskontroll-Entscheidungen auf Dienstebene.

2.1.2 Webservices und ihre Basistechnologien

Dienstorientierte Architekturen sind nicht fest mit einer konkreten Technologie assoziiert [RH+05]. Trotz allem werden Webservices sowie ihre zugehörigen Technologien und Standards von Autoren aus den verschiedensten Bereichen als die Implementierungsform zukünftiger, dienstorientierter Anwendungen gesehen. Für das wissenschaftliche Umfeld kann stellvertretend für viele Publikationen der Artikel [KL04] genannt werden. Dort wird der Einsatz von Webservices als logische Fortführung der klassischen Anwendungsintegration mit explizitem Geschäftsprozessbezug betrachtet. Auch im kommerziellen IT-Umfeld, das durch starken Wettbewerb geprägt ist und das sich vor kostenintensiven Fehlschlägen schützen muss, sind Webservices das Medium der Wahl zur Realisierung dienstorientierter Architekturen [Re06].

Der Begriff „Webservice“ wurde vom *World Wide Web Consortium* (W3C) geprägt und hat über die Jahre eine Präzisierung erlebt. In der aktuellen Version des W3C-Glossars [HB04] findet sich folgende Definition:

> *“A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.”*

Das W3C nimmt in dieser Aussage eine ausschließlich technische Sicht auf Webservices ein. Zentrale Aussage ist, dass Webservices zur Realisierung von Kommunikationsbeziehungen zwischen Maschinen untereinander eingesetzt werden. Damit wird die Beziehung Mensch-Maschine explizit ausgeschlossen. Daraus lässt sich eine Weiterentwicklung klassischer Integrationsszenarien erkennen. Weiterhin wird auf die *Extensible Markup Language* (XML) [Ra02b] verwiesen, die eine elementare Grundlage im Webservice-Umfeld darstellt. Zusätzlich werden zwei Basisstandards benannt, die beim Einsatz von Webservi-

ces zu verwenden sind: die *Web Services Description Language* (WSDL) als Beschreibungssprache für die Schnittstellen von Webservices und SOAP (früher: *Simple Object Access Protocol*) als das Format zum Nachrichtenaustausch zwischen Webservices. SOAP setzt auf dem im Webumfeld bewährten *Hyper-Text Transfer Protocol* (HTTP) auf, um XML-basierte Nutzdaten zu übermitteln. Des Weiteren gibt es einen ebenfalls etablierten Basisstandard für den Bereich der Dienstverzeichnisse, *Universal Description, Discovery and Integration* (UDDI). Dieser wird dazu genutzt, Dienstbeschreibungen zu verwalten und zur Dienstsuche bereitzuhalten [Kr07b]. Im Gegensatz zu WSDL werden SOAP und UDDI im Rahmen dieser Arbeit keine größere Rolle spielen, weshalb auf eine weitergehende Einführung verzichtet wird. Informationen zu SOAP und UDDI finden sich in den zugehörigen Standards [W3C-SOAP1.2] und [OASIS-UDDI3.0].

Ein wesentliches Beschreibungselement von Webservices ist ihre Dienstschnittstelle, die mittels *Web Services Description Language* (WSDL) formalisiert wird. Bei der WSDL handelt es sich um eine XML-basierte Sprache, die die angebotenen Methoden (Operationen), Datentypen, Nachrichten und Kommunikationsprotokolle eines Webservices beschreibt. Im Jahr 2001 wurde der WSDL-Standard in der Version 1.1 durch das W3C verabschiedet [W3C-WSDL1.1], der im Juni 2007 durch die Version 2.0 [W3C-WSDL2.0] abgelöst wurde.

Der logische Aufbau einer WSDL-basierten Dienstbeschreibung beginnt beim Beschreibungselement Dienst (engl. *Service*), der die umschließende Klammer bildet und eine Menge von Kommunikationsendpunkten (engl. *Port*) darstellt. Die Endpunkte werden über eine Bindung (engl. *Binding*) an einen sogenannten PortTyp (engl. *PortType*) gebunden, der wiederum eine Menge von Methoden (engl. *Operation*) darstellt. Mit einer Methode können drei verschiedene Typen von Nachrichten (engl. *Message*) verbunden werden: eingehende und ausgehende Nachrichten sowie Fehlernachrichten. Die Inhalte einer WSDL-basierten Webservice-Beschreibung lassen sich in zwei Gruppen aufteilen, in die abstrakten und in die konkreten Definitionen. Abstrakte Definitionen betreffen die plattform- und sprachunabhängigen Festlegungen des Webservices. Hierzu zählen die Elemente *Type* für die maschinen- und sprachunabhängige Definition von Datentypen, *Message* für die Spezifikation der ausgetauschten Nachrichten, die ein Webservice akzeptiert beziehungsweise zurücksendet, *Operation* zur Beschreibung der Methodensignatur sowie *PortType* als Element zur Gruppierung von Operationen. Zu den konkreten Definitionen in einem WSDL-Dokument

werden die Elemente *Binding* zur Verknüpfung der Elemente *PortType* mit konkreten Protokollen wie beispielsweise SOAP und das Element *Service* zur Angabe der als *Uniform Resource Identifier* (URI) dargestellten Webadressen gezählt.

Die reine Nutzung der vorgenannten Basisstandards ermöglicht jedoch nicht automatisch eine Interoperabilität, da die Standards viele Freiheitsgrade offen lassen [EA+04]. Um dieses Problem anzugehen, wurde mit der *Web Services Interoperability Organization* (WS-I) ein Gremium geschaffen, das sich ausschließlich der Interoperabilität im konkreten Produkteinsatz annimmt. In [BE+04] findet sich das Basis-Profil, das zur Sicherstellung größtmöglicher Interoperabilität weitergehende Anforderungen an konkrete Implementierungen stellt.

2.1.3 Dienstorientierte Architekturen

Bei Diensten an sich handelt es sich um Einzelartefakte, die zu einem größeren Ganzen, beispielsweise zu einem Software-System, zusammengefügt werden können. In der Informatik wird die Strukturierung eines Software-Systems durch die sogenannte Software-Architektur beschrieben. Diese Architektur stellt die Systemkomponenten und ihre Beziehungen untereinander dar. Unter einer Systemkomponente wird ein abgegrenzter Teil eines Software-Systems verstanden. Beziehungen umfassen jede mögliche Art der Verbindung zwischen den Software-Komponenten [Ba00].

Eine sehr ähnliche Definition für den Begriff „Software-Architektur“ findet sich im Handbuch der Software-Architektur [RH06] und wird so auch im Glossar der GI Fachgruppe Software-Architektur [GI-SWA] aufgeführt:

> *„Die Software-Architektur ist die grundlegende Organisation eines Systems, dargestellt durch dessen Komponenten, deren Beziehungen zueinander und zur Umgebung, sowie die Prinzipien, die den Entwurf und die Evolution des Systems beeinflussen.“*

Wenn eine Software-Architektur sich aus Diensten zusammensetzt, wird von einer dienstorientierten Architektur gesprochen. Eine zusammenfassende Definition des Begriffs „dienstorientierte Architektur“ findet sich in [RH06]:

> *„Eine dienstorientierte Architektur* (service-oriented architecture) *liegt vor, wenn die Funktionalitäten in Form von Diensten gekapselt sind, die über standardisierte, publizierte Schnittstellen verfügen. Weiterhin müssen die so gekapselten Funktionalitäten lose gekoppelt und atomar sein.“*

Neben der Kapselung von Funktionalität wird in dieser Definition zusätzlich die lose Kopplung hervorgehoben. Der Grad der Kopplung zweier Komponenten wird charakterisiert durch die Menge der Vereinbarungen und Annahmen, die zwischen beiden Komponenten getroffen wurden [TS07]. Je mehr Abhängigkeiten zwischen Diensten bestehen, desto enger sind diese gekoppelt. Kopplung bezieht sich dabei auf alle Arten der Abhängigkeiten von Services und stellt eine relative Größe dar. Es gibt unterschiedliche Arten von Abhängigkeiten. Eine zeitliche Abhängigkeit beschreibt das Kommunikationsverhalten, das beispielsweise synchron (enger gekoppelt) oder asynchron (loser gekoppelt) sein kann. Eine Datenabhängigkeit entsteht, wenn ein Dienst eine (interne) Datenstruktur eines anderen Dienstes nutzt, beispielsweise einen Primärschlüssel einer Datenbank. Somit kann der Dienstgeber die interne Repräsentation dieser Daten nicht mehr ändern. Je loser die Kopplung, desto weniger Vereinbarungen, insbesondere auch über das Innenleben der Komponente, werden getroffen. Ein Minimum stellt die Reduktion der Spezifikation auf die Schnittstellensicht dar. Gerade durch die Dienstorientiertheit und die gewünschte Wiederverwendbarkeit von Diensten ist eine lose Kopplung der fachlichen Dienste in einer dienstorientierten Architektur ein Ziel. Das bedeutet, eine Verflechtung der einzelnen Dienste miteinander sollte vermieden werden. Dies wird auch durch die oben genannte Forderung nach Atomizität, also der in sich Abgeschlossenheit eines Dienstes bekräftigt.

Dienstorientierte Architekturen können auch als Weiterentwicklung der klassischen Anwendungsintegration (engl. *Enterprise Application Integration*, EAI) gesehen werden. Eine Prägung des Begriffs „dienstorientierte Architektur“ aus diesem Blickwinkel findet sich in [DJ+05]:

> *„Unter einer SOA versteht man eine Systemarchitektur, die vielfältige, verschiedene und eventuell inkompatible Methoden oder Applikationen als wiederverwendbare und offen zugreifbare Dienste repräsentiert und dadurch eine plattform- und sprachenunabhängige Nutzung und Wiederverwendung ermöglicht.“*

Es herrscht allgemeiner Konsens darüber, dass dienstorientierte Architekturen losgelöst von konkreten Technologien und Produkten zu betrachten sind [Er05]. Es handelt sich dabei um ein Paradigma, das bereits in der Entwurfs-Phase der Software-Entwicklung angewendet wird und durch unterschiedliche Technologien wie beispielsweise CORBA oder Webservices realisiert werden kann. In [EK+07] wird durch den Autor gezeigt, wie technologieunabhängig eine Software-Entwicklung für Dienste in einer dienstorientierten Architektur durchgeführt werden kann. Dabei wird eine Trennung zwischen dem abstrakten Dienst und seinen konkreten Instanzen durchgeführt sowie eine Möglichkeit geschaffen, Dienste und ihre Beziehungen untereinander explizit zu modellieren.

Dienstorientierte Architekturen betrachten jedoch nicht nur die Software und ihre Architektur an sich. Durch die Einführung von standardisierten Schnittstellen zur vereinfachten Anwendungsintegration werden zusätzliche Möglichkeiten geschaffen [BC04]. Ein wesentliches Ziel ist es, Geschäftsprozesse effizienter mit der zugehörigen Rechnerunterstützung zu verzahnen. Dies spiegelt sich auch in der Definition in [BB+05] wider:

> *„A service-oriented architecture is a framework for integrating business processes and supporting IT infrastructure as secure, standardized components – services – that can be reused and combined to address changing business priorities.“*

Durch diese explizite Verzahnung von Geschäftsprozessen und zugehöriger Rechnerunterstützung können sich unter anderem folgende Vorteile ergeben [AS04]:

- Ein prozessorientierter Ansatz sorgt dafür, dass die Geschäftsprozesse des Unternehmens im Mittelpunkt stehen. Damit wird verhindert, dass der technisch gegebene Istzustand definiert, wie eine optimale Rechnerunterstützung aussehen sollte und muss.

- Mit einer formal beschriebenen und ausführbaren Dienstkomposition können Änderungen in den Geschäftsprozessen schneller auf die Rechnerunterstützung abgebildet werden als bisher.

- Geschäftsprozesse können auch durch Nicht-Informatiker am „Reißbrett“ entworfen und konfiguriert werden. Formalisierte Prozessartefakte können

dann unmittelbar zur Ausführung gebracht werden, anstatt aufwendig in Quellcode entwickelt werden zu müssen.

- Teilprozesse können wiederverwendet und anderen Organisationseinheiten zur Verfügung gestellt werden.

- Die realisierenden Technologien treten in den Hintergrund.

Welche Systemkomponenten und Artefakte man für die tatsächliche Realisierung einer dienstorientierten Architektur benötigt, wird nicht immer einheitlich gesehen. In [KB+05] findet sich eine Übersicht auf hohem Abstraktionsniveau, die eine Eingrenzung durchführt.

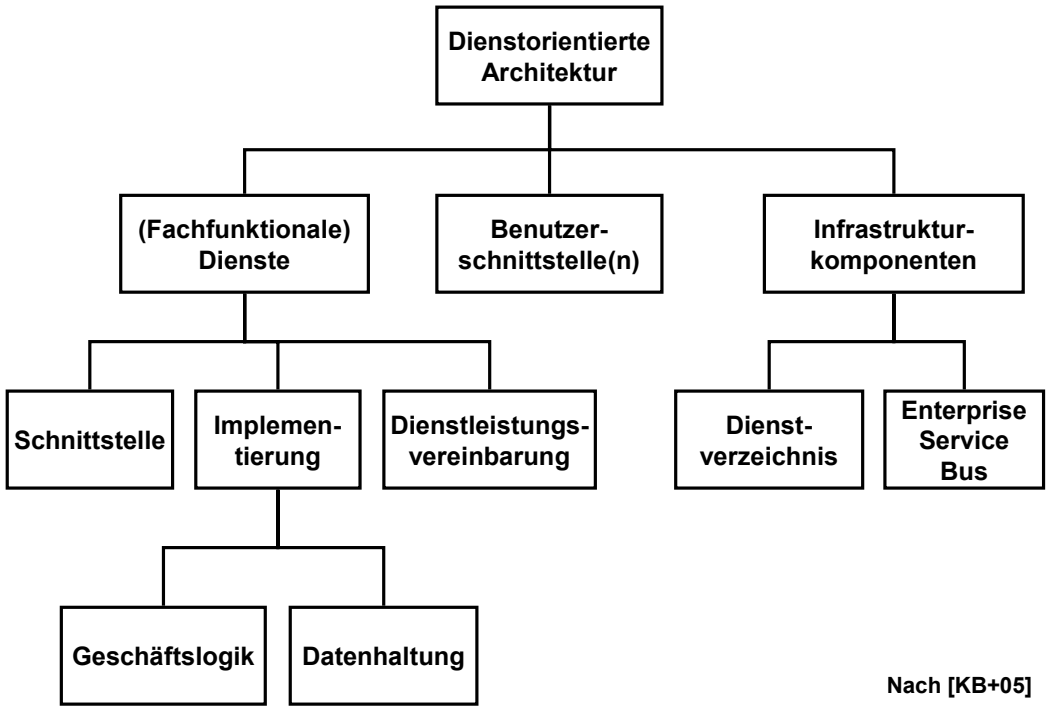


Abbildung 4: Artefakte einer dienstorientierten Architektur

Wie in Abbildung 4 dargestellt, besteht eine dienstorientierte Architektur aus einer Menge fachfunktionaler Dienste, beinhaltet mindestens eine Benutzerschnittstelle als Dienstzugangspunkt für menschliche Benutzer und nutzt eine Anzahl an Infrastrukturkomponenten. Die Dienste sind dabei nach außen definiert über ihre Schnittstelle und über spezifische, mit dem jeweiligen Dienstnehmer auszuhandelnden Dienstleistungsvereinbarungen. Nach innen sind die Dienste charakterisiert über ihre Implementierung, die sich wiederum logisch in Geschäftslogik und Datenhaltungsteil auftrennen lässt. Zur Menge der notwendigen Infrastruktur-

komponenten gibt es in der Literatur noch keinen abschließenden Konsens. Üblicherweise wird jedoch ein Dienstverzeichnis gesehen und ein Enterprise Service Bus (ESB). Der ESB stellt eine virtuelle Kommunikationsinfrastruktur dar, der unterschiedliche Mächtigkeit zugemessen wird: die Herstellung der Konnektivität zwischen den Diensten, die Rolle als Mediator zur Auflösung technischer Heterogenität, aber auch ein „intelligentes Routing", was Möglichkeiten einer Dienstkomposition zur Laufzeit bringt [Ko04].

Wie bereits eingeführt, handelt es sich bei dienstorientierten Architekturen um eine Software-Architektur, die fachlich motivierte, gekapselte Funktionalität an standardisierten (Dienst-)Schnittstellen zur Verfügung gestellt wird. Es stellt sich jedoch die Frage, wie eine solche fachliche Architektur aufgebaut ist und an welcher Stelle Identitätsmanagement und Zugriffskontrolle positioniert werden sollten.

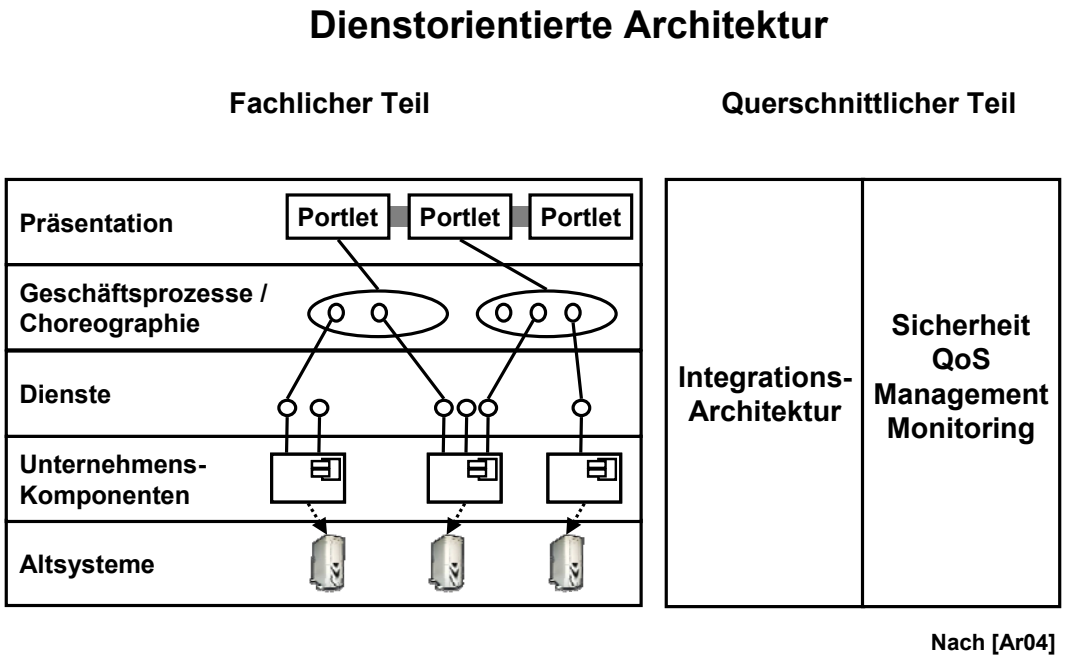


Abbildung 5: Logische Schichtung einer dienstorientierten Architektur

Eine frühe, aber richtungweisende Darstellung der logischen Struktur einer dienstorientierten Architektur wird in [Ar04] gegeben, die in Abbildung 5 dargestellt wird. Sie zeigt in der Grobstruktur eine Aufteilung der Architektur in einen fachlichen und einen querschnittlichen und damit übergreifenden Bereich. Der fachliche Teil besteht aus den Altsystemen (engl. *Legacy Systems* oder *Operational Systems*), die durch Individualität sowohl in Programmiersprachen

als auch ihrem internen Aufbau gekennzeichnet sind. Darauf aufbauend werden unternehmensrelevante Komponenten in den Altsystemen identifiziert, die über sogenannte *Application Server* bereitgestellt werden. Eine gesonderte Dienstschicht stellt die Schnittstelle dieser Funktionalität dar und ermöglicht eine Komposition analog der definierten Geschäftsprozesse. Die Benutzerinteraktion wird über Portale (engl. *Portal, Portlet*) abgewickelt.

Wichtig ist, dass sich dieser Teil nur auf den fachlichen Bereich konzentriert. In dienstorientierten Architekturen sind querschnittliche Aspekte (engl. *Cross-Cutting Concern*) „vor die (fachliche) Klammer zu ziehen". Als Beispiele solcher Querschnittsaufgaben werden die eigentliche Integration, also die Kommunikations- und Interaktionsinfrastruktur genannt, aber auch Bereiche wie Sicherheit, Dienstqualität, Management und Überwachung. Unter dem Schlagwort Sicherheit ist dabei nicht nur die Absicherung der Kommunikation zwischen den verteilten Komponenten zu verstehen, sondern insbesondere auch das Ermöglichen von Zugriffskontrolle auf die fachlichen Komponenten und Dienste. Die Kopplung zwischen fachlichem und querschnittlichem Teil wird jedoch nicht spezifiziert. Ebenfalls bleibt offen, aus welchen Elementen und zugehörigen Prozessen sich der Bereich der Sicherheit zusammensetzt. Damit stellt [Ar04] eine Vorarbeit dar, die die Positionierung und Anbindung der in Kapitel 4 beschriebenen Zugriffskontroll-Architektur begründet.

Auch in der Forschungsgruppe Cooperation & Management (C&M, Prof. Abeck), in der diese Dissertation entstanden ist, wird an der Strukturierung von dienstorientierten Architekturen gearbeitet. Abbildung 6 zeigt die bei C&M entwickelte und durch den Autor mitgestaltete Referenzarchitektur für dienstorientierte Architekturen, die in etwas angepasster Form in [Re05] veröffentlicht und in [EL+06] durch den Autor fortgeschrieben wurde. Neben einer Verfeinerung der Schichtung wird insbesondere die Notwendigkeit querschnittlicher Infrastrukturdienste wie eines Dienstverzeichnisses und eines Identitätsdienstes berücksichtigt. Durch die grafisch nicht explizite Trennung dieser Querschnittsdienste von den fachlichen Diensten soll ihre grundsätzliche Interoperabilität angedeutet werden. Aus technischer Sicht, insbesondere auch auf die nach außen öffentlichen Dienstschnittstellen, gibt es keine Unterschiede zwischen fachlichen und querschnittlichen Diensten. Bereits in dieser Veröffentlichung stellten die Autoren die durch die dienstorientierte Architektur hinzugekommenen, logischen Schichten heraus: Mit den elementaren Webservices (engl. *Core Web Services*) findet der Übergang von klassischen Architekturmustern in die Webservice-

Architektur statt. Aus elementaren Webservices können Webservice-Kompositionen erstellt werden. Aus technischer Sicht sind die Dienstschnittstellen von elementaren und komponierten Webservices nicht zu unterscheiden. Auch müssen alle angebotenen Funktionalitäten nach der Definition in Kapitel 2.1.2 atomar sein. Um eine Verwechslung mit dem Begriff der Atomizität zu vermeiden, werden die nicht komponierten Webservices in dieser Arbeit als elementare Webservices bezeichnet, und nicht, wie ebenfalls in der Literatur zu finden, als atomare Webservices benannt.

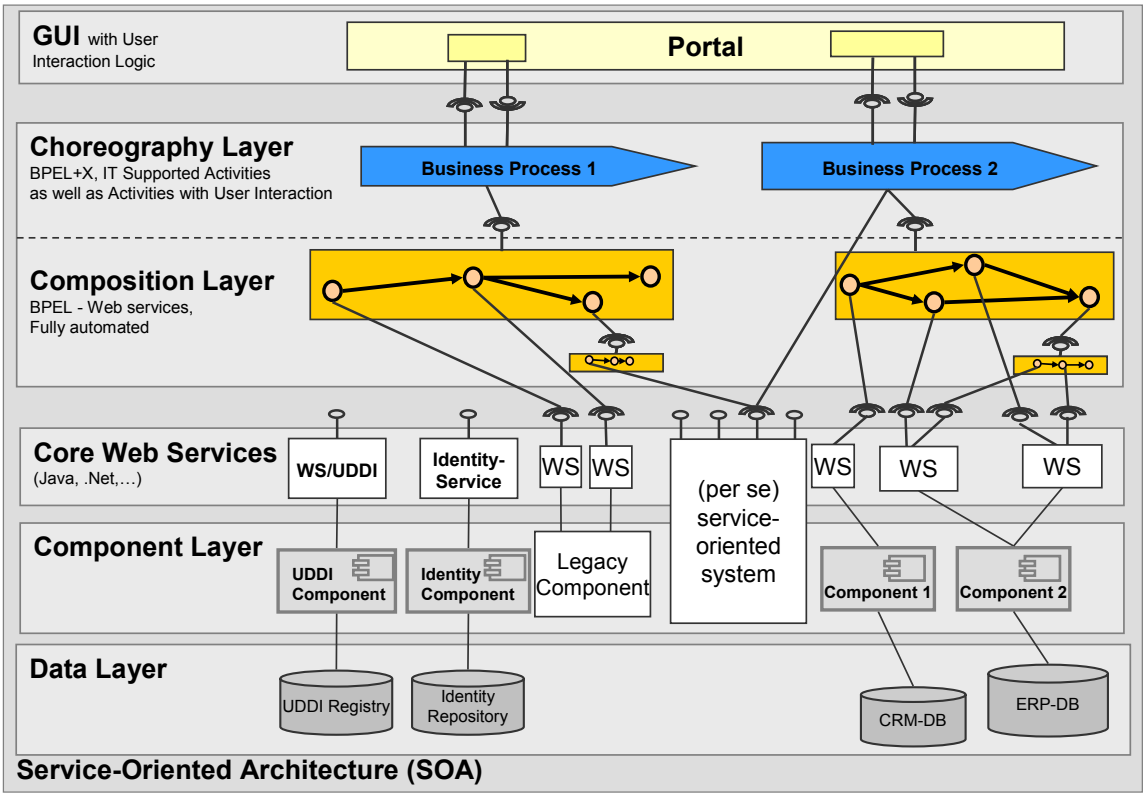


Abbildung 6: Die dienstorientierte Architektur von C&M

Weitere Quellen wie [HH+06] führen mit der Quasar-Architektur für dienstorientierte Architekturen eine weitere Präzisierung im fachfunktionalen Bereich durch und erhärten den Anspruch auf die Trennung von fachlichen Diensten und dem notwendigen Sicherheitsbereich der dienstorientierten Architektur. Es wird explizit definiert, dass die Funktionskomponenten, die die Dienstfunktionalität bereitstellen, keine Benutzerdaten und keinen Sitzungskontext kennen sollen. Dies soll außerhalb der fachlichen Dienste behandelt werden. Diese Trennung stellt, wie bereits gesagt, eine wesentliche Anforderung dar, die auch in Kapitel 3.1 festgeschrieben wird.

2.2 Zugriffskontrolle und Identitätsmanagement

Diese Arbeit thematisiert die Behandlung von Zugriffskontrolle in dienstorientierten Architekturen. Nach der Behandlung der zentralen Begriffe und relevanten Basistechnologien für dienstorientierte Architekturen wird daher nun in den Bereich der Zugriffskontrolle und des Identitätsmanagements eingeführt. Zur Sicherstellung einer effektiven Zugriffskontrolle müssen zwei Anforderungen betrachtet werden: Zunächst muss überprüft werden, ob einem anfragenden Subjekt der Zugriff auf ein geschütztes (System-)Objekt gestattet ist oder nicht. Anschließend muss die tatsächliche Umsetzung dieses Ergebnisses sichergestellt werden [GN07].

2.2.1 Mathematische Grundlagen der Zugriffskontrolle

Mathematische Modelle bilden die formale Grundlage, um den Zugriff auf Ressourcen zu beschränken. Seit Mitte der sechziger Jahre des vergangenen Jahrhunderts wird dieser Themenbereich kontinuierlich bearbeitet. Die ersten formalen Grundlagen finden sich in [La69]. Hier werden sehr abstrakt, aber dennoch formal Berechtigungskonzepte für Betriebsystemumgebungen definiert. Dabei werden die Konzepte „Subjekt“ als anfragendes Element und „Objekt“ als zu schützendes Element einer zugriffsbeschränkten Umgebung eingeführt. Zur Formulierung einer Zugriffsberechtigung wird ein Element *s* aus der Menge der Subjekte *S* mit einem Element *o* aus der Menge der Objekte *O* verknüpft und mit einem elementaren Zugriffsrecht *r* aus der Menge der Zugriffsrechte *R* verbunden. Die Menge aller Zugriffsberechtigungstupel *(s, o, r)* wird Zugriffskontroll-Relation *ZR* genannt und lässt sich in Form einer Zugriffsmatrix darstellen. Diese einfache Form der Zugriffsbeschränkung ohne weitere Stufen der Indirektion wird heute der identitätsbasierten Zugriffskontrolle (engl. *Identity-Based Access Control*, IBAC) zugeordnet, die in Kapitel 2.2.2 dargestellt wird.

Diese einfache Subjekt/Objekt-Relation wird durch das Bell-LaPadula-Sicherheitsmodells [BL73] verfeinert; es gilt als das erste vollständig formalisierte Zugriffskontroll-Modell [Ec06]. Zugriffskontrolle wird hier beschrieben als die Beschränkung des Zugriffs von aktiven Instanzen (Subjekte) auf passive Instanzen (Objekte), wobei eine Instanz je nach Kontext sowohl Subjekt als auch Objekt sein kann. Des Weiteren wird eine Menge an universellen Zugriffsrechten fixiert. Da das Bell-LaPadula-Sicherheitsmodell im militärischen Umfeld entstanden ist, liegt der Fokus des Modells auf der Kontrolle des Informationsflus-

ses. Aufbauend auf den vorgenannten Grundlagen wird dazu jedem Objekt eine Sicherheitsstufe (engl. *Security Classification*) zugewiesen. Jedes Subjekt, einem handelnden Menschen entsprechend, wird ebenfalls einer Sicherheitsstufe (engl. *Security Clearance*) zugeordnet. Unter Berücksichtigung der Sicherheitsstufen wird durch das Modell verhindert, dass ein Subjekt Zugriff auf ein Objekt höherer Sicherheitsstufe erhält. Dies wird unter dem Begriff *No-Read-Up* oder *Simple Security Property* zusammengefasst. Zusätzlich wird zu Geheimhaltungszwecken überprüft, dass ein Subjekt nicht Informationen in (Daten-)Objekte mit niedrigerer Einstufung schreiben kann. Diese Eigenschaft wird *No-Write-Down* oder *-*Property* genannt.

Es gibt einige weitere Modelle, die ähnliche Ziele verfolgen wie das Bell-LaPadula-Sicherheitsmodell. Sie werden unter dem Begriff verbandbasierte Zugriffskontrolle (engl. *Lattice-Based Access Control*, LBAC) zusammengefasst. Stellvertretend sind zu nennen das Biba-Sicherheitsmodell [Bi77], das Clark-Wilson-Modell [CW87] oder das *Chinese Wall*-Modell [BN89]. Während das Biba-Modell eine Umkehrung des Bell-LaPadula-Sicherheitsmodells mit ebenfalls starrem Sicherheitsstufenschema darstellt, betrachtet das Clark-Wilson-Modell die Integrität eines Software-Systems und hat sich fast ausschließlich im Finanzsektor durchgesetzt. Das *Chinese-Wall*-Modell wurde zur Vermeidung von Interessenskonflikten handelnder Akteure entworfen und löst dieses durch die Einführung von Konfliktklassen, die durch die Bildung einer Zugriffshistorie Zugriffe auf ähnliche Objekte in unterschiedlichen Kontexten verhindert. Weitergehende Vertiefung findet sich unter anderem in [Am94] und [Sa93]. Die vorgenannten Modelle werden dem Bereich der systembestimmten Zugriffskontroll-Modelle (engl. *Mandatory Access Control*, MAC) zugeordnet, da Policies systemweit definiert und ohne die Möglichkeit der einzelfallbedingten Abweichung umgesetzt werden.

Trotz ihrer Vorteile haben sich in der Praxis die systembestimmten Zugriffskontroll-Modelle noch nicht nachhaltig durchsetzen können. Stattdessen werden vorwiegend Modelle der benutzerbestimmbaren Zugriffskontrolle (engl. *Discretionary Access Control*, DAC) eingesetzt [SS94]. Der grundsätzliche Unterschied beider Ansätze liegt darin, dass in systembestimmten Modellen die Regeln innerhalb des gesamten Systems gelten, ohne dass Abweichungen möglich wären, während bei benutzerbestimmten Modellen die Regeln individuell objektbezogen definiert werden. Benutzerbestimmte Modelle sind üblicherweise mit dem Eigentümerprinzip verbunden. Das bedeutet, dass der Eigentümer eines

Objekts selbst oder ein beauftragter Administrator die Zugriffsrechte auf das Objekt festlegen können. Damit ergibt sich zwar ein deutlicher Flexibilitätsvorteil zugunsten von Modellen der benutzerbestimmten Zugriffskontrolle, jedoch weisen sie Schwächen bei der Durchsetzung globaler Zugriffsregeln auf, da nur sehr aufwendig überprüft werden kann, ob die Objektbesitzer die Zugriffsrechte tatsächlich im Sinne unternehmensweiter Richtlinien vergeben haben [No05].

2.2.2 Basismodelle zur Zugriffskontrolle

Nachfolgend werden die heute üblicherweise eingesetzten Modelle zur Zugriffskontrolle aufgeführt und ihre Evolution betrachtet.

Identitätsbasierte Zugriffskontrolle (engl. *Identity-Based Access Control*, IBAC) stellt den wohl naheliegendsten Weg zur Darstellung einer Subjekt/Objekt-Relation dar. Subjekte werden charakterisiert über ein eindeutiges Merkmal, auch Identifikator (engl. *Identifier*) genannt. In objektbezogenen Zugriffskontroll-Listen (engl. *Access Control List*, ACL) werden für jeden Identifikator die zugehörigen Zugriffsberechtigungen hinterlegt. Alternativ können Zugriffsrechte auch in subjektbezogen Berechtigungen (engl. *Capabilities*) definiert werden [DH08]. Die Zugriffskontroll-Liste für ein Objekt o_1 stellt damit mathematisch gesehen die Teilmenge der Tupel der Zugriffsrelation ZR dar, für die gilt: $ACL_{o_1} = \{ (s, o, r) \in ZR \mid o = o_1 \}$.

Der Zugriffsschutz eines durch ein IBAC-Modell abgesicherten Systems zu einem gegebenen Zeitpunkt t lässt sich nach [Ec06] durch eine $| S_t | \times | O_t |$ Matrix mit folgenden Eigenschaften modellieren:

- Die Zeilen der Matrix werden durch die Menge S_t der Subjekte und
- die Spalten der Matrix werden durch die Menge O_t der Objekte definiert.
- Es gilt: $M_t : S_t \times O_t \rightarrow 2^R$, wobei R die Menge der Zugriffsrechte festlegt. (2^R bezeichnet die Potenzmenge der Menge R)

Ein Eintrag $M_t (s, o) = \{ r_1, ..., r_n \}$ beschreibt also die Menge der Rechte, die das Subjekt s zum Zeitpunkt t an dem Objekt o besitzt.

Da bei einer anwachsenden Menge von Subjekten und Objekten eine identitätsbasierte Zugriffskontrolle naheliegender Weise nicht skaliert, wurden mit der rollenbasierten Zugriffskontrolle (engl. *Role-Based Access Control*, RBAC) zusätzliche Indirektionsstufen eingeführt. Vereinfacht dargestellt wird das Konzept der Rollen eingeführt, die gleichzeitig mehreren Subjekten zugeordnet werden können. Es sind nun die Rollen und nicht mehr die einzelnen Benutzer selbst, die mit Zugriffsberechtigungen verknüpft werden. Einem (menschlichen) Benutzer wird pro Sitzung eine Teilmenge seiner möglichen Rollen zugeordnet. Diese feste, sitzungsbezogene Rollenzuordnung stellt dann das handelnde Subjekt dar. Systemseitig werden dann ausschließlich auf Basis dieser Rollen die Rechte auf den Objekten vergeben, für die Zugriffskontroll-Entscheidung ist die eigentliche Identität des Benutzers selbst nicht mehr maßgeblich.

Mathematisch kann das RBAC-Basismodell aus [SC+96] durch folgendes Tupel definiert werden [Ec06]:

$$RBAC = (S, O, RL, P, sr, pr, session)$$

Dabei gilt:

- S ist die Menge der Benutzer eines Systems.
- O repräsentiert die Menge der zu schützenden Objekte.
- RL stellt die Menge der Rollen dar.
- P ist die Menge der Zugriffsberechtigungen.
- *sr, pr* und *session* beschreiben Relationen, die nachfolgend spezifiziert werden.

Die Abbildung *sr* modelliert die Rollenzugehörigkeit eines Subjekts $s \in S$:

$$sr : S \rightarrow 2^{RL}$$

Falls $sr(s) = \{ R_1, \ldots, R_n \}$ gilt, dann ist das Subjekt s autorisiert, die Rollen $R_1, \ldots, R_n$ zu aktivieren.

Die Berechtigungen einer Rolle $R \in RL$ werden über die Abbildung *pr* definiert:

$$pr : RL \rightarrow 2^P$$

Die Relation $session \subseteq S \times 2^{RL}$ definiert Paare *(s, rl)*, die als Sitzungen bezeichnet werden, wobei gelten muss: $rl \subseteq sr(s)$. Für ein Subjekt $s \in S$ und für eine Sitzung $(s, rl) \in session$ gilt dann, dass *s* alle Berechtigungen der Rollen $R \in rl$ besitzt.

Bei der Betrachtung der Rollen aus dem RBAC-Basismodell gilt es zu beachten, dass es unterschiedliche Arten von Rollen gibt: Rollen aus der Geschäftsprozessmodellierung und systemspezifische Rollen. Eine unmittelbare Abbildung dieser Rollen aufeinander wäre zwar wünschenswert, ist jedoch nicht trivial.

Mit attributbasierter Zugriffskontrolle (engl. *Attribute-Based Access Control*, ABAC) wird das Ziel verfolgt, Subjekte und Objekte noch unabhängiger voneinander zu betrachten als bei rollenbasierter Zugriffskontrolle. Dazu werden im Kern von ABAC zwei verschiedene Attributtypen definiert. Subjektattribute beschreiben und charakterisieren die Subjekte, und das unabhängig von den möglichen Objekten. Beispiele für Subjektattribute sind der Identifikator, der Name oder die Organisationseinheit. Rollenbezeichnungen aus der Geschäftsprozessmodellierung oder dem RBAC-Modell können auch als Subjektattribute betrachtet werden. Objekte werden ebenfalls durch Attribute, sogenannte Objektattribute beschrieben. Ein Textdokument hat beispielsweise Attribute wie Autor, Titel, Kategorie. Objektattribute können aus Metadaten gewonnen werden, wenn diese bereits explizit vorliegen. Das Kernmodell von ABAC ermöglicht Erweiterungen um weitere Attributstypen wie Attribute, die den Umgebungszustand mit einbeziehen. Diese Erweiterungen verändern die Formalisierung des Modells jedoch nicht grundlegend.

Mathematisch betrachtet lässt sich das Grundmodell der attributbasierten Zugriffskontrolle nach [YT05] wie folgt darstellen:

- *S* und *O* sind die Mengen der Subjekte $(s \in S)$ und Objekte $(o \in O)$.

- $SA_i \ (1 \leq i \leq I)$ und $OA_k \ (1 \leq k \leq K)$ sind vordefinierte Attribute für Subjekte und Objekte.

- *ATTRIBUTE(s)* ist eine Zuordnungsrelation, die einem Subjekt *s* eine Menge an Subjektattributen zuweist:
  $ATTRIBUTE(s) \subseteq SA_1 \times SA_2 \times ... \times SA_I$

- *ATTRIBUTE(o)* ist eine Zuordnungsrelation, die einem Objekt eine Menge an Objektattributen zuweist:
  $ATTRIBUTE(o) \subseteq OA_1 \times OA_2 \times ... \times OA_K$

- Die Entscheidung, ob ein Subjekt auf ein Objekt zugreifen darf, wird auf eine Menge von allgemeinen Policies zurückgeführt, die wiederum auf Subjekt- und Objektattributen basieren. Abstrakt formuliert lassen sich damit boolesche Zugriffsfunktionen nach [YT05] wie folgt darstellen:
  Policy: zugriff_erlaubt (s, o) ← f (ATTRIBUTE(s), ATTRIBUTE(o))

- Die Formulierung von Regeln als flexible Zugriffskontroll-Policies ermöglicht die von Subjekten und Objekten unabhängige Darstellung von Zugriffsrechten. Beispielsweise kann die Policy, dass nur der Eigentümer eines Objekts auf dieses zugreifen darf, objektunabhängig und systemweit wie folgt dargestellt werden:
  Policy P1: zugriff_erlaubt (s, o) ← (Identifikator(s) = Eigentümer(o))

Durch die Auslagerung von Zugriffskontroll-Entscheidungen in nicht unmittelbar mit Subjekten oder Objekten verknüpften Policies wird die Verzahnung der Subjekt-/Objektrelation vermindert. Die Gesamtkomplexität im Bereich der Zugriffskontrolle wird damit nicht verringert, jedoch der Pflegeaufwand der Zugriffsberechtigungen vereinfacht.

2.2.3 Die digitale Identität

Den Ausgangspunkt der Subjekt-/Objekt-Relation stellt das handelnde Subjekt dar. Ein Subjekt im Rahmen der Zugriffskontrolle steht üblicherweise stellvertretend für eine real existierende Person, kann aber auch ein autonom agierendes Software-System darstellen. Das Subjekt wird dabei durch seine digitale Identität repräsentiert. Eine digitale Identität ist die Abbildung von Identitätsmerkmalen der realen Welt auf maschinenverarbeitbare Elemente.

Für den Begriff der digitalen Identität finden sich viele Definitionen. Nachfolgend werden zwei Definitionen vorgestellt, die sich gegenseitig gut ergänzen. In [Wi05] wird folgende Aussage getroffen:

> *"A digital identity contains data that uniquely describes a person or thing [...] but also contains information about the subject's relationships to other entities."*

Der erste Aspekt ist, dass eine digitale Identität charakteristische Daten beinhaltet, die durch ihre Eineindeutigkeit eine Unterscheidung von Identitäten untereinander ermöglichen. Diese Daten werden unter dem Begriff Identifikator (engl. *Identifier*) zusammengefasst. Ähnlich wie im Datenbankumfeld bei der Festlegung des Primärschlüssels wird versucht, die Menge an Daten, die den Identifikator bilden, so gering wie möglich zu halten. Der Identifikator muss lediglich ein eindeutiges Merkmal darstellen innerhalb der Menge an Identitäten, für die er als Identifikator gilt. Ein Beispiel für einen Identifikator kann eine unternehmensweite Personalnummer sein. Die Kombination aus Vorname und Nachname genügt üblicherweise bereits auch in kleinen Gruppen nicht als eindeutiges Merkmal und damit als Identifikator.

In [Bl05] wird die vorgenannte Definition etwas präzisiert:

> *"A digital identity is the representation of a subject that includes a unique number or identifier, credential(s), and attributes (a profile); it may comprise one account or a collection of accounts referring to the same subject."*

Die Daten, die die digitale Identität umfasst, werden nun konkretisiert in einem Identifikator, in Berechtigungsnachweisen (engl. *Credentials*) und sonstigen Attributen, die wiederum in Profilen gruppiert werden können. Ein Berechtigungsnachweis ist ein Mittel, um im digitalen Umfeld die Identität eines Benutzers oder technischen Systems zu bestätigen. Dies geschieht oft durch die Kombination einer Benutzerkennung mit einem Verifikator wie einem Passwort. Zusätzlich wird in dieser Definition dargestellt, dass eine digitale Identität mit mehreren Benutzerkennungen verknüpft sein kann.

Die Attribute einer digitalen Identität können in zwei Bereiche aufgetrennt werden [Ch04]. Kerndaten einer digitalen Identität sind in einer Vielzahl von

Nutzungskontexten relevant. Die Anschrift, Telefonnummern oder E-Mail-Adressen sind Beispiele für solche Kerndaten. Dem gegenüber stehen kontextspezifische Daten. Dabei handelt es sich um Daten, die zwar unmittelbar mit einer digitalen Identität verknüpft sind, jedoch nur für einen speziellen Nutzungskontext Relevanz besitzen. Ein Beispiel für solche kontextspezifischen Daten sind Berechtigungen für ein konkretes Anwendungssystem.

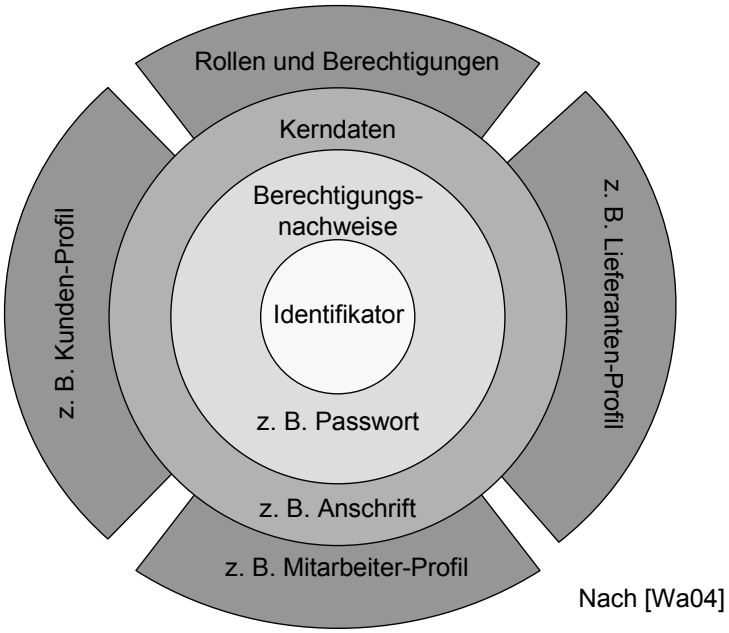


Abbildung 7: Die digitale Identität im Schalenmodell

Oft wird ein Schalenmodell zur Darstellung von digitalen Identitäten verwendet (vgl. Abbildung 7). Es findet sich unter anderem in [Wa04], in einer vereinfachten Version auch in [Ch04]. Den Kern des Schalenmodells einer digitalen Identität bildet der Identifikator. In der ersten Schale finden sich Berechtigungsnachweise, mit denen in der digitalen Welt zweifelsfrei überprüft werden kann, ob es sich tatsächlich um die durch die digitale Identität repräsentierte Entität handelt. Die zweite Schale stellt dann die allgemein relevanten Kerndaten dar, während in der dritten Schale unabhängig voneinander weitere Daten wie anwendungsspezifische Berechtigungen untergebracht sind.

2.2.4 Verzeichnisse und Verzeichnisdienste

Zur Speicherung und Nutzung von digitalen Identitätsdaten werden Verzeichnisse, genauer gesagt Benutzerverzeichnisse, eingesetzt. Ein Verzeichnis lässt sich allgemein beschreiben als eine nach bestimmten Merkmalen gegliederte und

hierarchisch aufgebaute Anordnung von Informationen. Verzeichnisse sind optimiert für Lese- und Suchzugriffe und bieten wesentlich höhere Zugriffsgeschwindigkeiten als es beispielsweise bei Datenbanken der Fall ist. In Verzeichnissen sind die gespeicherten Objekte grundsätzlich unabhängig voneinander. Sie sind nur durch eine strenge Hierarchie, eine Baumstruktur, miteinander verbunden. Dafür gibt es in Verzeichnissen feste Namensschemata und Objektdefinitionen. Konzepte aus dem Datenbankbereich wie Transaktionen werden üblicherweise aufgrund der Atomizität der Zugriffe in Verzeichnissen nicht unterstützt.

Verzeichnisse und Verzeichnisdienste spielen im Kontext von verteilten Systemen wie dem Internet, aber auch in unternehmensinternen Intranets eine wichtige Rolle bei der Bereitstellung von Benutzerdaten. Seit der Spezifikation des *Lightweight Directory Access Protocol* (LDAP) [RFC-2251], das sich am äußerst umfangreichen ITU-T Standard X.500 orientiert, hat sich LDAP zum heute weitverbreiteten Standard für Verzeichnisdienste entwickelt [Zö05]. Die LDAP-Spezifikation selbst definiert dabei nur die Schnittstellen und das Schnittstellenverhalten. Die dahinterliegende Implementierung, insbesondere auch die des konkreten Verzeichnisses als Datenhaltung selbst, wird nicht festgelegt.

LDAP basiert auf dem Dienstgeber-/Dienstnehmer-Modell (engl. *Client/Server Model*), das eine Möglichkeit darstellt, Aufgaben und Dienstleistungen innerhalb eines Netzwerkes zu verteilen. Der Dienstnehmer stellt dabei eine Anfrage an den Dienstgeber und bekommt eine Antwort von diesem. Die LDAP-Spezifikation baut LDAP aus vier logischen Modellen auf [TE+04]:

- Das **Informationsmodell** spezifiziert, wie die einzelnen Einträge (Objekte) in einem LDAP-Verzeichnis aufgebaut sind.

- Das **Namensmodell** definiert den Aufbau der hierarchischen Struktur des Verzeichnisses.

- Das **Funktionsmodell** beschreibt die Operationen, die zum Zugriff auf das Verzeichnis durch den Verzeichnisdienst angeboten werden.

- Das **Sicherheitsmodell** gibt Möglichkeiten, das Verzeichnis als Ganzes oder Teilinformationen daraus vor unberechtigtem Zugriff zu schützen.

Für diese Arbeit wichtig sind insbesondere das Informationsmodell und das Namensmodell, die nachfolgend erläutert werden. Ein LDAP-*Server* ermöglicht den Zugriff auf einen hierarchisch angeordneten Datenspeicher, den Verzeichnisbaum (engl. *Directory Information Tree*, DIT). Der Verzeichnisbaum besteht aus einer Menge von Knoten, wobei die Informationen nicht nur in den Blättern, sondern in allen Knoten abgelegt werden. Jeder Knoten im Verzeichnisbaum entspricht einem Objekt, wobei ein Objekt unterschiedliche Ausprägungen haben kann. Bei einem Objekt kann es sich beispielsweise um eine Person, eine Organisation, eine Organisationseinheit oder eine Hardwarekomponente handeln. Objekte stellen eine Sammlung an sogenannten Attributen, also Schlüssel/Wert-Paaren, dar (engl. *Key/Value Pair*). Die syntaktische Korrektheit der Attribute, aber auch die Kombination mehrerer Attribute zu einem Objekt wird im LDAP-Informationsmodell durch die Definition von Schemata gewährleistet, die ebenfalls hierarchisch aufeinander aufbauen können. Ein Schema definiert seinen Bezug zu anderen Schemata durch Vererbungs- oder Erweiterungsbeziehungen und definiert die Pflichtattribute sowie seine optionalen Attribute eines Objekts.

Jedes Objekt in einem LDAP-Verzeichnis wird eindeutig über seinen *Distinguished Name* (DN) identifiziert. Neben der Eigenschaft als Identifikator beschreibt der DN, wo genau innerhalb der Verzeichnishierarchie sich ein Objekt befindet. Der DN setzt sich aus einer Reihe von Teilen zusammen, den sogenannten *Relative Distinguished Names* (RDN), ähnlich wie ein Pfad im Betriebssystem aus den Verzeichnis- und Unterverzeichnisnamen zusammengesetzt ist. Diese Hierarchie wird Verzeichnisbaum (engl. *Directory Information Tree*, DIT) genannt. Jedes Objekt außer dem Wurzelknoten hat dabei genau einen Vaterknoten, eine Menge an Geschwisterknoten und eine Menge an Kindknoten. Um eine Eindeutigkeit des DN zu gewährleisten, genügt es sicherzustellen, dass der RDN von Geschwisterknoten eindeutig ist. Um eine weltweite Eindeutigkeit zu erreichen, wird das Wurzelobjekt einer Organisation üblicherweise mit ihrem ebenso eindeutigen DNS-Domänennamen versehen.

Das Funktionsmodell von LDAP beschreibt die Operationen, die an der LDAP-Schnittstelle zur Verfügung gestellt werden. Die wichtigste Operation ist die Suche, die das Auffinden von Objekten im Verzeichnis ermöglicht, wobei die Menge an Suchkriterien groß ist. Es besteht die Möglichkeit, Teilbäume für die Suche zu fixieren und individuelle Suchfilter zu definieren. Zusätzlich gibt es Methoden für den Vergleich von Objekten sowie für das Hinzufügen, Ändern und Löschen. Des Weiteren gibt es Operationen für die Authentifizierung von

Benutzern, die auf den im LDAP-*Server* gespeicherten Benutzerinformationen arbeitet. Dies führt direkt in das vierte Modell, das Sicherheitsmodell über. Die LDAP-Spezifikation bestimmt, wie Informationen als Nachrichten zwischen LDAP-*Client* und -*Server* ausgetauscht werden. In diesen Nachrichten werden die Operationen spezifiziert, die der *Client* durch den Verzeichnisdienst im Verzeichnis ausführen möchte (Objekte suchen, bearbeiten, löschen, …). Die LDAP-Spezifikation legt die Datenformate der Nachrichten und das Interaktionsschema fest. Der *Client* baut zunächst eine Verbindung zum *Server*, dem Verzeichnisdienst, auf. Dieser Vorgang wird unter dem Begriff *Binding* zusammengefasst. Durch das Übermitteln eines Benutzernamens und eines Passworts kann sich der *Client* als berechtigter Benutzer authentisieren. Es sind aber grundsätzlich auch anonyme Zugriffe mit definierten Standardberechtigungen möglich. Der *Client* führt im Anschluss die gewünschten Operationen auf dem LDAP-*Server* aus, beispielsweise führt er eine Objektsuche durch oder ändert Objekteinträge. Die wohl häufigste Operation auf einem LDAP-*Server* stellt die Suche dar. Der *Client* kann Daten nach frei wählbaren Kriterien suchen. Er kann zum Beispiel festlegen, in welchem Teil des Verzeichnisbaums gesucht werden soll, und welche Informationen er zurückgeliefert haben will. Es gibt auch Suchfilter für eine Suche über boolesche Bedingungen. Wenn der *Client* fertig ist, schließt er die Sitzung explizit (engl. *Unbinding*).

Um Datenobjekte zwischen LDAP-*Server* auszutauschen, wird das sogenannte *LDAP Data Interchange Format* (LDIF) [RFC-2849] genutzt. Es handelt sich dabei um ein ASCII-basiertes Datenformat zur Darstellung von Informationen aus einem LDAP-Verzeichnis. Da LDAP die interne Darstellung der Objekte in einem LDAP-*Server* nicht spezifiziert, ist diese bei konkreten Implementierungen verschiedener Hersteller unterschiedlich. Um dennoch einen einfachen Austausch von Daten auch zwischen heterogenen Verzeichnissen zu ermöglichen, wurde LDIF als Austauschformat spezifiziert. Wie auch bei LDAP wurde bei der Entwicklung von LDIF darauf geachtet, das Format möglichst einfach zu gestalten. Durch die Beschränkung auf eine rein textuelle Darstellung bleibt das Format menschenlesbar. Ein LDAP-Objekt wird durch mehrere LDIF-Zeilen beschrieben. Am Anfang steht immer der *Distinguished Name*, der die absolute Position in der LDAP-Verzeichnishierarchie angibt. Es folgen die Attribute in Form von Schlüssel/Wert-Paaren, die den eigentlichen Inhalt darstellen. Jede Objekt-Definition wird durch eine Leerzeile abgeschlossen. Um Zeichen außerhalb des 7-Bit-ASCII-Zeichensatzes darstellen zu können, wie es für Binärdateien wie Bilder notwendig ist, werden Verfahren zur Codierung von 8-Bit-

Binärdaten in eine Zeichenfolge, die nur aus wenigen ASCII-Zeichen besteht, durchgeführt. Als Beispiel hierfür ist Base64 [RFC-2045] zu nennen.

2.2.5 Prozesse für Zugriffskontrolle und Identitätsmanagement

Je größer ein Unternehmen ist, desto mehr digitale Identitäten müssen verwaltet werden. Dazu werden sogenannte Zugriffskontroll-Architekturen eingesetzt. Bevor in Kapitel 2.2.6 auf die Elemente einer Zugriffskontroll-Architektur eingegangen wird, werden zunächst die relevanten Prozesse eingeführt.

Die Zugriffskontrolle wird aus mehreren Teilprozessen zusammengesetzt, wobei der Prozessablauf beginnt, sobald ein Zugriffsversuch erkannt wird. Der Prozess der Zugriffskontrolle lässt sich in die Teilschritte Identifizierung, Authentifizierung und Zugriffsentscheidung auftrennen [No05]. Vorgelagert werden bereits durch den Teilprozess der Autorisierung Zugriffsberechtigungen vergeben.

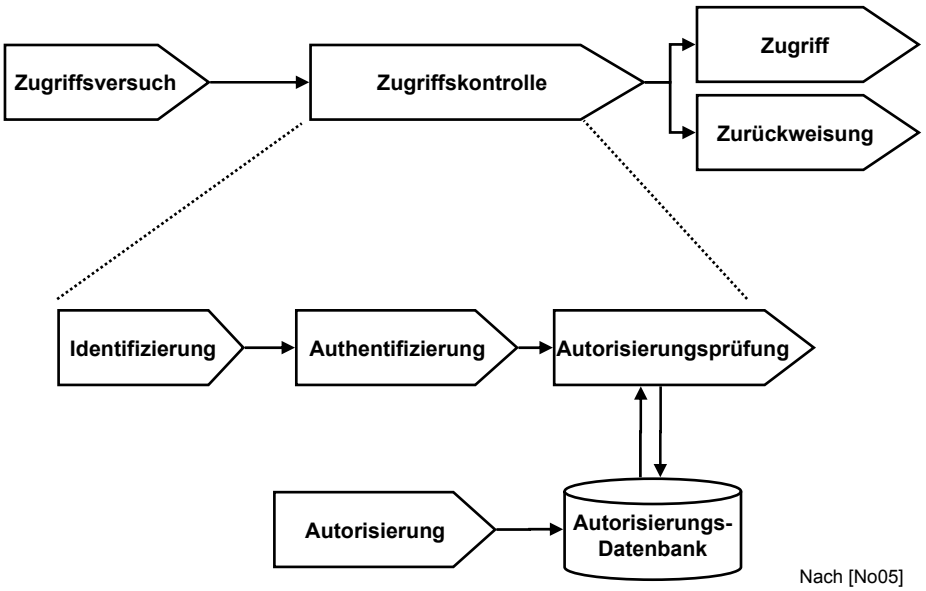


Abbildung 8: Zugriffskontrolle aus Prozesssicht

Beim **Zugriffsversuch** auf ein geschütztes Objekt wird der Zugriff zunächst gestoppt und es findet eine **Zugriffskontrolle** statt. Diese beginnt mit der Identifizierung des aufrufenden Subjekts. Bei der **Identifizierung** handelt es sich um den Vorgang, der zum eindeutigen Erkennen einer Person oder eines Objektes führt. Dabei ist zunächst zu prüfen, ob der Zugreifende dem System überhaupt bekannt ist. Dies erfolgt überwiegend durch eine Suche in einem oder mehreren Benutzerverzeichnissen [JH94].

Bei der **Authentifizierung** wird die Identität einer Person oder eines Software-Programms geprüft. In direktem Zusammenhang dazu steht der Begriff der Authentisierung, der das Nachweisen der Identität aus der umgekehrten Sicht, der des Nutzers meint. Bei der Identitätsüberprüfung lassen sich also zwei beteiligte Rollen feststellen: Der zu Überprüfende authentisiert sich und wird vom Prüfer authentifiziert. Im Englischen spiegelt sich diese unterschiedliche Sichtweise nicht wider, beides wird unter demselben Begriff *Authentication* zusammengefasst. Auch im Deutschen werden die beiden Begriffe Authentifizierung und Authentisierung zwischenzeitig oft bedeutungsgleich verwendet [Ke07]. Die Überprüfung der Identität eines Anfragenden im Rahmen der Authentifizierung wird mit Hilfe von Berechtigungsnachweisen (engl. *Credentials*) durchgeführt. Die Typen der Berechtigungsnachweise lassen sich wie folgt klassifizieren [Di06]:

- Authentifizierung mittels Wissen (engl. *What-You-Know*), typischerweise umgesetzt durch Benutzername/Passwort-Verfahren,

- Authentifizierung durch Besitz (engl. *What-You-Have*), beispielsweise realisierbar durch materialisierte Schlüssel wie Smartcards und

- Authentifizierung durch das Sein (engl. *What-You-Are*), also durch biometrische Merkmale wie Fingerabdrücke.

Nach der erfolgreichen Authentifizierung eines Benutzers kann ihm ein temporärer Authentifizierungsnachweis zur Aufrechterhaltung des Authentifizierungsstatus erteilt werden. Diese Maßnahmen gehören zum Themenbereich des *Single Sign-On*, der nachfolgend dargestellt wird.

Eine gültige Authentifizierung und damit die Feststellung, welche Entität den Zugriff durchführen möchte, bildet die Grundlage für den Übergang in den nächsten Teilprozess, der Durchführung der tatsächlichen Zugriffsentscheidung. Bevor über den Zugriffswunsch entschieden werden kann, müssen in einem vorgelagerten Prozess der **Autorisierung** (engl. *Authorization*) zunächst Zugriffsberechtigungen zugewiesen werden. Die Erstellung, Pflege und finale Löschung dieser Daten, die unmittelbar mit den digitalen Identitäten verknüpft sind, wird dem Identitätsmanagement zugeordnet.

Für die **Autorisierungsprüfung**, also der eigentlichen Zugriffsentscheidung, werden dann die im Rahmen der Autorisierung erstellten Berechtigungsdaten für das authentifizierte Subjekt geladen. Entsprechend des Basismodells zur Zugriffskontrolle wird dann geprüft, ob ein entsprechender Eintrag in der Zugriffskontroll-Relation vorliegt. Falls das der Fall ist, wird der Zugriff gewährt, falls nicht, wird der Zugriff als unzulässig abgewiesen.

Der Prozess der Zugriffskontrolle muss in realen Systemumgebungen ständig durchgeführt werden. Jede Aktion eines Benutzers muss auf ihre Zulässigkeit geprüft werden. Der aufwendigste Teilschritt, bei dem zudem geheimhaltungswürdige Daten zwischen Benutzer und System ausgetauscht werden müssen, ist die Authentifizierung. Um dem Nutzer eine ständige Eingabe von Passwörtern zu ersparen, werden Techniken des *Single Sign-On* eingesetzt. Ziel dabei ist es, dass sich ein (menschlicher) Benutzer innerhalb eines definierten Zeitfensters nur ein einziges Mal authentisieren muss und dann alle Ressourcen benutzen kann, für die er Zugriffsrechte besitzt. Da trotz allem die Authentifizierung aus Systemsicht eine notwendige Vorbedingung für die Zugriffsgewährung darstellt, wird nach der erfolgreichen Authentifizierung eines Benutzers ein temporärer Authentifizierungsnachweis zur Aufrechterhaltung des Authentifizierungsstatus ausgestellt. Dieser wird dem Nutzer übergeben und in der Regel durch technische Maßnahmen transparent für den Benutzer automatisch vorgelegt. Ein Beispiel für einen temporären Authentifizierungsnachweis stellen die sogenannten *Cookies* zur Zustandshaltung im Webumfeld dar. Weitergehende Informationen zu *Single Sign-On*, der Möglichkeit der Kategorisierung und Taxonomie finden sich in [GK+03, PM03, St06b].

Um eine effektive Zugriffskontrolle umsetzen zu können, bedarf es der Pflege der Identitätsdaten einerseits und der Autorisierungsdaten andererseits. Die zugehörigen Prozesse werden dem Identitätsmanagement zugeordnet.

Der Lebenszyklus einer digitalen Identität beginnt mit der initialen Erzeugung, bei der neben ihrem Kern auch die entsprechenden Berechtigungen zugewiesen werden. Der Begriff der Provisionierung fasst im Identitätsmanagement die Verbindung von digitalen Identitäten mit den Zugriffsrechten für die Anwendungsfunktionalität, die sie im Rahmen ihrer Geschäftstätigkeit verwenden müssen [Wa04]. Die zugewiesenen Berechtigungen sollten sich möglichst automatisiert innerhalb des Unternehmens auf die relevanten Software-Systeme verteilen lassen. Hierzu werden sogenannte Provisionierungssysteme eingesetzt,

die über Adaptoren mit den Anwendungssystemen verbunden sind. Der Provisionierung wird auch die semiautomatische Versorgung von Systemen zugerechnet, für die es nicht möglich ist oder sich nicht lohnt, spezifische Adaptoren zu entwickeln. Das Denken in Geschäftsprozessen und damit über Systemgrenzen hinweg verlangt eine einheitliche Infrastruktur. Isoliert pro Anwendung definierte Identitäten können die Provisionierung erschweren.

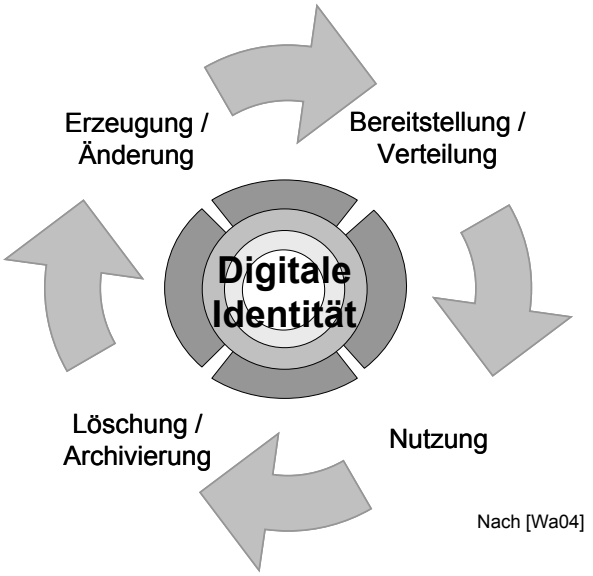


Abbildung 9: Lebenszyklus digitaler Identitäten

Analog zu den Änderungen im betrieblichen Umfeld wie dem Arbeitsplatzwechsel, dem Übernehmen neuer Aufgabenbereiche oder der Durchführung von Reorganisationsmaßnahmen müssen die betroffenen digitalen Identitäten auch auf technischer Ebene nachgezogen werden. Das führt zur Anpassung von Berechtigungen, die mit digitalen Identitäten verknüpft sind. Diese Änderungen müssen wiederum auf alle relevanten Systeme verteilt werden. Auch wenn Änderungen an einer digitalen Identität von einem System ausgelöst werden, müssen zur Konsistenzhaltung alle relevanten Systeme aktualisiert werden. Der Prozess der Verteilung von Änderungsinformationen wird im Gegensatz zur Neuanlage von Identitäten der Synchronisation zugeordnet.

Am Ende des Lebenszyklus einer digitalen Identität steht die Deprovisionierung, das heißt ihre Löschung. Wenn aus rechtlichen oder sonstigen Gründen eine Löschung nicht zulässig oder möglich ist, kann auch eine Inaktivierung an diese Stelle treten. Die Deprovisionierung gehört zu den am häufigsten unterschätzten Prozessen. Eine nicht oder fehlerhaft durchgeführter Provisionierung führt zu

fehlenden Rechten und damit fehlenden Möglichkeiten auf Seite des Benutzers, der sich in aller Regel bei der zuständigen Administration meldet, um den Fehler beheben zu lassen. Bei der Deprovisionierung werden dagegen Rechte eingezogen analog einer Schlüsselabgabe. Die Interessenlage ist dabei umgekehrt: Der Eigentümer einer zugriffsbeschränkten Ressource sollte in diesem Fall das Interesse haben, ausgeschiedenen Benutzern diese Rechte wieder zu entziehen, da sich die Benutzer üblicherweise nicht eigeninitiativ melden. Früher wurde dies Deprovisionierung vor allem unter dem Blickwinkel der (System-)Sicherheit gesehen. Heute kommen die Aspekte Wirtschaftlichkeit und Gesetzeseinhaltung (engl. *Compliance*) hinzu. Beispiele für Wirtschaftlichkeit sind Lizenzmodelle auf Nutzerbasis. Unternehmenskosten können reduziert werden, wenn nicht mehr benötigte digitale Identitäten und die damit verbundenen Benutzerkonten gelöscht werden. Auch Normen wie das deutsche Gesetz zur Kontrolle und Transparenz im Unternehmensbereich (KonTraG) oder der amerikanische *Sarbanes-Oxley Act* zwingen Unternehmen heute dazu, Prozesse im Bereich des Identitätsmanagements explizit zu formulieren und deren Einhaltung zu kontrollieren. Neben der laufenden Prozessüberwachung können dazu stichprobenartig Auditierungen, also Soll/Ist-Vergleiche durchgeführt werden.

2.2.6 Architekturelemente für Zugriffskontrolle und Identitätsmanagement

Um Identitätsmanagement und Zugriffskontrolle zu ermöglichen, wird der fachfunktionale Teil einer Software-Architektur um zusätzliche Komponenten erweitert. Eine domänenunabhängige Referenzarchitektur für den Teilbereich der Zugriffskontrolle wird in [ISO89] vorgestellt und in [ITU-X.812] verfeinert. Eine zentrale Aussage in beiden etablierten Standards ist die Trennung zwischen fachfunktionalen Komponenten und Sicherheitsfunktionalität (engl. *Separation of Concerns*). Zusätzlich wird der Kernbereich der Zugriffskontrolle auf zwei getrennte Komponenten aufgeteilt: die Zugriffskontroll-Entscheidungsfunktion (engl. *Access Control Decision Function*, ACDF) und die Zugriffskontroll-Durchsetzungsfunktion (engl. *Access Control Enforcement Function*, ACEF). Dabei wird die Zugriffskontroll-Durchsetzungsfunktion der eigentlichen Fachfunktionalität vorgeschaltet mit dem Ziel, unberechtigte Zugriffe zu verhindern. Die tatsächliche Entscheidung, ob eine Zugriffsberechtigung vorliegt oder nicht, wird jedoch von der Zugriffskontroll-Durchsetzungsfunktion an die oft örtlich getrennte Zugriffskontroll-Entscheidungsfunktion ausgelagert. Diese prüft die

Anfrage und sendet eine Freigabe oder Zurückweisung an die Zugriffskontroll-Durchsetzungsfunktion zurück.

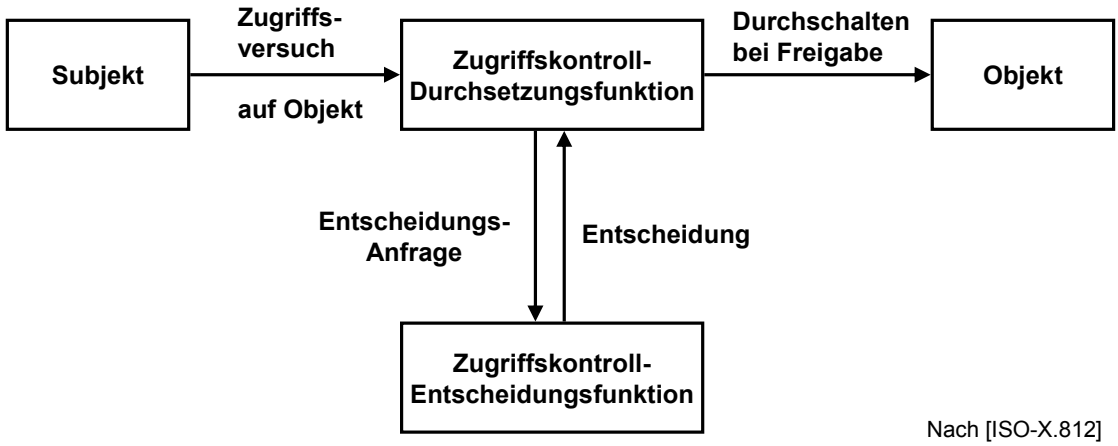


Abbildung 10: Basiselemente für Zugriffskontrolle

Beide Standards definieren weder die Anzahl noch die exakte Positionierung der beiden Funktionseinheiten in einer Software-Architektur. Jedes in sich geschlossene Anwendungssystem nutzt innerhalb der Systemgrenzen integrierte Zugriffskontroll-Durchsetzungs- und Zugriffskontroll-Entscheidungspunkte. Durch die enge Bindung innerhalb fester Systemgrenzen besteht eine Vertrauensbeziehung, die dem fachfunktionalen Objekt die Auslagerung seiner Zugriffskontrolle vereinfacht. In verteilten Systemen, insbesondere auch in dienstorientierten Architekturen, ist es jedoch ein Ziel, diese Zugriffskontroll-Funktionalität übergreifend anzubieten. Normalerweise befinden sich beide Komponenten in verteilten Systemen im Vertrauensbereich des Objekts, um keine unmittelbare Vertrauensbeziehung gegenüber dem zugreifenden Subjekt voraussetzen zu müssen. Das Vertrauen des Anfragenden in die größtmögliche Geheimhaltung seiner personenbezogenen Daten, die zur Durchführung der Zugriffskontrolle den verschiedenen Komponenten offengelegt werden muss, ist ein aktuelles Forschungsthema [DR06].

Die beiden vorgenannten Kernkomponenten setzen weitere Komponenten, Datenquellen und Betriebsprozesse voraus, um ihre Aufgaben durchführen zu können. Als Beispiele für Datenquellen sind dabei zu nennen die Autorisierungsdatenbank, in der die Zugriffsberechtigungen hinterlegt sind, oder die Benutzerverzeichnisse, in denen die Daten über die möglichen Subjekte hinterlegt sind. Benutzerverzeichnisse selbst können auch wiederum isoliert betrachtet werden oder über Konzepte wie Meta-Verzeichnisse oder virtuelle Verzeichnisse zu-

sammengefasst werden [Le98]. Zu den Betriebsprozessen gehört die Pflege der Autorisierungsdaten. Zur Verteilung und Konsistenthaltung der Autorisierungsdaten werden Provisionierungskomponenten und Synchronisationseinrichtungen eingesetzt.

Die Begriffsbildung im Englischen hat sich in den letzten zwanzig Jahren verändert. In aktueller Literatur [SN+05, Di06] und Standards [RFC-2753] werden zwischenzeitlich die Begriffe *Policy Enforcement Point* (PEP) anstelle von *Access Control Enforcement Function* (ACEF) und *Policy Decision Point* (PDP) anstatt *Access Control Decision Function* (ACDF) benutzt. Auch in der vorliegenden Arbeit werden die aktuellen Begriffe PEP und PDP verwendet.

Der Policy-Begriff tritt in den verschiedensten Domänen auf. In der Informatik sind Policies als Statuten, Richtlinien oder Verhaltensregeln für die Benutzung von Systemen, Netzwerken oder Diensten bekannt. Im Gegensatz zum Programmcode werden sie nicht kompiliert, sondern zur Laufzeit ausgewertet, das heißt interpretiert.

In [LS99] findet sich eine gute Definition des Begriffs „Policy“:

> *„A policy is information which can be used to modify the behavior of a system. […] Separating policies from the systems which interpret them permits the modification of the policies to change the behavior of the system without recoding it. The system can then adapt to changing requirements by disabling policies or replacing old policies with new ones without shutting down the system.“*

Policies können als Konfigurationsdaten verstanden werden, die nicht hart im fachfunktionalen Programmcode fixiert wurden. Dies erlaubt die Anpassung eines Programms oder Systems an neue Randbedingungen, ohne es neu übersetzen zu müssen. Gerade für den hier betrachteten Bereich der Zugriffskontrolle ermöglichen Policies als änderbare Konfigurationsdaten die Trennung zwischen Fachfunktionalität und zugehörigen Zugriffsberechtigungen. Letztere können beliebig zur Laufzeit angepasst werden, was sich gerade für Dienstwiederverwendung in dienstorientierte Architekturen vereinfachend auswirkt.

Policies werden nicht nur in sehr unterschiedlichen Domänen eingesetzt, sie werden auch auf unterschiedlichen Abstraktionsebenen ausgeprägt. Nach [Wi95]

ergibt sich eine Policy-Hierarchie mit vier Ebenen. Die Hierarchie beginnt oben mit unternehmensweiten Policies (engl. *Corporate Policies*), die unmittelbar aus Geschäftszielen abgeleitet werden. Sie sind abstrakt gehalten, geschäftsorientiert und treffen nur in geringem Umfang technologische Festlegungen. Diese allgemeinen Policies lassen sich auf Geschäftsprozesse anwenden. Hierbei entstehen individuelle Geschäftsprozess-bezogenen Policies (engl. *Business Process-Oriented Policies*). Der Grad an technologischen Festlegungen nimmt zu, ebenso wie der Detaillierungsgrad der Policies. Die nächst tiefere Ausprägung stellen die funktionalen Policies (engl. *Functional Policies*) dar, die die zur Rechnerunterstützung eingesetzten Systeme konfigurieren. Die unterste Schicht an Policies adressieren direkt die atomarsten, zu verwaltenden Objekte (engl. *Managed Objects*). Sie sind ausschließlich technologisch geprägt und hinreichend detailliert, um unmittelbar zur Anwendung in Laufzeitumgebungen gebracht zu werden.

2.3 Modellierung und modellgetriebene Entwicklung

Diese Arbeit adressiert Zugriffskontrolle in dienstorientierten Architekturen. Nach der Einführung von dienstorientierten Architekturen und der Behandlung der relevanten Grundlagen in den Bereichen Identitätsmanagement und Zugriffskontrolle wird in diesem Kapitel der Grundstein zur formalen Darstellung der entwickelten Informatik-Artefakte gelegt; Modellierung spielt dabei eine zentrale Rolle. In der Informatik werden Modelle eingesetzt, um Komplexität beherrschbar zu machen. Ein Modell stellt dabei ein Abbild eines realen oder abstrakten Objekts dar, das Analogien aufweist und durch einen oder mehrere Aspekte eine Abstraktionsbeziehung zum Ausgangsobjekt besitzt. Modelle verfolgen das Ziel, durch Weglassen oder Vernachlässigen von Eigenschaften die Komplexität des Ausgangsobjekts zu reduzieren [PM06].

In [GI-IB] wird ein Modell definiert als eine

> *„idealisierte, vereinfachte, in gewisser Hinsicht ähnliche Darstellung eines Gegenstands, Systems oder sonstigen Weltausschnitts mit dem Ziel, daran bestimmte Eigenschaften des Vorbilds besser studieren zu können“.*

Ein Modell kann jedoch mehr als ein reines Anschauungsobjekt sein. Je höher sein Formalisierungsgrad ist, desto besser kann es im Rahmen von modellgetrie-

bener Software-Entwicklung eingesetzt werden, die in Abschnitt 2.3.4 beschrieben wird.

Ein Modell weist nach [St73] und [RH06] drei Eigenschaften auf:

- **Abbildungstreue und Richtigkeit**. Die Beziehung zwischen einem Modell und dem durch das Modell dargestellten Objekt wird durch eine (mathematische) Abbildung bestimmt. Das Objekt muss dabei nicht real existieren. Je nach Perspektive (engl. *Viewpoint*) kann ein Objekt unterschiedlich modelliert werden. Durch Modelle lassen sich sowohl statische Strukturen als auch dynamische Verhaltensweisen darstellen.

- **Abstraktion und Relevanz**. Ein Modell stellt eine Repräsentation eines Objekts dar, wobei es sich auf bestimmte Aspekte beschränkt, die für den Zweck des Modells relevant sind. Das Weglassen von für die weitere Betrachtung irrelevanten Aspekten wird Abstraktion genannt. Auch Modelle selbst können durch die Reduktion weiterer Aspekte weiter abstrahiert werden, ihr Abstraktionsgrad wird dadurch erhöht.

- **Pragmatik**. Ein Modell wird für einen bestimmten Zweck angefertigt. Der Zweck bestimmt sowohl den Bezug zwischen Objekt und Modell, also die Durchführung der Abbildung als auch die Komplexitätsverschattung, also die Abstraktion.

Durch die Fokussierung auf dienstorientierte Architekturen ist für diese Arbeit insbesondere ein Verständnis über Geschäftsprozess- und Systemmodelle wichtig. Geschäftsprozesse definieren, wie in einem Unternehmen gearbeitet wird, und bilden die Brücke zu den Systemmodellen, die die Informatiksysteme beschreiben, die für die zugehörige Rechnerunterstützung eingesetzt werden. Nach der Darstellung dieser beiden Modellierungsbereiche folgt eine Einführung in die Metamodellierung, die die Basis für modellgetriebene Software-Entwicklung darstellt.

2.3.1 Geschäftsprozessmodellierung

In DIN 66201 wird der Begriff „Prozess“ beschrieben als Umformung und/oder Transport von Materie, Information oder Energie. In der Informatik wird der Prozessbegriff formal definiert als eine Abfolge von Aktionen in einem Zu-

standsraum [Ba00]. Auf dieser allgemein gehaltenen Aussage kann die Definition eines Geschäftsprozesses aufgebaut werden. Bei einem Geschäftsprozess handelt es sich um einen speziellen Prozess, der die wiederkehrende Abfolge unternehmensspezifischer Aktivitäten betrachtet, die durchgeführt werden, um ein geschäftsrelevantes Resultat zu erzielen. In [OW+03] wird der Begriff des Geschäftsprozesses zusammenfassend definiert als ein

> *„geschäftliche[r] Ablauf, der von einem geschäftlichen Ereignis ausgelöst wird, und der mit einem Ergebnis endet, das für den Unternehmenszweck und damit die Gewinnerzielungsabsicht einen mittelbaren oder unmittelbaren geschäftlichen Wert darstellt“.*

Die gewünschte Wiederholbarkeit der Durchführung eines Geschäftsprozesses grenzt ihn zu einem einmalig durchzuführenden Projekt ab. Geschäftsprozesse sind zunächst unabhängig von ihrer möglichen Rechnerunterstützung zu betrachten. Es sind lediglich die am Prozess beteiligten Rollen (Menschen und/oder Software-Systeme), ihre Arbeitsabläufe und ihre Beziehungen und Interaktionen untereinander von Bedeutung. Ausgehend von einem formalen Geschäftsprozessmodell kann dann strukturiert auf eine passende Rechnerunterstützung geschlossen werden.

Ein Geschäftsprozessmodell ist eine abstrakte Abbildung eines tatsächlichen oder beabsichtigten Geschäftsprozesses, der von Personen und/oder Software-Systemen ausgeführt wird. Es gibt unterschiedliche Notationen, um Geschäftsprozesse zu modellieren. Zu den relevantesten gehören nach [Da07] die Modelle der *Unified Modeling Language* (UML) [Sp00], die *Business Process Modeling Notation* (BPMN), erweiterte ereignisgesteuerte Prozessketten (eEPK) sowie Petri-Netze. Durch definierte Transformationen sind die Diagramme der BPMN in direkt ausführbare Prozessprogramme umsetzbar, wie der Autor in [EM+05] gezeigt hat. Im BPMN-Standard sind dafür Transformationsregeln auf die im Webservice eingesetzte *Business Process Execution Language* (BPEL) als Kompositionssprache definiert. Eine Einführung zur BPMN findet sich in [Wh04] und [OR03], der vollständige Standard, in dem auch die Transformationsregeln zu ausführbarem BPEL-Programmcode spezifiziert sind, findet sich in [OMG-BPMN]. Die UML als Standardsprache im Bereich der Software-Entwicklung stellt eine wichtige Grundlage für diese Arbeit dar und wird im nachfolgenden Kapitel eingeführt.

2.3.2 Die Unified Modeling Language (UML)

Bei der *Unified Modeling Language* (UML) handelt es sich um eine durch die *Object Management Group* (OMG) standardisierte Sprache. Unter anderem lässt sie sich zur Konstruktion und Visualisierung von Modellen für Software-Systeme und Geschäftsprozesse einsetzen [BH+04, ER02]. Durch die Standardisierung und ihre breite Akzeptanz ermöglicht sie, den Entwurf und die Entwicklung von Modellen im Informatikkontext auf Basis einer einheitlichen Sprache zu diskutieren. Die UML stellt dabei keine Methode im Sinne einer systematischen Abarbeitung einzelner Schritte eines Software-Entwicklungsprozesses dar [St05]. In der Software-Technik wird unter dem Begriff „Methode“ die Kombination einer Notation mit Konzepten und einer methodischen Vorgehensweise verstanden [Ba00]. Die UML spezifiziert bewusst nur die Notation zur formalen Beschreibung eines Modells, sie ist weitestgehend methodenneutral. Detaillierte Vorgehensweisen zum Einsatz der UML in einem Software-Entwicklungsprozess werden außerhalb des UML-Standards behandelt. Ein Beispiel für einen auf der UML aufsetzenden Software-Entwicklungsprozess ist der *Rational Unified Process* (RUP) [JB+99], der ein iteratives Vorgehen der Firma Rational Software (zwischenzeitlich: IBM) definiert. Um durch ein gemeinsames Verständnis von UML-Diagrammen eine Qualitätsverbesserung in der Software-Entwicklung zu erreichen, ist es wichtig, sie richtig zu nutzen. Grundsätzlich können mehrere Diagramme eingesetzt werden, um ein und dasselbe Objekt zu modellieren. Dabei können Diagramme aufeinander aufsetzen, wie Anwendungsfalldiagramme und Aktivitätsdiagramme im Bereich der Geschäftsprozessmodellierung. Es gibt auch die Möglichkeit der Überlappung von Diagrammen. Die dadurch entstehende Redundanz darf jedoch nicht zu Inkonsistenzen führen.

Die UML ist heute eine der dominierenden Sprachen für die Modellierung von betrieblichen Software-Systemen. Die erste Version der UML entstand in den neunziger Jahren des vergangenen Jahrhunderts als Zusammenführung von verschiedenen Modellierungsansätzen, welche die zu dieser Zeit aufkommende objektorientierte Software-Entwicklung unterstützen sollten. Die UML-Version 1.0 wurde 1997 von der OMG als Standard angenommen und seither kontinuierlich fortgeschrieben. Die Versionsreihe 1.x wurde 2005 durch eine grundlegend überarbeitete Version abgelöst, die unter dem Begriff UML2 zusammengefasst wird. Die UML2 setzt sich aus mehreren Teilstandards zusammen, wobei der Kern gebildet wird durch die UML-*Infrastructure* [OMG-UML2-Infra], die UML-*Superstructure* [OMG-UML2-Super] und die *Object Constraint Language*

(OCL) [OMG-OCL2]. Die UML-*Infrastructure*-Spezifikation bildet die Basis für die UML2, indem sie die Kernelemente von UML2 einführt sowie Möglichkeiten zur Spracherweiterung gibt. In der UML-*Superstructure* findet dann eine Erweiterung der UML-*Infrastructure* um konkrete Diagramme statt. Die Spezifikation ist in mehrere Blöcke gegliedert, wobei in jedem Block eine Menge von zusammenhängenden Modellierungselementen eingeführt wird, mit denen ein Entwickler einen ausgewählten Aspekt eines Systems darstellen kann. Der Einsatz der *Object Constraint Language* soll es ermöglichen, die bestehenden Modelle weiter zu präzisieren. Während durch die UML Strukturen, Abläufe und Beziehungen zwischen Objekten modelliert werden, werden in OCL zusätzliche Randbedingungen und Zusicherungen eines Modells spezifiziert wie beispielsweise Invarianten oder Vor-/Nachbedingungen. OCL-Ausdrücke werden deklarativ spezifiziert und stellen widerspruchsfreie Aussagen dar. Sie können automatisiert ausgewertet werden, wobei sie nicht das eigentliche Modell verändern, sondern eher seine Korrektheit sicherstellen.

Die dreizehn Diagrammtypen der UML lassen sich in zwei große Bereiche aufteilen: die Strukturdiagramme und die Verhaltensdiagramme. Strukturdiagramme drücken statische Beziehungen aus, während Verhaltensdiagramme zur Modellierung dynamischer Aspekte verwendet werden.

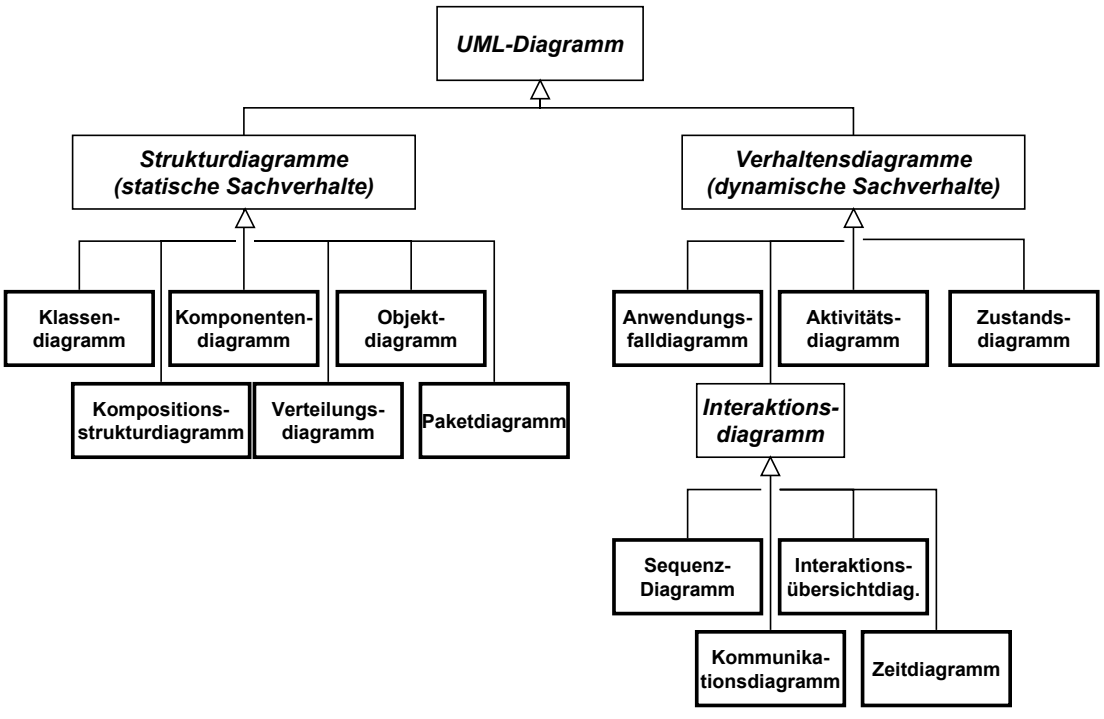


Abbildung 11: Diagrammtypen der UML

Die dreizehn Diagrammtypen der UML2 werden in Abbildung 11 dargestellt und den vorgenannten Hauptaspekten „Struktur“ beziehungsweise „Verhalten“ zugeordnet. Strukturdiagramme stellen statische Sachverhalte dar, beispielsweise die Struktur eines Software-Systems zu einem bestimmten Zeitpunkt. Ein Systemzustand wird dabei fixiert, also eingefroren. Oft sind diese Aspekte sogar unabhängig von einem konkreten Zeitpunkt und stellen dauerhafte Beziehungen zwischen Modellelementen dar. Verhaltensdiagramme werden zur Modellierung von dynamischen Aspekten verwendet. Im Gegensatz zu (statischen) Strukturdiagrammen wird hier nicht ein Zeitpunkt, sondern ein Zeitraum betrachtet. Dabei spielen das Handeln und die Aktionen einzelner Objekte, insbesondere aber auch die Interaktion von Objekten miteinander eine Rolle. In der Gruppe der Verhaltensdiagramme nehmen Interaktionsdiagramme eine Sonderstellung ein. Es gilt jedoch festzuhalten, dass die Grenzen zwischen den Diagrammtypen weniger scharf verlaufen, als diese strikte Aufteilung es vermuten lässt. Die UML2-Spezifikationen verbieten es nicht, dass ein Diagramm grafische Elemente enthält, die eigentlich zu unterschiedlichen Diagrammtypen gehören. Es ist sogar denkbar, dass Elemente aus einem Strukturdiagramm und aus einem Verhaltensdiagramm auf dem gleichen Diagramm dargestellt werden, wenn damit eine besonders treffende Aussage zu einem Modell erreicht wird.

Für eine detaillierte Darstellung der einzelnen Diagrammtypen sei auf die UML-Webseite (http://www.uml.org) oder eines der vielen UML-Einführungsbücher wie [St05, BH+04, Oe04, ER02] verwiesen.

2.3.3 Metamodellierung

Der Aufbau der UML nutzt eine Metamodellierungshierarchie, die in der OMG-Spezifikation *Meta Object Facility* (MOF) [OMG-MOF] beschrieben wird. Der bereits in Abschnitt 2.3 eingeführte Begriff des Modells wird dabei um den des Metamodells ergänzt.

Im Handbuch der Software-Architektur [RH06] wird ein Metamodells definiert als

> *„ein Modell, das eine Menge anderer Modelle definiert, die als Instanzen des Metamodells bezeichnet werden. […] Metamodelle sind ebenfalls Modelle, da sie selbst eine verkürzte oder partielle Sicht der Definition einer Modellierungssprache verkörpern“.*

Etwas ausführlicher wird die Meta-Eigenschaft eines Modells in [PM06] dargestellt:

> *„Ein Modell ist – bezogen auf die Metaebene eines anderen Modells – dann ein Metamodell, wenn es in einer Abstraktionsbeziehung im Sinne eines höheres Abstraktionsgrades zu diesem steht, Aussagen über das andere Modell (und nicht den Gegenstandsbereich des Modells) macht und die Modellelemente des anderen Modells als Instanzen der Metamodellelemente des Metamodells verstanden werden können. Das Metamodell befindet sich auf der höheren (Meta-)Ebene als das Modell. Modelle und Metamodelle können dieselbe konkrete Syntax verwenden, besitzen jedoch eine unterschiedliche abstrakte Syntax beziehungsweise Semantik."*

Metamodellierung bedeutet vereinfacht gesagt, dass jedes konkrete Objekt eine Instanz eines Modells ist. Ein Modell seinerseits stellt die Instanz eines Metamodells dar. Eine Ebene höher sind die Metamodelle dann Instanzen von Meta-Metamodellen. Diese Hierarchiebildung wird auf der Ebene der Meta-Metamodelle abgeschlossen, da diese als Instanzen ihrer selbst definiert sind [St05].

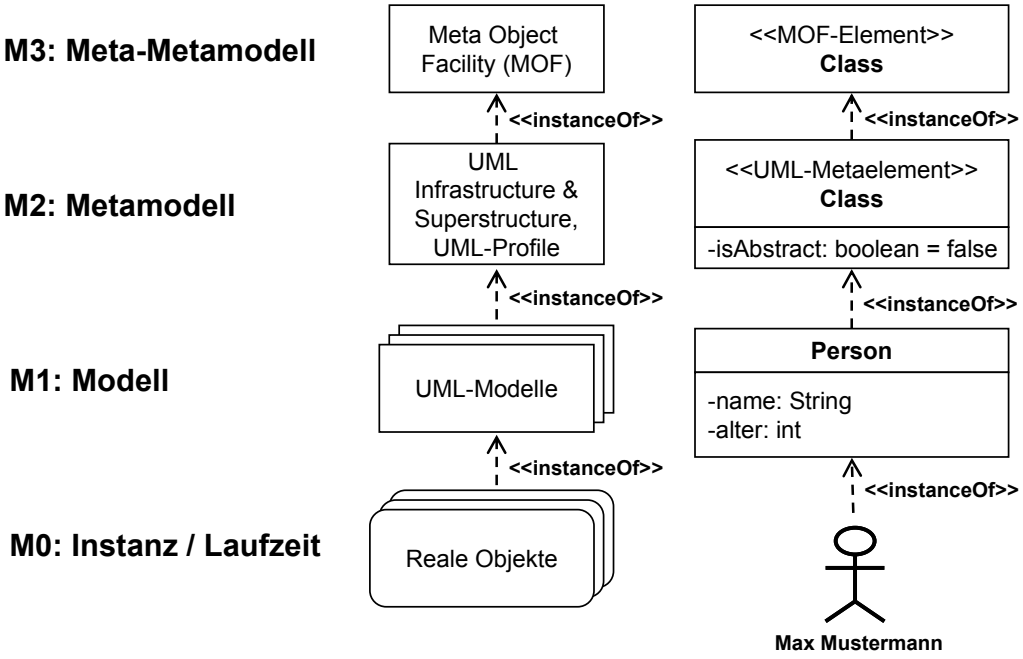


Abbildung 12: Die Metamodellierungshierarchie der UML

Eine Darstellung der spezifischen UML-Metamodellierungshierarchie wird im linken Bereich von Abbildung 12 aufgezeigt. Der Schicht M0 werden dabei die real existierenden Objekte zugeordnet. Beispiele können konkrete Objekte wie Datensätze in einer Datenbank oder ein Programmzustand zur Laufzeit sein. Die Modellschicht M1 umfasst die bekannten UML-Modelle aus Abschnitt 2.3.2, die zur Beschreibung der konkreten Objekte auf der M0-Ebene eingesetzt werden. Die Metamodellschicht M2 definiert die (abstrakte) Syntax und Semantik der Modelle auf der M1-Schicht. Den Kern der UML-Metamodellschicht M2 bildet die UML-*Infrastructure*, die durch die UML-*Superstructure* ergänzt wird. Darüber hinaus gibt es die Möglichkeit, durch sogenannte UML-Profile weitere, domänenspezifische Erweiterungen vorzunehmen. Die Hierarchie der UML-Metamodellierunghierarchie wird mit der *Meta Object Facility* auf der Meta-Metamodellebene M3 abgeschlossen, die als Instanz ihrer selbst definiert ist.

Zur Erhöhung des Formalisierungsgrades von Modellen wurde mit der *Object Constraint Language* (OCL) [OMG-OCL2] eine Möglichkeit geschaffen, zusätzliche Restriktionen, Randbedingungen oder Zusicherungen wie Invarianten oder Vor-/Nachbedingungen zu spezifizieren. Seit der UML2 können mittels OCL beliebige Ausdrücke über alle Elemente eines UML-Diagramms formuliert werden. Bei der OCL handelt es sich um eine typisierte und deklarative Sprache mit einer mathematisch fundierten Semantik. Eine gute Einführung in die OCL findet sich in [WK03].

2.3.4 Modellgetriebene Software-Entwicklung

Modellgetriebene Software-Entwicklung (engl. *Model-Driven Software Development*, MDSD) verfolgt das Ziel, durch den Einsatz von Modellen, die auf verschiedenen Abstraktionsebenen spezifiziert werden können, und von zugehörigen Generatoren die Komplexität im Erstellungsprozess für den Software-Entwickler zu reduzieren und bei steigender Entwicklungsgeschwindigkeit qualitativ hochwertigere Ergebnisse sicherzustellen. Ziel ist es, den gesamten Prozess der Software-Entwicklung, von der fachlichen Anforderungsanalyse und der Aufarbeitung der Fachdomäne in der Analyse-Phase, dem Entwurf der Software-Architektur bis hin zur Implementierung des Zielsystems in Modellen abzubilden, sodass das System selbst zu einem hohen Anteil generativ, also über Modelltransformation, erzeugt wird. Während dieser Ansatz viele Freiheitsgrade gibt, hat die OMG in der *Model Driven Architecture* (MDA) [OMG-MDA], einer Spezialisierung der modellgetriebenen Software-Entwicklung, einige Konkreti-

sierungen vorgenommen. Im MDA-Handbuch [OMG-MDA-Guide] stellt die OMG die Grundprinzipien der MDA dar, die in [Me07] zusammengefasst werden:

- Software-Entwicklung findet auf Basis von (formalen) Modellen statt.

- Die formale Definition der Modelle über eine Metamodellhierarchie erlaubt eine Integration der verschiedenen Modelle und ermöglicht Transformationen zwischen ihnen.

- Die eigentliche Entwicklung von Software-Systemen erfolgt dann durch die Erstellung von Modellen, die durch Modelltransformationen miteinander verbunden sind.

Die Bezeichnung „modellgetrieben“ soll insbesondere herausstellen, dass Modelle zur Steuerung aller Aspekte im Lebenszyklus verwendet werden und nicht wie bisher lediglich dokumentarischer Natur sind. Durch die explizite Nutzung der Modellierung kann auch eine höhere Wiederverwendungsquote bei der Durchführung weiterer Projekte mit derselben Architektur erreicht werden [VS+05].

Die OMG hat bei der Einführung der MDA ein Rahmenwerk definiert, das aus vier verschiedenen Modellebenen und verbindenden Transformationen besteht, die in Abbildung 13 dargestellt sind. Alle vier Modellebenen befinden sich auf der M1-Ebene der UML-Metamodellierungshierarchie.

Das erste der vier Modellebenen wird durch das rechnerunabhängige Modell (engl. *Computation-Independent Model*, CIM) ausgefüllt. Durch das CIM wird ein Software-System zunächst so abstrakt definiert, dass lediglich die Problemdomäne, nicht jedoch die mögliche Umsetzung dargestellt wird. Es wird also eine rein fachliche Sicht eingenommen. Ein CIM kann zur Diskussion und Festlegung von Anforderungen an ein Software-System genutzt werden und bildet eine Brücke zwischen dem Auftraggeber, also dem Kunden beziehungsweise dem Endanwender, und den Software-Entwicklern [VS+05].

Das plattformunabhängige Modell (engl. *Platform-Independent Model*, PIM) ist eines der beiden wichtigsten Modelle der MDA. Ein PIM stellt die Blaupause eines zu entwickelnden Software-Systems dar. Es bleibt dabei nach wie vor

unabhängig von der konkreten Plattform, die zur Umsetzung verwendet wird. Der Plattformbegriff wird in der Literatur relativ gesehen. Je nach Standpunkt kann ein Modell in einem Kontext als plattformunabhängig betrachtet werden, in einem anderen jedoch schon plattformabhängig sein. Erst bei der Analyse zweier Modelle kann eine Relation der Plattformabhängigkeit zwischen ihnen objektiv festgestellt werden.

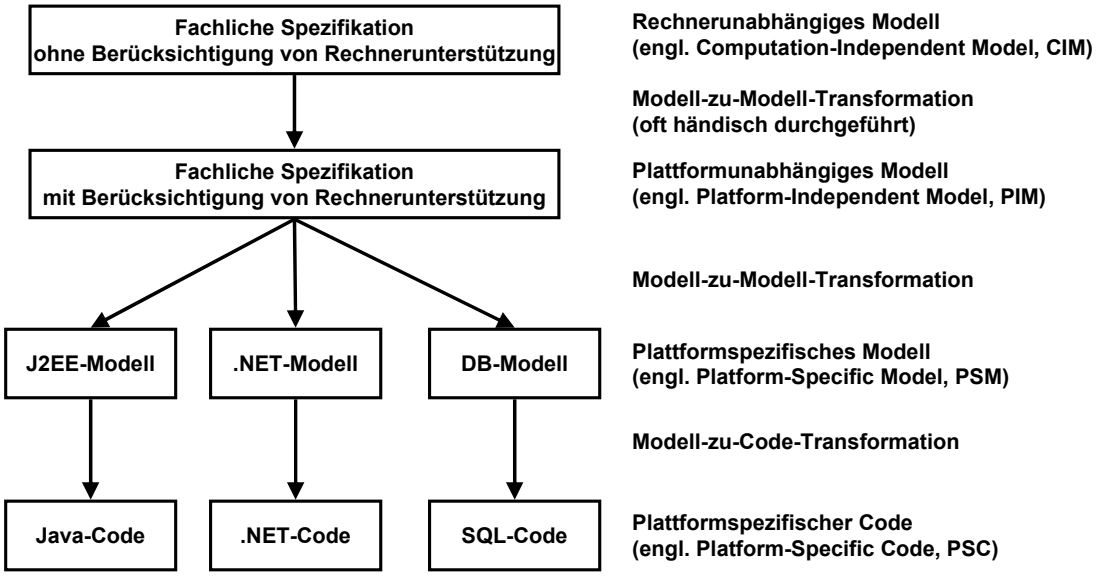


Abbildung 13: Die Modellebenen der MDA

Der Übergang zu plattformabhängigen Modellen (engl. *Platform-Specific Model*, PSM) findet durch eine Modelltransformation statt. Unter einer Modelltransformation versteht man eine berechenbare Abbildung, die einem Quellmodell (hier: PIM) ein Zielmodell (hier: PSM) zuordnet [RH06]. Zur Durchführung dieser Modell-zu-Modell-Transformation werden Metamodelle auf der M2-Ebene eingesetzt, die die Zielplattform an sich beschreiben. Diese Metamodelle werden Plattformmodelle (PM) genannt. Durch die Kombination von einem PIM mit einem PM kann durch Modelltransformation das Zielsystem, welches im Sinne der MDA auch wieder nur ein Modell ist, generiert werden. Durch die Relativität der Plattformabhängigkeit kann es jedoch mehrstufige Transformationen geben. So kann ein PSM auf einer höheren Ebene beispielsweise die Plattform einer Webservice-orientierten Architektur auf Basis von JEE beschreiben, eine konkretere Ebene würde hingegen möglicherweise den Einsatz eines bestimmten *Application Server* auf einem konkreten Betriebssystem darstellen. Das führt dazu, dass es mehrfach zu PIM-PSM-Transformationen kommen kann, wobei

das resultierende PSM aus dem vorangegangenen Transformationsschritt zum PIM der nachfolgenden Transformation wird.

Im letzten Schritt der Software-Entwicklung wird durch sogenannte Modell-zu-Code-Transformationen aus den plattformspezifischen Modellen der ausführbare Code (engl. *Platform-Specific Code*, PSC) erzeugt. Durch die unmittelbare technologische Nähe des PSM zur Zielplattform ist diese Umsetzung durchaus überschaubar.

2.3.5 Muster in der Software-Entwicklung

Im Rahmen eines Software-Entwicklungsprozesses müssen die vom Kunden vorgegebenen Anforderungen auf eine geeignete Software-Architektur abgebildet und implementiert werden. Die während der Entwurfs-Phase getroffenen Entscheidungen beeinflussen maßgeblich das späteren Verhalten des Anwendungssystems sowie dessen Eigenschaften, etwa im Hinblick auf seine Anpassbarkeit oder Leistungsfähigkeit. Häufig treten wiederkehrende Anforderungen auf, für die bereits effektive und effiziente Lösungen entwickelt wurden. Muster dokumentieren bestehendes Entwurfswissen und unterstützen den Software-Entwickler bei der Suche nach geeigneten Lösungen zu bestehenden Entwurfsanforderungen oder Entwurfsproblemen [ES03]. Ein Muster beschreibt eine Lösung für eine bestimmte Klasse von wiederkehrenden Entwurfsproblemen und gibt ein nachgewiesenes, generisches Schema für deren Lösung an [GH+95]. Das Lösungsschema wird durch die Beschreibung der beteiligten Elemente, deren Verantwortlichkeiten und Beziehungen sowie deren Zusammenspiel festgelegt. Beim Einsatz eines Musters müssen die Elemente des Lösungsschemas an den entsprechenden Einsatzkontext angepasst werden.

Muster können sich auf unterschiedlichen Abstraktionsebenen einer Architektur bewegen und lassen sich nach [BM+96] in die folgenden drei Kategorien einteilen:

Ein **Architekturmuster** (engl. *Architectural Pattern*) drückt ein fundamentales, strukturelles Organisationsschema für Anwendungssysteme aus. Durch ein Architekturmuster wird der Grundaufbau, sozusagen das Fundament der Anwendung festgelegt. Es stellt eine Menge an vordefinierten Subsystemen bereit, spezifiziert deren Verantwortlichkeiten und enthält Richtlinien für die Gestaltung

ihrer Beziehungen. Eine dienstorientierte Architektur, wie sie in Kapitel 2.1.3 eingeführt wurde, stellt beispielsweise ein Architekturmuster dar.

Ein **Entwurfsmuster** (engl. *Design Pattern*) stellt ein Schema für die Verfeinerung eines Subsystems oder einer Komponente eines Anwendungssystems oder deren Beziehungen dar. Es beschreibt eine allgemein wiederkehrende Struktur kommunizierender Komponenten, die ein allgemeines Entwurfsproblem in einem bestimmten Kontext lösen [GH+95].

Ein **Idiom** (engl. *Idiom*) ist ein Entwurfsmuster speziell für eine bestimmte Programmiersprache. Es beschreibt, wie bestimmte Aspekte von Komponenten oder deren Beziehungen unter Nutzung der Möglichkeiten einer gegebenen Sprache zu implementieren sind [BM+96].

Neben allgemeinen Mustern lassen sich domänenspezifische Muster definieren, die allgemeine Ansätze in den Einsatzbereich einer bestimmten Fachdomäne übertragen oder neue, domänenspezifische Entwurfslösungen liefern. Die Definition, was ein Muster ist und was nicht, ist unter anderem von der eingenommenen Perspektive abhängig [GH+95].

2.4 Semantische Integration und Ontologien

Benutzerdaten spielen eine wichtige Rolle im Identitätsmanagement und für Zugriffskontrolle. Ein gemeinsames, einheitliches Verständnis über diese Daten ist notwendig, um eine zielgerichtete und effektive Zugriffskontrolle zu ermöglichen.

2.4.1 Benutzerdaten und ihre semantische Integration

Benutzerdaten werden entweder direkt in den Anwendungssystemen gespeichert, in denen sie benötigt werden, oder sie werden in zentralisierten Benutzerverzeichnissen abgelegt. Auch die Datenhaltung selbst kann unterschiedlich durchgeführt werden. Obwohl sich, wie auch schon in Kapitel 2.2.4 dargestellt, Benutzerverzeichnisse ideal dafür eignen, besteht die Möglichkeit, auch andere Speicherformen zu nutzen. Als Beispiele hierfür können relationale Datenbanken oder auch simple Textdateien genannt werden. Um ein effizientes Identitätsmanagement als Vorbedingung für effektive Zugriffskontrolle zu ermöglichen,

müssen die oft vielfältig vorhandenen Benutzerdaten synchron gehalten werden. Dies kann unter dem Begriff der Benutzerintegration zusammengefasst werden. Die Integration dieser heterogenen Daten kann auf unterschiedlichen Ebenen durchgeführt werden, auf einer syntaktischen oder einer semantischen Ebene. Die Syntax beziehungsweise die syntaktische Struktur von Daten wird in [Ab05] definiert als „Beziehungen, die sich unmittelbar aus der Anordnung von Zeichen ergeben“. Ihnen wird eine formale Grammatik zugrunde gelegt, die erlaubte Konstruktionen festlegt und Unerlaubte ausschließt. Im Bereich der LDAP-basierten Benutzerverzeichnisse wird dies durch das LDAP-Informationsmodell, insbesondere durch die strikte Definition von Schemata erreicht. Der Begriff Semantik baut auf dem Syntaxbegriff auf und umfasst weitergehende Beziehungen, die sich aus einer Interpretationsvorschrift ergeben [Ab05]. Das bedeutet, dass einer Menge syntaktisch korrekter Zeichen, Wörter und Sätze eine konkrete Bedeutung zugemessen wird. Der üblicherweise verwendete Integrationsansatz im Benutzerdatenmanagement findet auf einer syntaktischen Ebene statt. Das bedeutet, dass die Datenstruktur jeweils einer Datenquelle und -senke analysiert wird und darauf aufbauend Abbildungsregeln definiert werden. Diese werden dann fest codiert in eine rechnerunterstützte Synchronisationsanwendung eingebracht.

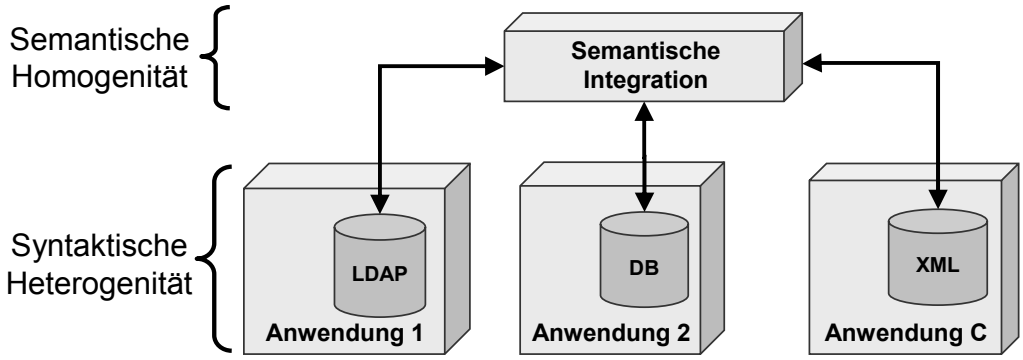


Abbildung 14: Semantische Integration von Benutzerdaten

Um eine semantische Integration von Benutzerdaten zu erreichen, ist eine einheitliche Sichtweise der zugrunde liegenden Daten erforderlich. Obwohl durch die unterschiedliche Repräsentation von Benutzerdaten in unterschiedlicher Datenhaltungstechnologie mit verschiedenen Datenstrukturen in unterschiedlichen Geschäftskontexten eine große syntaktische Heterogenität vorherrscht, lässt sich durch die Reduktion der Daten auf Benutzer, genauer gesagt, digitale Identitäten, auf der inhaltlichen Ebene eine semantische Homogenität feststellen. Die semantische Integration von Daten, wie sie in Abbildung 14 an

dem für diese Arbeit relevanten Bereich der Identitätsdaten von Benutzern dargestellt wird, setzt ein einheitliches Verständnis über die Semantik der betrachteten Daten voraus. Zu der notwendigen Formalisierung der Semantik werden im Allgemeinen Ontologien eingesetzt.

2.4.2 Ontologien zur expliziten Modellierung von Semantik

Die regelmäßig zitierte Definition des Begriffs Ontologie findet sich in [Gr93]:

> *„Eine Ontologie ist eine formale und explizite Spezifikation einer geteilten Konzeptualisierung."*

Dabei wird mit dem Begriff Konzeptualisierung ein abstraktes Modell eines bestimmten Weltausschnitts verstanden, das die für diesen Ausschnitt relevanten Elemente identifiziert. Unter einer formalen Spezifikation wird die Beschreibung mittels einer Notation verstanden, deren Semantik eindeutig definiert ist. Eine solche Notation stellt eine sogenannte formale Sprache dar. Ziel ist die formalisierte und präzise Beschreibung in einem in sich konsistenten und geschlossenen Modell. Definiert werden darin die Klassen von Objekten, Relationen, Abhängigkeiten und Ableitungsregeln [GK04]. Die Betonung, dass die Spezifikation explizit vorliegen muss, stellt keine große Einschränkung dar. Sie meint lediglich, dass die Begriffe nicht nur gedanklich vorliegen dürfen, sondern dass sie konkret notiert werden müssen.

Ontologien, die für Integrationsszenarien eingesetzt werden sollen, müssen spezifisch für die Einsatzdomäne entwickelt werden. Diese syntaktischen Elemente der Ontologie sind Annahmen über Zusammenhänge in einem Problembereich - und müssen sich in konkreten fachlichen Zusammenhängen bewähren. Sofern Informationssysteme einer gemeinsamen Domäne zuzuordnen sind, können ihre Strukturen über eine solche Ontologie integriert werden [GK04]. Dies ist zwar für den Bereich der Anwendungssysteme nicht allgemein gegeben, jedoch hinsichtlich der jeweils verwalteten Benutzerdaten ist die notwendige semantische Homogenität durch das grundsätzliche Verständnis einer digitalen Identität wie im Schalenmodell in 2.2.3 dargestellt.

Um sinnvoll eingesetzt werden zu können, sollten Ontologien analog der Software-Entwicklung auf ingenieurmäßigem Weg erstellt und bestimmte Regeln beachtet werden. Anforderungen an eine Ontologie werden unter anderem

gestellt an ihre Modularität, ihre interne Kohärenz und die Möglichkeit der Erweiterung [SB+98]. Ontologien sind nicht an eine bestimmte formale Sprache gebunden. Es gibt eine Vielzahl an Ontologiesprachen, die auf einer wohldefinierten und rechnerverarbeitbaren Syntax aufbauen. Zusätzlich beinhalten Ontologien einen Ordnungsmechanismus, der die Darstellung von Konzepten, Eigenschaften und Beziehungen ermöglicht. Ein wesentliches Element sind die sogenannten Schlussfolgerungsoperationen (engl. *Reasoning*), die es ermöglichen, Anfragen an die Wissenszusammenhänge einer Ontologie zu stellen [Kl06].

Im Rahmen dieser Arbeit wird als formale Sprache zur Beschreibung von Ontologien die *Web Ontology Language* (OWL) [W3C-OWL] genutzt. OWL basiert technisch auf der Syntax des *Resource Description Framework* (RDF) [W3C-RDF] und historisch auf der *DARPA Agent Markup Language* und des *Ontology Inference Layer* (DAML+OIL). Damit lässt sich eine Hierarchie aufbauen, deren Fundament XML ist. Darauf setzt RDF auf, und auf RDF wiederum OWL.

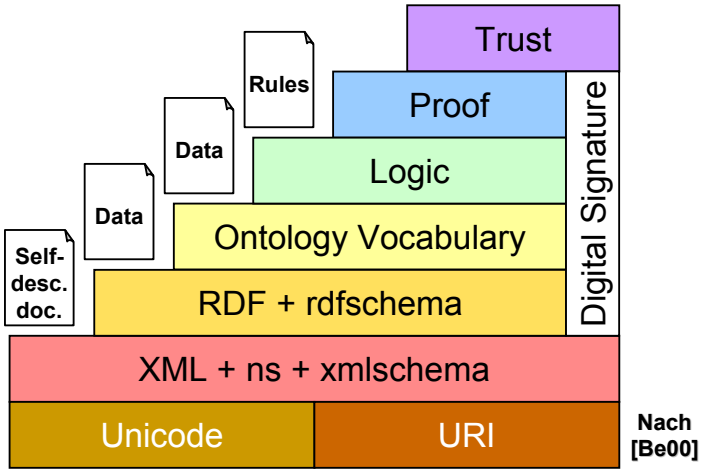


Abbildung 15: Schichtung des Semantic Web

In Abbildung 15, die aus [Be00] entnommen ist, wird die vorgenannte Schichtung aus Sicht von Tim Bernes-Lee verdeutlicht. Während XML einen standardisierten Ansatz bietet, um Daten zu strukturieren, wird mit RDF die Möglichkeit geschaffen, Aussagen (engl. *Statement*) in Form von 3-Tupeln zu formulieren. Ein solches 3-Tupel besteht aus Subjekt, Prädikat und Objekt. Die Begriffe Subjekt und Objekt haben in diesem Kontext nicht die Semantik wie im Bereich des Identitätsmanagements und der Zugriffskontrolle. Da RDF lediglich eine gemeinsame Syntax definiert, wird noch die Festlegung eines gemeinsamen

Vokabulars benötigt. Dies wird mit *Resource Description Framework Schema* (RDFS) [W3C-RDFS] ermöglicht. RDFS bietet die Möglichkeit, das Vokabular für eine bestimmte Domäne einheitlich festzulegen. Mit diesem Ansatz lassen sich schon einfache Ontologien formal beschreiben. Mit OWL wird die Ausdrucksmächtigkeit erweitert. Insbesondere handelt es sich um die Möglichkeit, Disjunktheit und Äquivalenz von zweien oder mehreren Klassen explizit zu modellieren oder Eigenschaften von Beziehungen wie Eindeutigkeit oder Inversität anzugeben [Kl06].

Zur weiteren Vertiefung des Themengebietes der Ontologien sei auf Grundlagenbücher wie [SS04] verwiesen.

2.5 Zusammenfassung

In diesem Kapitel wurden die für das Verständnis dieser Arbeit benötigten Grundlagen gegeben. Um den Themenkomplex „Zugriffskontrolle in dienstorientierten Architekturen“ zu umreißen, wurden dienstorientierte Architekturen betrachtet und in den Themenbereich Zugriffskontrolle eingeführt. Es wurden neben den mathematischen Grundlagen auch Prozesse und Architekturelemente für Zugriffskontrolle dargestellt. Die für den Software-Entwurf der Zugriffskontrolle notwendige Kenntnis im Bereich der Modellierung wurde ebenfalls vermittelt. Um eine formale Modellierung von Autorisierungsdaten zu ermöglichen, wurde in die mathematischen Grundlagen und in die Basismodelle für Zugriffskontrolle eingeführt. Um eine semantische Integration von Benutzerdaten zu erreichen, wurde der Themenbereich der Ontologien betrachtet. Das sich anschließende Kapitel betrachtet den auf den Grundlagen aufbauenden Stand der Forschung und Technik.

3 Stand der Forschung und Technik

In diesem Kapitel werden sowohl der Stand der Forschung als auch das kommerzielle Produktumfeld (Stand der Technik) dargestellt und bewertet. Das Kapitel beginnt mit der Anforderungsspezifikation, die anhand der in Kapitel 1 definierten Ziele vorgenommen wird. Die Struktur dieses Kapitels ist an die Abfolge der nachfolgenden Kapitel angelehnt. Zunächst werden bestehende Zugriffskontroll-Architekturen und konkrete Zugriffskontroll-Produkte thematisiert. Nach der Diskussion der architekturellen Bausteine werden bestehende Zugriffskontroll-Modelle hinsichtlich ihrer Einsetzbarkeit in Webservice-basierten Architekturen untersucht. Dabei wird deutlich, dass architekturelle Veränderungen Weiterentwicklungen an bestehenden Zugriffskontroll-Modellen erforderlich machen. Um Zugriffskontroll-Informationen bereits frühzeitig im Software-Entwicklungsprozess erfassen zu können, müssen Zugriffskontroll-Policies möglichst lange plattformunabhängig gehalten werden. Hierzu wird der Themenkomplex der Zugriffskontroll-Policies aus dem Blickwinkel der modellgetriebenen Software-Entwicklung betrachtet. Policies alleine genügen jedoch nicht, um Zugriffskontrolle durchführen zu können. Eine integrierte Benutzerverwaltung ist ebenfalls eine wesentliche Voraussetzung. Daher werden Lösungen für den Bereich der Benutzerintegration vorgestellt, die zeigen, wie eine unternehmensweite Synchronisation von Benutzerdaten durchgeführt werden kann. Abschließend wird anhand eines Resümees der Handlungsbedarf für diese Arbeit festgestellt.

3.1 Anforderungen

Die Erweiterung einer dienstorientierten Architektur um die Fähigkeit, Dienste mit Zugriffskontrolle zu versehen und diese damit vor unautorisierter Nutzung zu schützen, setzt einerseits architekturelle Ergänzungen im Sinne zusätzlicher Komponenten voraus. Andererseits werden ein Zugriffskontroll-Modell und eine zugehörige Policy-Sprache benötigt, die bei der Formulierung von Zugriffsberechtigungen die Besonderheiten von dienstorientierten Architekturen berücksichtigen. Ausgehend von der in Kapitel 1.3 eingeführten Problemstellung und Zielsetzung im Kontext des in Kapitel 1.2 motivierten Szenarios lassen sich Anforderungen hinsichtlich der erforderlichen Erweiterungen ableiten. Diese Anforderungen werden nachfolgend für die Analyse und Bewertung bestehender Ansätze im Bereich der Zugriffskontrolle in dienstorientierten Architekturen

genutzt. Die durch diese Arbeit vorgestellte Lösung wird ebenfalls gemäß diesem Anforderungskatalog bewertet.

3.1.1 Allgemeine Anforderungen

Es ergeben sich allgemeine Anforderungen, die sich sowohl auf die zu entwickelnde Zugriffskontroll-Architektur als auch auf das Zugriffskontroll-Modell und die Policy-Sprache anwenden lassen. Diese Anforderungen werden nachfolgend dargestellt.

Anforderung A1.1: Allgemeingültigkeit

Sowohl die Zugriffskontroll-Architektur als auch das Zugriffskontroll-Modell und die zugehörige Policy-Sprache müssen allgemeingültig sein. Für den Kontext dieser Arbeit bedeutet das, dass sie unabhängig von einer spezifischen Fachdomäne einsetzbar sein müssen. Einschränkungen hinsichtlich der Allgemeingültigkeit werden lediglich durch die in Kapitel 1.5 getroffenen Prämissen vorgenommen. Prämisse P3 nimmt eine Konkretisierung von dienstorientierten Architekturen auf Webservice-Technologien vor. Die Anforderung nach Allgemeingültigkeit erhöht die Wiederverwendbarkeit des Ansatzes [Ba00].

Anforderung A1.2: Einbeziehung des Fachentwicklers ohne Aufwandssteigerung

Um das Problem der nachgelagerten und entkoppelten Erstellung von Zugriffskontroll-Policies für Fachfunktionalität durch Sicherheitsexperten zu entschärfen, sind die Software-Entwicklungsprozesse anzupassen. Die unmittelbare Verknüpfung von fachlich motivierter Programmlogik und zugehöriger Zugriffsberechtigung macht die Einbindung der Fachentwickler sinnvoll. Die daraus resultierende Komplexitätssteigerung und den damit einhergehenden zusätzlichen Entwicklungsaufwand muss durch den vorgestellten Lösungsansatz jedoch so gering wie möglich gehalten werden. Diese stellt eine Spezialisierung der in [Me07] getroffenen Anforderung für den Bereich der Zugriffskontrolle dar.

3.1.2 Anforderungen an den Entwurf der Zugriffskontroll-Architektur

Neben allgemeinen Anforderungen ergeben sich Anforderungen, die die Zugriffskontroll-Architektur adressieren.

Anforderung A2.1: Interoperable Dienstschnittstellen

Die Zugriffskontroll-Architektur muss ihre Funktionalität an klar spezifizierten und herstellerunabhängig definierten Dienstschnittstellen bereitstellen. Dies projiziert die Grundsätze dienstorientierter Architekturen auf die Zugriffskontroll-Architektur und verringert die Komplexität für den Fachentwickler, da er Zugriffskontrolle als Dienst einbinden kann. Um eine interoperable Verknüpfung zwischen Zugriffskontroll-Architektur als Dienstgeber und der Webservice-basierten Facharchitektur zu erreichen, ist die Dienstschnittstelle als Webservice auszuprägen und mittels *Web Services Description Language* (WSDL) zu beschreiben. Diese Anforderung nach Dienstorientierung auch in der Zugriffskontroll-Architektur korreliert damit unmittelbar mit der Anforderung A1.2 und wird unter anderem in [AZ+07, Re05, HH+06] gefordert. Die Ausprägung der Dienste als Webservices erleichtert die Schaffung von syntaktischer Interoperabilität zwischen *Application Server* der Facharchitektur und den Zugriffskontroll-Diensten auch beim Einsatz von Produkten unterschiedlicher Hersteller in der Facharchitektur. Auf die Notwendigkeit von Interoperabilität insbesondere im Sicherheitskontext wird durch [Ra02a] hingewiesen.

Anforderung A2.2: Portabilität

Die Erweiterung der Produkte in der Facharchitektur zur Verknüpfung mit der Zugriffskontroll-Architektur muss portabel gestaltet sein. Die Unterstützung von Heterogenität, im Speziellen hinsichtlich der *Application Server*, also der Laufzeitumgebungen für Webservices und auch hinsichtlich der BPEL-*Engines*, den Laufzeitumgebungen für Webservice-Kompositionen, soll ohne Vergrößerung des Entwicklungsaufwandes möglich sein. Diese Forderung wird unter anderem in [Ra02a] gestellt.

Anforderung A2.3: Integrationsfähigkeit

Im Sicherheitsbereich gibt es eine große Zahl kommerzieller Produkte, die derzeit die Durchführung von Zugriffskontrolle sicherstellen. Ihre Entwicklung begann lange vor der Einführung dienstorientierter Architekturen, und sie werden erst sukzessive an die Gegebenheiten dienstorientierter Architekturen angepasst. Ziel ist es, die zu entwerfende Zugriffskontroll-Architektur als Blaupause zu verstehen, die sich hinsichtlich der Integrationsfähigkeit bestehender Sicherheitsprodukte messen lassen kann. Diese Anforderung leitet sich nach [SN+05] daraus ab, dass es viele Fallstricke bei der konkreten Implementierung von Sicherheitstechnologie gibt und eine Individualentwicklung wegen der fehlenden Domänenspezifika üblicherweise nicht notwendig ist. Die Forderung nach Flexibilität bei der Produktunterstützung im Sicherheitsbereich findet sich in [Ec06].

3.1.3 Anforderungen an das Zugriffskontroll-Modell und die Sprache zur Spezifikation von Zugriffskontroll-Policies

Auch an das Zugriffskontroll-Modell und die zugehörige Sprache zur Policy-Spezifikation ergeben sich gezielte Anforderungen.

Anforderung A3.1: Plattformunabhängigkeit und Webservice-Bezug

Das Zugriffskontroll-Modell muss plattformunabhängig hinsichtlich der tatsächlich eingesetzten Sicherheitskomponenten und -produkte gestaltet sein und sich damit rein auf die entwickelte Funktionalität in der Facharchitektur beziehen. Dies erlaubt die Einbeziehung von Fachentwicklern und Geschäftsanalysten (engl. *Business Analysts*) bei der Policy-Entwicklung. Unter Berücksichtigung der Prämissen P3 und P4 ist ein Zugriffskontroll-Modell notwendig, das die Eigenschaften von Webservice-basierten, dienstorientierten Architekturen zur Abbildung voll automatisiert umsetzbarer Geschäftsprozesse berücksichtigen.

Anforderung A3.2: Einbindung in den Software-Entwicklungsprozess

Die in Anforderung A3.1 geforderte Plattformunabhängigkeit definiert nur die Mengen und Relationen, die zur Durchführung einer Zugriffskontroll-Entscheidung herangezogen werden können. Zur effektiven Anwendung muss jedoch eine zugehörige Policy-Sprache bereitgestellt werden, die analog die

Abstraktion auf ein rein fachliches Niveau ermöglicht. Dies ermöglicht die Spezifikation der Zugriffskontroll-Policies bereits während frühen Phasen in der Software-Entwicklung [Lo03]. Die zu erstellenden Artefakte müssen jedoch formal spezifiziert werden, um eine spätere, möglichst automatisierte Transformation in effektiv auswertbare Zugriffskontroll-Policies eines hinter der Dienstschnittstelle der Zugriffskontroll-Architektur eingesetzten Sicherheitsprodukts zu ermöglichen. Diese Anforderung ist unmittelbar mit der Anforderung A1.2 verknüpft.

3.1.4 Anforderungen an die modellgetriebene Entwicklung von Zugriffskontroll-Policies

Es besteht ein Bedarf, den Bereich der Zugriffskontrolle ebenso systematisch anzugehen wie den der fachlichen Software-Entwicklung. Gerade die Auftrennung in Kern- und Querschnittsaspekte analog den Grundsätzen der aspektorientierten Programmierung ermöglichen eine getrennte Entwicklung, die trotzdem über Verknüpfungspunkte zusammengeführt werden kann. Dies ermöglicht die Ausweitung des modellgetriebenen Ansatzes auch auf den Bereich von Zugriffskontroll-Policies. Darauf aufbauend ergeben sich folgende Anforderungen, die im Kontext dieser Arbeit behandelt werden.

Anforderung A4.1: Abstraktion und explizite Definition von Transformationen

Aufbauend auf der durch Anforderung A3.1 geforderten plattformunabhängigen Policy-Sprache muss ein Vorgehen zur Entwicklung eines plattformabhängigen Zugriffskontroll-Modells aufgezeigt werden, das eine Sprache für Zugriffskontroll-Policies einer konkreten Zugriffskontroll-Architektur darstellen kann. Zwischen der plattformunabhängigen und der plattformspezifischen Sprache sollen dann explizit definierte Transformationen die Verwendung eines modellgetriebenen Ansatzes ermöglichen. Die Vorgehensweise, zunächst eine produktunabhängige Spezifikation durchzuführen (Abstraktion), die anschließend über explizite Transformationen um die notwendigen Produktspezifika ergänzt wird, wird unter anderem durch die OMG in [OMG-MDA-Guide] gefordert.

3.1.5 Anforderungen an eine semantische Integration von Benutzerdaten

Die Integration der organisationsweit relevanten Benutzerdaten in Form von digitalen Identitäten stellt eine wesentliche Voraussetzung für die Schaffung einer ebenfalls unternehmensweiten Zugriffskontroll-Architektur dar. Harte Verdrahtung der Quell- und Zielstrukturen eines Synchronisationsvorgangs stellen die Regel dar. Ein übergreifendes und explizit modelliertes Verständnis über die Semantik lässt sich nicht erkennen. Ausgehend von dem in diesem Kapitel erarbeiteten Stand der Technik lassen sich folgende Anforderungen an eine semantische Integration von Benutzerdaten formulieren.

Anforderung 5.1: Explizite Definition der Semantik

Die Semantik der digitalen Identitäten muss durch ein gemeinsames Modell explizit dargestellt werden, das plattformunabhängig zu einem konkreten Produkt sein sollte. Die Nutzung von Ontologien zur Schaffung eines gemeinsamen Semantik-Verständnisses ist zu betrachten.

3.1.6 Anforderungskatalog

Die in den Abschnitten 3.1.1 bis 3.1.5 beschriebenen Anforderungen werden in Tabelle 1 übersichtlich zu einem Anforderungskatalog zusammengefasst. Dieser wird sowohl zur Bewertung des in Kapitel 3 beschriebenen Stands der Forschung und Technik herangezogen werden als auch für die Entwicklung eines eigenen Ansatzes für Zugriffskontrolle in dienstorientierten Architekturen (Kapitel 4 ff.).

Anforderungskatalog	
Allgemeine Anforderungen	
A1.1	Allgemeingültigkeit
A1.2	Einbeziehung des Fachentwicklers ohne Aufwandssteigerung
Anforderungen an den Entwurf der Zugriffskontroll-Architektur	
A2.1	Interoperable Dienstschnittstellen

A2.2	Portabilität
A2.3	Integrationsfähigkeit
Anforderungen an die Sprache zur Spezifikation von Zugriffskontroll-Policies	
A3.1	Plattformunabhängigkeit und Webservice-Bezug
A3.2	Einbindung in den Software-Entwicklungsprozess
Anforderungen an die modellgetriebene Entwicklung von Zugriffskontroll-Policies	
A4.1	Abstraktion und explizite Definition von Transformationen
Anforderungen an eine semantische Integration von Benutzerdaten	
A5.1	Explizite Definition der Semantik

Tabelle 1: Anforderungskatalog

3.2 Zugriffskontroll-Architekturen

Um Zugriffskontrolle durchführen zu können, werden entsprechende Komponenten benötigt, die die Facharchitektur erweitern. Diese werden unter dem Begriff „Zugriffskontroll-Architektur“ zusammengefasst. Nachfolgend werden Zugriffskontroll-Architekturen aus Forschung und Technik vorgestellt und zugehörige Sicherheitsstandards diskutiert.

3.2.1 Forschungsarbeiten im Bereich Zugriffskontroll-Architekturen

Die meisten wissenschaftlichen Arbeiten, die sich mit Zugriffskontrolle beschäftigen, fokussieren sich auf einen ihrer Teilaspekte wie kryptografische Methoden, Authentifizierung, Autorisierung oder Zugriffskontroll-Modelle. Die Arbeiten, die es sich zum Ziel setzen, eine Zugriffskontroll-Architektur als Ganzes für Webservice-Umgebungen aufzubauen, lassen sich zwei Strömungen zuordnen: Einerseits finden sich sicherheitsbereichsbasierte Architekturen (engl. *Perimeter-Based Security Architecture*), die den Anfragenden je nach seiner räumlichen Entfernung eine Anzahl an *Firewalls* (dt. virtueller Schutzzaun) überwinden lässt. Diese *Firewalls* können auf unterschiedlichen Schichten arbeiten, sei es im

Transportsystem oder auf Anwendungs-, also Nachrichtenebene. Andererseits gibt es Architekturen, die unmittelbar die in Kapitel 2.2.6 eingeführten Elemente *Policy Enforcement Point* (PEP) und *Policy Decision Point* (PDP) aufgreifen und mit spezifischen Kommunikationsprotokollen und Verteilungsschemata ausprägen. Nachfolgend wird für beide Realisierungsformen jeweils ein relevanter Stellvertreter untersucht und bewertet.

3.2.1.1 Firewall-basierte Ansätze

Exemplarisch für den Bereich der *Firewall*-basierten Zugriffskontroll-Architekturen wird die Nedgty-*Firewall* aus [BS+05] betrachtet. Sie stellt eine Weiterentwicklung traditioneller *Firewalls* dar, die im Transportsystem und damit bezogen auf TCP/IP-basierte Netzwerke auf Schicht 3 (IP) beziehungsweise Schicht 4 (TCP/UDP) arbeiten. Die Nedgty-*Firewall* nutzt dieselben Funktionsweisen, betrachtet aber nicht die Steuerungsdaten der Transportschicht, sondern den eigentlichen Nachrichteninhalt. Das Ziel der Nedgty-*Firewall* ist es, die zwischen Webservices ausgetauschten SOAP-Nachrichten zu untersuchen und anhand des Absenders, des Empfängers sowie weitere Inhalte der Nachricht eine Durchleitung zu erlauben oder nicht. Dazu setzt die Architektur auf die bestehenden unternehmensweiten *Firewalls* auf und filtert den sogenannten Port-80-Verkehr aus. Port-80-Verkehr beinhaltet jegliche HTTP-Kommunikation, die auch von SOAP genutzt wird. Ein vollständiges Ausfiltern allen Port-80-Verkehrs an der Bereichsgrenze würde auch die nicht-SOAP-basierte Kommunikation wie die *Web Server*-Anfragen betreffen. Daher findet sich ein SOAP-Filter als erstes Element der Nedgty-*Firewall*. Der SOAP-Filter gibt alle Nicht-SOAP-Pakete unbehandelt wieder in das ursprüngliche System zurück. Alle SOAP-Pakete werden dann intern behandelt, nach einer Deserialisierung werden die Nutzdaten der SOAP-Nachricht an einen Validierer übergeben, der passende Policies aus der Policy-Datenbank anfragt und darauf aufbauend die Zugriffskontroll-Entscheidung trifft. Bei positiver Entscheidung muss durch das System aus technischen Gründen eine neue SOAP-Nachricht erzeugt werden, die dann über das Transportsystem an den eigentlichen Empfänger durchgestellt wird.

Bewertung

Firewall-basierte Lösungen sind allgemeingültig einsetzbar im Sinne der Anforderung A1.1, da sie keine Einschränkungen hinsichtlich einer Fachdomäne

vornehmen, sondern lediglich den in Webservice-Umgebungen üblichen SOAP-Nachrichtenverkehr voraussetzen. Jedoch ist der Fachentwickler in diesem Konzept nicht als Informationsgeber für die Zugriffsberechtigungen eingebunden. Dies führt zu dem in Kapitel 1.3 eingeführten Problem der nachgelagerten Policy-Erstellung durch Sicherheitsexperten und zur Nichterfüllung der Anforderung A1.2.

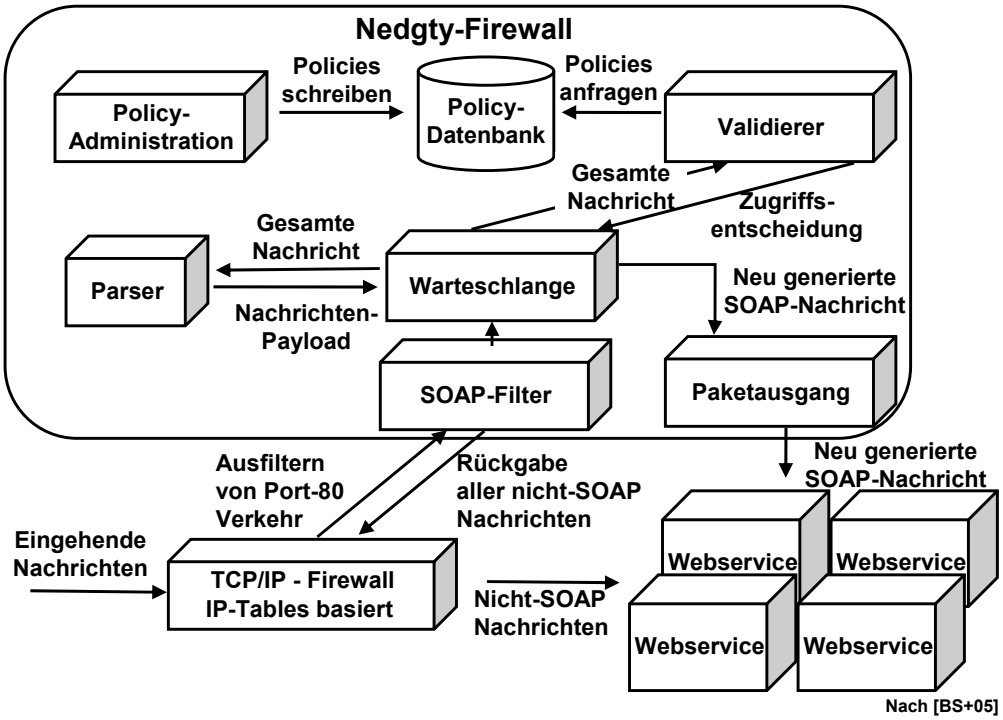


Abbildung 16: Architektur der Nedgty-Firewall für Webservices

Die angebotenen Dienstschnittstellen zwischen dem Ausfilterungspunkt und der *Firewall*-Architektur sind in einem proprietären Format spezifiziert. Dies verhindert nicht nur die Erfüllung der Anforderung A2.1, sondern zusätzlich auch von A2.3, da auch die Integration bestehender Sicherheitsprodukte sich sehr schwierig gestaltet. Die in A2.2 geforderte Portabilität der Lösung ist auf Betriebssystemebene gegeben, da die *Firewall* lediglich ein Unix/Linux-basiertes Betriebssystem voraussetzt. Eine Portierung der IP-Tables-basierten *Firewall* auf ein anderes (Linux-)Betriebssystem ist daher vergleichsweise problemlos möglich. Die eingesetzten Policies sind direkt auf diese Lösung zugeschnitten und dadurch nicht plattformunabhängig. Die Anforderungen A3.1 und A3.2, also die Plattformunabhängigkeit der Policies sowie die Einbindung in den Software-Entwicklungsprozess sind damit nicht gegeben. Daher ist die Betrachtung der Anforderung A4.1 obsolet, da keine Policy-Transformationen in diesem Ansatz

vorgesehen sind, ebenso wenig wie die in Anforderung A5.1 geforderte Integration von Benutzerdaten.

Ein allgemeines Problem von *Firewall*-basierten Zugriffskontroll-Systemen ist die Positionierung der *Firewall* als Aussprungpunkt für die Zugriffskontroll-Entscheidung. *Firewalls* dienen als Filter und damit als virtuelle Eingangstür zu einem Sicherheitsbereich. Dadurch wird eine Innen-/Außensicht erzielt, die davon ausgeht, dass potenzielle Störer nur außerhalb dieses Bereichs aktiv sind, nicht jedoch innerhalb. *Firewall*-Konzepte sind analog eines Grenzzauns um ein Unternehmen mit einer definierten Pforte zu sehen. Auch wenn es zwischenzeitlich üblich ist, mehrstufig *Firewalls* analog einer verschachtelten, zunehmenden Sicherheitstiefe einzusetzen, ergeben sich Probleme durch eine solche Lösung hinsichtlich der Durchsetzung von Zugriffskontrolle im Webservice-Umfeld. Webservices agieren autonom, das heißt, sie können nicht nur von außen, sondern auch von innen aufgerufen werden, und damit mit einem nicht durch die *Firewall* untersuchten Nachrichtenverkehr. Gerade durch explizite Dienstkompositionen ist es im Webservice-Kontext üblich, dass Webservices sich gegenseitig aufrufen. Diese Aufrufe würden innerhalb des gleichen Sicherheitsbereiches durchgeführt werden und damit nicht durch die *Firewall* untersucht werden. Das Problem ließe sich lösen, in dem die Sicherheitsbereiche immer enger gezogen werden. Im Extremfall wäre jeder einzelne Webservice mit einer solchen *Firewall* zu versehen, was jedoch für die vorgestellte Architektur mit einer großen Zahl an unterschiedlichen Webservices und Webservice-Kompositionen nicht skalieren würde. Während *Firewall*-basierte Lösungen zwar keine Lösung zur Realisierung von Zugriffskontrolle für Webservices bieten, lassen sie sich jedoch als Schutzelement für andere Zwecke einsetzen. Beispielsweise können extern initiierte, verteilte Störungsangriffe (engl. *Distributed Denial of Service*, DDoS) erkannt und unmittelbar an der Unternehmensgrenze ausgefiltert werden. Ebenfalls können SOAP-Nachrichten auf ihre grundsätzliche Korrektheit überprüft werden [HH07], beispielsweise ob ein syntaktisch korrektes Sicherheitstoken enthalten ist.

3.2.1.2 PEP-/PDP-basierte Ansätze

Vielversprechendere Ansätze basieren unmittelbar auf der Ausprägung der in Kapitel 2.2.6 eingeführten Kernelemente *Policy Enforcement Point* (PEP) und *Policy Decision Point* (PDP). Der PEP wird dabei in unmittelbarer räumlicher Nähe zu der Fachfunktionalität positioniert, für die er Zugriffskontrolle ermögli-

chen soll. Exemplarisch wird im Folgenden die durch die von der OASIS in der *Extensible Access Control Markup Language* (XACML) [OASIS-XACML2.0] spezifizierte Zugriffskontroll-Architektur diskutiert.

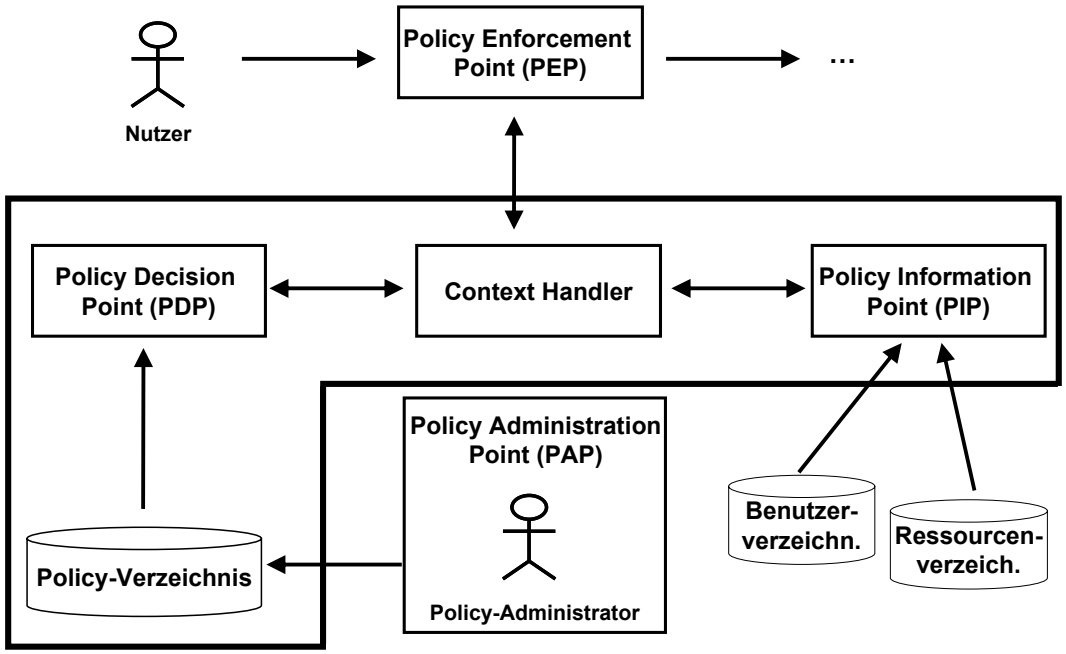


Abbildung 17: Die Zugriffskontroll-Architektur von XACML

Die Komponenten, die dem Kern einer XACML-basierten Zugriffskontroll-Architektur zugeordnet werden, wurden in Abbildung 17 fett umrandet. Es handelt sich dabei um den *Context Handler* als Dienstzugangspunkt der Zugriffskontroll-Architektur, den *Policy Decision Point*, der auf die reine Auswertung einer Policy bei vorliegenden Attributinformationen reduziert wird, das Policy-Verzeichnis sowie den *Policy Information Point*, der die notwendigen Attribute aus externen Quellen nachlädt. Initial sind durch den Policy-Administrator über den *Policy Administration Point* (PAP) eine Grundmenge an Policies bereitzustellen und in einem gemeinsamen Policy-Verzeichnis abzulegen. Die Zugriffskontrolle beginnt mit dem Zugriff eines Benutzers, der vom *Policy Enforcement Point* (PEP) abgefangen wird. Dieser übergibt die Anfrage an den *Context Handler*, der im Gegensatz zum PEP der XACML-Architektur zugeordnet wird. Der *Context Handler* kann als Mediator zwischen den weiteren Komponenten der XACML-Architektur verstanden werden. Er leitet die Anfrage an den *Policy Decision Point* (PDP) weiter. Dieser wählt die auszuwertenden Policies aus und fordert dann die für die Durchführung der Zugriffskontroll-Entscheidung notwendigen Attribute über Benutzer und Ressourcen über den *Context Handler* vom *Policy Information Point* (PIP) an. Dieser lädt die Daten

aus den angeschlossenen Benutzer- und Ressourcen-Verzeichnissen nach, über die der XACML-Standard jedoch keinerlei Aussagen trifft, weder über ihre Anbindung noch über ihr zugrundeliegendes Informationsmodell. Die Attribute werden dann über den *Context Handler* zum *Policy Decision Point* weitergegeben, der dann die Policy im Sinne einer algebraischen Operation auswertet. Als Rückgabewerte werden dem reinen „Zugriff erlaubt“ (engl. *Permit*) beziehungsweise „Zugriff verboten“ (engl. *Deny*) auch die Möglichkeiten „kein Ergebnis“ (engl. *Indeterminate*) und keine Policy anwendbar (engl. *Not Applicable*) vorgesehen.

Bewertung

Die in XACML definierte Zugriffskontroll-Architektur ist allgemeingültig. Sie trifft weder eine Einschränkung hinsichtlich einer Fachdomäne noch ist sie auf dienstorientierte Architekturen oder noch präziser, den Webservice-Kontext, beschränkt (A1.1). Es ist jedoch keine Einbeziehung von Fachentwicklern vorgesehen (A1.2). Die Dienstschnittstellen selbst, insbesondere die zwischen PEP und *Context Handler* werden durch die XACML-Spezifikation nicht präzise definiert. Dies stellt ein Hindernis für die in Anforderung A2.1 geforderte Interoperabilität der Dienstschnittstellen dar; theoretisch ist jedoch eine WSDL-basierte Schnittstellenbeschreibung denkbar. Die Komponenten an sich können portabel gestaltet werden, je nach Implementierungsform des *Context Handlers*. Wenn dieser nicht unmittelbar in die herstellerspezifischen Nachrichten-*Queues* der *Application Server* eingehängt wird, sondern als separate Komponente entwickelt wird, kann er einfach in einen anderen *Application Server* umgezogen werden. Der *Policy Decision Point* selbst wird üblicherweise als eigenständige Komponente realisiert. Die Anforderung A2.3 nach Integrationsfähigkeit bestehender Produkte ist problematisch, da nicht im Standard vorgesehen. Einige Hersteller haben jedoch die Erweiterung ihrer Produkte zur Einbindung in eine XACML-Architektur im Laufe der nächsten Jahre geplant. Zurzeit (Dezember 2007) gibt es jedoch noch kein Produkt mit einsetzbarer XACML-Implementierung. Da XACML neben der Architektur auch eine ausführliche Policy-Sprache beinhaltet, wird dieses hinsichtlich der Anforderungsblöcke A3 und A4 gesondert in Kapitel 3.3 diskutiert. Die Benutzerintegration (A5.1) steht nicht im Fokus von XACML.

3.2.1.3 Realisierungsmöglichkeiten eines PEP

Die Realisierung von Zugriffskontrolle an sich stellt keinen neuen Themenbereich dar und ist nur durch veränderte Randbedingungen ein Spezifikum dienstorientierter Architekturen. Aus architektureller Sicht ist die Verknüpfung des zu schützenden fachlichen Bereichs mit der Zugriffskontrolle besonders zu betrachten. In der Vergangenheit wurde eine Vielzahl unterschiedlicher Möglichkeiten für diese Verknüpfung entwickelt. Einige davon wurden als Entwurfsmuster (siehe Kapitel 2.3.5) beschrieben. Drei davon sind für den Bereich der Webservice-orientierten Architekturen besonders relevant und werden nachfolgend diskutiert. Mit dem Referenzmonitor wird eine abstrakte Lösung vorgestellt, die mit den Entwurfsmustern *Secure Service Proxy* (SSP) und *Intercepting Web Agent* (IWA) konkretisiert werden.

Referenzmonitor

Das Konzept des Referenzmonitors (engl. *Reference Monitor*) stellt eine Möglichkeit dar, um Zugriffskontrolle auf eine zu schützende Ressource zu ermöglichen. Das Konzept des Referenzmonitors wird unter anderem in [Am94] vorgestellt. Der Referenzmonitor stellt eine logische Einheit dar, die für die Kontrolle und Durchsetzung von Zugriffsberechtigungen zuständig ist. Hinsichtlich der Einordnung in die Komponentenstruktur einer Zugriffskontroll-Architektur, wie sie in Kapitel 2.2.6 motiviert wurde, lässt sich der Referenzmonitor als *Policy Enforcement Point* (PEP) einordnen. Die Art und Weise, wie die Zugriffskontroll-Entscheidung letztlich getroffen wird, wird jedoch nicht betrachtet. Der Referenzmonitor entscheidet für jeden Zugriff eines Subjektes auf ein Objekt anhand von vorher definierten Regeln, ob der Zugriff erlaubt wird. Der Referenzmonitor ist sehr eng (aus technischer und semantischer Sicht) an das abzusichernde System gekoppelt. Für eine nachträgliche Ankopplung eines Referenzmonitors an ein zu schützendes System muss tief in die Interna dieses Systems eingegriffen werden. Neben der Vorschaltung des Referenzmonitors muss sichergestellt werden, dass die Subjekte nicht mehr direkt auf die eigentlichen Objekte zugreifen können, sondern nur über den Referenzmonitor in Kontakt mit den Objekten treten können. Dazu ist es wichtig, dass der Referenzmonitor und die Daten, auf denen er seine Zugriffsentscheidungen trifft, vor Fremdeinwirkung geschützt werden. Für die Beurteilung des Sicherheitsniveaus eines Systems können die Eigenschaften des Referenzmonitors herangezogen

werden, unter anderem kann die konkrete Implementierung einer formalen Verifikation unterzogen werden.

Bewertung

Das Konzept des Referenzmonitors ist sehr allgemein spezifiziert und erfüllt damit die Anforderung A1.1. Um das Konzept im Webservice-Kontext zum Einsatz zu bringen, sind jedoch weitere Präzisierungen erforderlich. Der Software-Entwickler der Facharchitektur muss sich selbst um die Implementierung der Komponenten und der zugehörigen Zugriffskontroll-Informationen kümmern (A1.2). Hinsichtlich der Schnittstellen werden durch das Konzept keinerlei Vorgaben gemacht (A2.1). Theoretisch könnten diese auf Basis von WSDL erstellt werden. Durch die feste Verdrahtung zwischen Referenzmonitor und Fachkomponente ist seine Portabilität erschwert (A2.2). Die Integration bestehender Sicherheitsprodukte steht nicht im Vordergrund (A2.3), ebenso wenig werden Aussagen getroffen über die konkreten Zugriffskontroll-Policies (A3.1 und A3.2). Modellgetriebene Policy-Transformationen (A4.1) und die formale Spezifizierung der Semantik der Benutzerdaten (A5.1) sind ebenfalls nicht Kern dieses Konzepts und lassen sich daher nicht bewerten.

Die nachfolgend beschriebenen Entwurfsmuster *Secure Service Proxy* und *Intercepting Web Agent* stellen relevante Konkretisierungen des Referenzmonitors dar.

Secure Service Proxy

Das Einbringen neuer Sicherheitsprotokolle in bestehende Software-Systeme kann erhebliche Schwierigkeiten und Sicherheitsrisiken mit sich bringen. Gerade bei der Migration bestehender Software-Systeme in dienstorientierte Architekturen ist diese Situation gegeben. Die Bereitstellung der Fachfunktionalität an standardisierten Schnittstellen, die auch Sicherheitsfragestellungen berücksichtigt, macht die Integration neuer, einheitlicher Sicherheitsprotokolle erforderlich. Einen Lösungsansatz, der Änderungen im bestehenden Software-System gering hält, ist der *Secure Service Proxy* (SSP), der in [SN+05] als Entwurfsmuster beschrieben wird. Er ermöglicht es, sämtliche Prozesse zur Realisierung der Zugriffskontrolle extern durchzuführen. Der SSP setzt auf dem Entwurfsmuster Stellvertreter (engl. *Proxy*) auf, der ein gängiges Muster in der Software-Entwicklung darstellt. Das Muster verschiebt die Kontrolle über ein Objekt auf

ein vorgelagertes Stellvertreterobjekt, der Zugriff auf das Objekt läuft ausschließlich über den Stellvertreter. Der SSP stellt eine Spezialisierung des allgemeinen Stellvertreter-Musters dar, der den Aspekt der Zugriffskontrolle in den Vordergrund rückt. Individuell entwickelte Software-Systeme, die durchaus mit verschiedenen Programmiersprachen entwickelt wurden und sich in ihrem internen Aufbau deutlich unterscheiden können, sollen ihre Fachfunktionalität an einheitlichen Dienstschnittstellen zur Verfügung stellen. Um diese Schnittstellen nicht unmittelbar in das Altsystem (engl. *Legacy System*) einbringen zu müssen, was je nach eingesetzter Technologie einen nicht zu unterschätzenden Aufwand darstellt, wird im Bereich der Webservice-orientierten Architekturen ein anderes Vorgehen verwendet, das sogenannte *Wrapping to Web Services*. In vorgelagerten *Application Server* werden Stellvertreterobjekte platziert, die mit den proprietären Protokollen auf die individuellen Schnittstellen der diensterbringenden Altsysteme zugreifen. Aus Sicht der dienstorientierten Architektur stellt der SSP die Dienstschnittstelle dar. Den Sachverhalt, dass die eigentliche Implementierung der Fachfunktionalität in einem anderen System durchgeführt ist, wird durch dieses Konzept verschattet. Der SSP stellt in diesem Rahmen sowohl die Authentifizierung sicher und führt auch die eigentliche Zugriffskontrolle durch. Durch die Eigenschaft als Stellvertreter findet eine Auftrennung der Kommunikation in die zwischen *Client* und SSP und die zwischen SSP und Altanwendung statt. Dies ermöglicht die Nutzung unterschiedlicher Protokolle, was insbesondere im Webservice-Kontext von Vorteil ist. Altsysteme sind von Haus aus oft nicht in der Lage, mit den Protokollen und Standards aus dem Webservice-Kontext umzugehen. Zusätzlich können weitere Eigenschaften des Stellvertreter-Musters genutzt werden, wie die Möglichkeit der Lastverteilung auf mehrere gleichartige Anwendungssysteme. Auch diese Eigenschaft wird durch den SSP gegenüber dem *Client* verschattet. Das Entwurfsmuster lässt Freiheitsgrade, wie eine Anwendung ihre Funktionalität über SSP bereitstellen kann. Es besteht die Möglichkeit einer 1:1-Verknüpfung, das heißt, alle zur Verfügung zu stellende Funktionalität wird in einem einzigen SSP zusammengefasst. Dieser muss dann alle relevanten Methoden selbst wieder bereitstellen. Der Nachteil ist, dass sich bei einer Bereitstellung des SSP als Webservice die Schnittstellenbeschreibung in WSDL ändert, was ein neues Einbinden des SSP-basierten Webservices auf Seiten des Dienstnehmers notwendig macht. Es spricht jedoch nichts dagegen, die Funktionalität einer Altanwendung auch auf mehrere SSP aufzuteilen. Ebenfalls besteht die Möglichkeit, einen SSP mit mehren Instanzen einer Altanwendung zu verknüpfen, was eine Lastverteilung ermöglicht sowie die Ausfallsicherheit erhöhen kann.

Bewertung

Der *Secure Service Proxy* ist hinreichend allgemeingültig, um als Baustein einer Zugriffskontroll-Architektur für Webservice-Umgebungen einsetzbar zu sein. Er erfüllt damit die Anforderung A1.1. Der Software-Entwickler der Facharchitektur muss selbst die Entwicklung der zusätzlichen Zugriffskontroll-Komponenten und –Policies durchführen, was jedoch zu einem nicht zu unterschätzenden Aufwand führt, der durch das Entwurfsmuster jedoch nicht thematisiert wird (A1.2). Über die Schnittstellen sagt das Muster nichts aus. Theoretisch können sie per WSDL interoperabel für den Webservice-Kontext bereitgestellt werden (A2.1). Da die SSP-Komponente nicht unmittelbar mit dem *Application Server* verwoben ist, ist eine Portabilität grundsätzlich gegeben (A2.2). Die Integration bestehender Sicherheitsprodukte wird nicht betrachtet, ist jedoch durch den Einsatz des SSP nicht ausgeschlossen (A2.3). Hinsichtlich der Zugriffskontroll-Policies werden keine Aussagen getroffen (A3.1 und A3.2). Ebenfalls wird der Bereich der modellgetriebenen Policy-Entwicklung einerseits (A4.1) und der Benutzerintegration (A5.1) andererseits nicht behandelt.

Intercepting Web Agent

Die Nachrüstung von einheitlichen. vorgeschalteten Authentifizierungs- und Zugriffskontroll-Komponenten für jede bestehende Anwendung im Rahmen der Integration in einer dienstorientierten Architektur kann aufwendig und kostenintensiv sein. Um einerseits die tatsächliche Durchführung der vorgenannten Prozesse sicherzustellen, den Software-Entwickler dabei jedoch im Gegensatz zum Entwurfsmuster SSP besser zu entlasten, wird in [SN+05] mit dem *Intercepting Web Agent* (IWA) ein Entwurfsmuster vorgestellt, bei dem weder die bestehenden Anwendungen verändert, noch individuelle SSP erstellt werden müssen. Ebenso wie der SSP ermöglicht das Entwurfsmuster IWA die Erstellung eines unternehmensweiten Sicherheitskontexts im Sinne eines *Single Sign-On*. Die individuellen Benutzerdaten, die in den einzelnen Altsystemen hinterlegt sind, bleiben für die Zugriffsentscheidung irrelevant, die Zugriffsberechtigungen werden unternehmensweit und außerhalb der Altsysteme gepflegt und durch die Konzepte SSP beziehungsweise IWA außerhalb des Kontexts der Altsysteme überprüft und den eigentlichen Zugriff erst bei positiver Bestätigung gewährt. Im Gegensatz zum SSP wird der IWA direkt in die Nachrichten-*Queues* des *Application Server* eingehängt, der die Bereitstellung der Fachfunktionalität der Altsysteme an Webservice-Schnittstellen zuständig ist. Der IWA ist damit tief

genug verwurzelt, um sämtliche eingehende Nachrichten abfangen zu können. Nach einer Prüfung durch einen externen PDP lässt der IWA die Nachricht passieren oder er weist sie ab. Im Unterschied zum SSP findet die weitere Kommunikation direkt zwischen *Client* und der Zielanwendung statt. Der zentrale Unterschied zwischen einem IWA und einem SSP liegt darin, dass für jeden *Application Server*, der zur Bereitstellung von Webservices genutzt wird, genau ein IWA implementiert werden muss. Dieser besitzt eine sehr starke Kopplung zu diesem *Application Server*, da er sich tief in die Nachrichtenkanäle des Systems einhängt. Nach der Auswertung der Zugriffskontroll-Policies schaltet er die Nachricht entweder durch und ermöglicht eine direkte Kommunikationsbeziehung oder er bricht den Kontakt ab. Wie auch im Fall einer SSP-basierten Lösung wird der PDP außerhalb des *Application Server*, in dem der IWA installiert ist, platziert. Das bringt einige Vorteile mit sich. So können die Zugriffskontroll-Policies zentral verwaltet werden, auch wenn mehrere IWA auf mehreren *Web Server* installiert sind.

Bewertung

Ebenso wie der SSP ist der IWA als Zugriffskontroll-Komponente für den Bereich der Webservice-basierten Facharchitekturen einsetzbar (A1.1). Sicherheitslösungen auf Basis des IWA können als seriengefertigte Produkte (engl. *Commercial Of The Shelf*, COTS) von namhaften Herstellern bezogen werden. Als Referenzprodukte können CA eTrust SiteMinder und IBM Tivoli Access Manager genannt werden, die im nachfolgenden Kapitel 3.2.3 diskutiert werden. Eine Einbindung des Fachentwicklers ist nicht explizit vorgesehen (A1.2), die zugehörigen Zugriffskontroll-Policies werden durch Sicherheitsexperten nachgelagert entwickelt. Die Dienstschnittstellen des IWA hinsichtlich der Einhängung in den jeweiligen *Application Server* sind durch diesen fest vorgegeben, jedoch können die Schnittstellen zwischen IWA und Zugriffskontroll-Architektur theoretisch auf WSDL basieren (A2.1). Durch die feste Bindung zwischen IWA und *Application Server* ist die Portabilität des IWA stark eingeschränkt (A2.2). Die Integration bestehender Sicherheitsprodukte ist möglich, jedoch findet üblicherweise eine feste Verbindung zwischen IWA und Zugriffskontroll-Architektur statt, insbesondere im kommerziellen Produktumfeld. Sowohl die Bereiche der plattformunabhängigen Policy-Sprache (A3.1 und A3.2) sowie die modellgetriebene Entwicklung von Zugriffskontroll-Policies (A4.1) als auch die Benutzerdatenintegration (A5.1) werden durch das Entwurfsmuster nicht behandelt.

3.2.2 Sicherheitsstandards für Webservices

Organisationen wie das *World Wide Web Consortium* (W3C) oder die *Organization for the Advancement of Structured Information Standards* (OASIS) entwickeln unter anderem Sicherheitsstandards für den Webservice-Kontext. Die vorgestellten Standards lassen sich jedoch nicht nahtlos zu einer Zugriffskontroll-Architektur für Webservices zusammenfügen. Nachfolgend werden mit WS-Security, WS-Policy und WS-SecurityPolicy die drei für diese Arbeit relevantesten Spezifikationen vorgestellt und hinsichtlich ihrer Einsetzbarkeit im Rahmen dieser Arbeit bewertet.

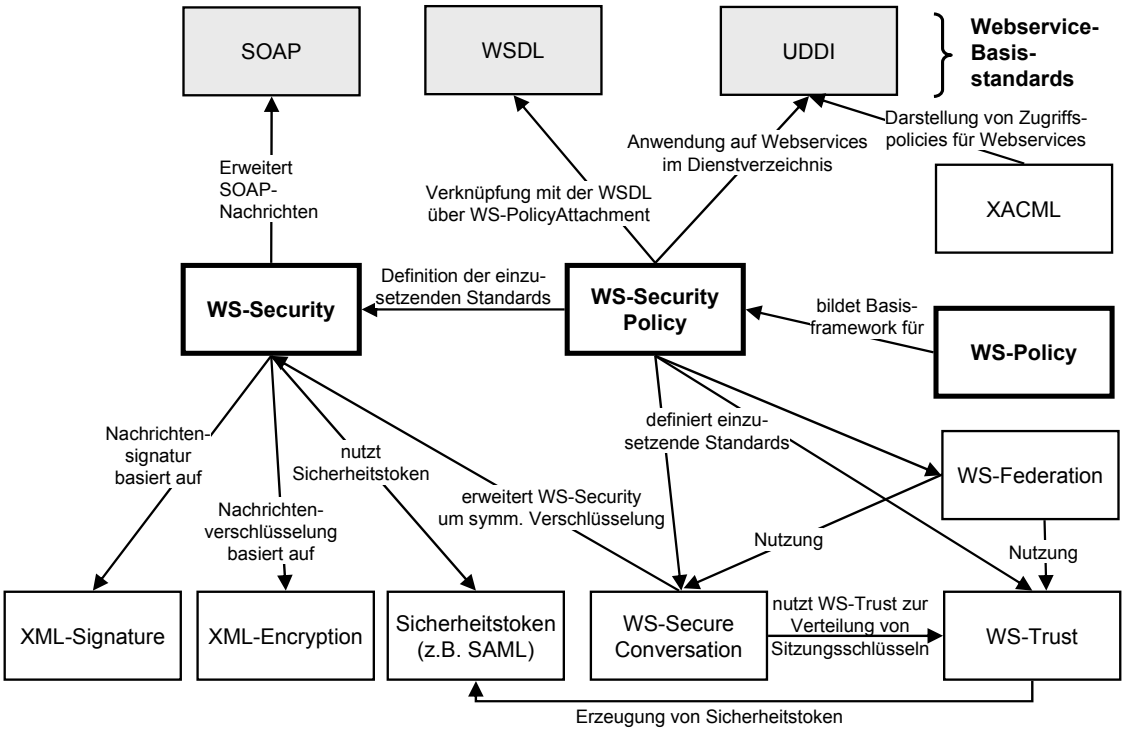


Abbildung 18: Zusammenhänge zwischen bestehenden Sicherheitsstandards

In Abbildung 18 werden die relevantesten Sicherheitsstandards im Webservice-Umfeld aufgeführt und in zueinander in Beziehung gesetzt. Ausgangspunkt bilden die drei Webservice-Basisstandards WSDL, SOAP und UDDI, die mit grauem Hintergrund am oberen Rand eingezeichnet sind. Die nachfolgende Diskussion verfolgt das Ziel, diese hinsichtlich ihrer Einsetzbarkeit in einer Zugriffskontroll-Architektur zu bewerten. Spezifikationen für das föderierte Identitätsmanagement wie die *Security Assertion Markup Language* (SAML) [OASIS-SAML2.0], WS-Trust [OASIS-WST1.3] und WS-Federation

[WSFED1.1] werden in Abbildung 18 nur der Vollständigkeit halber eingeordnet.

3.2.2.1 WS-Security

Einstiegspunkt in die Sicherheitsstandards im Webservice-Kontext bildet WS-Security [OASIS-WS-Sec1.1]. Der Name des Standards suggeriert zwar einen umfassenden Sicherheitsstandard, dies ist jedoch nicht der Fall. Eigentlich müsste der Standard eher „WS-SecureMessaging“ heißen, da es sich im Kern um eine Erweiterung des SOAP-basierten Nachrichtenaustausches zwischen Webservices handelt. Diese Erweiterung soll eine Absicherung des Nachrichtenaustausches ermöglichen und definiert eine syntaktische Standardisierung für das Signieren von SOAP-Nachrichten mittels XML-Signature [W3C-XML-Sig], das Verschlüsseln von SOAP-Nachrichten mittels XML-Encryption [W3C-XML-Enc] sowie das Anhängen von Sicherheitstoken an SOAP-Nachrichten. Sicherheitstoken können Benutzername/Passwort-Kombinationen, Zertifikate oder Binärtoken sein und werden für die Authentifizierung verwendet. WS-Security ermöglicht nur, einen verschlüsselten Kanal auf Nachrichtenebene zwischen Kommunikationspartnern aufzubauen, und stellt damit eine Lösung für die vertrauliche Übermittlung von Daten auf Nachrichtenebene dar, die mit den bekannten Technologien wie *Secure Sockets Layer* (SSL) beziehungsweise *Transport Layer Security* (TLS) nur auf Ebene der Transportschicht erbracht werden können.

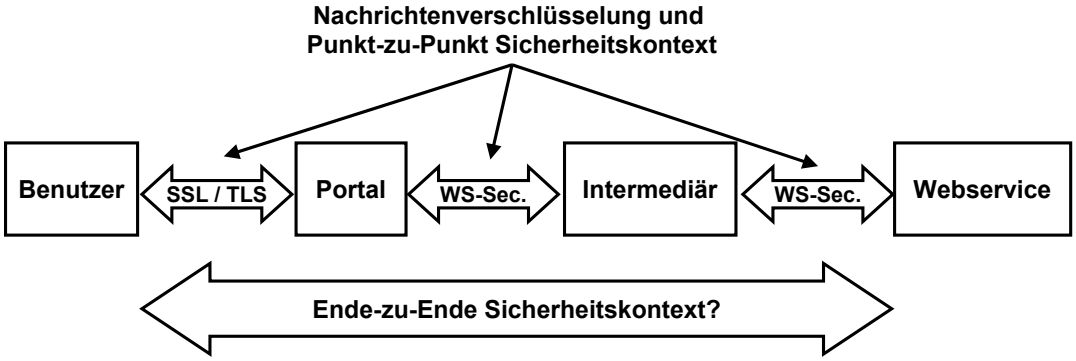


Abbildung 19: Einsatz und Grenzen von WS-Security

Im Webservice-Umfeld ist es spätestens bei der Verwendung von Intermediären wie beispielsweise aktiven Kommunikationsinfrastrukturen (engl. *Enterprise Service Bus*, ESB) nicht mehr möglich, die Kommunikation ausschließlich auf der Transportebene zu verschlüsseln [DJ+05]. Zusätzlich wird durch den Einsatz

von WS-Security ermöglicht, Ausschnitte einer Nachricht zu verschlüsseln oder durch den Einsatz unterschiedlicher Schlüssel Bereiche einer Gesamtnachricht gezielt nur für bestimmte Empfänger lesbar zu machen. Neben dem Bereich der Signatur und Verschlüsselung kommt als dritte Option der Einsatz von Sicherheitstoken zur Authentifizierung hinzu. Allerdings gilt es festzuhalten, dass die für die Nutzung von Sicherheitstoken notwendige Zugriffskontroll-Architektur ausdrücklich nicht im Standard thematisiert wird. Es wird auch nicht auf andere Spezifikationen verwiesen, die eine passende Software-Architektur definieren würde. Ebenfalls werden die zugehörigen Prozesse, beispielsweise die der Authentifizierung vorgelagerte Verteilung von Identitätsdaten, nicht von WS-Security definiert.

Bewertung

Die WS-Security-Spezifikation ist auf den Webservice-Kontext fixiert. Sie ist dort jedoch in beliebigen Fachdomänen einsetzbar (A1.1). Eine Einbindung des Fachentwicklers ist nicht vorgesehen (A1.2). Sicherheitsexperten konfigurieren die Nachrichtenverschlüsselung direkt auf den *Application Server* und damit für den Fachentwickler transparent. Die definierten Schnittstellen entsprechen den Webservice-Spezifikationen (A2.1). Die Einbindung in die *Application Server* findet in den Referenzimplementierungen über eine Einhängung in die Nachrichten-*Queues* statt. Dadurch bleiben die Schnittstellen WSDL-konform, jedoch reduziert dies die Portabilität der WS-Security-implementierenden Komponenten erheblich (A2.2). Die Integration von bestehenden Sicherheitsprodukten wird langfristig propagiert, jedoch ist man hierbei auf die Weiterentwicklung der Hersteller angewiesen (A2.3). Der Bereich der Autorisierung im Ende-zu-Ende Sicherheitskontext wird weitestgehend ausgeklammert (A3.1 und A3.2), was eine modellgetriebene Entwicklung von Zugriffskontroll-Policies verhindert (A4.1). Die Benutzerdatenintegration stellt für den Einsatz von WS-Security zwar eine wesentliche Voraussetzung dar, allerdings werden in den handelsüblichen Lösungen die Benutzerdaten unmittelbar an den *Application Server* angekoppelt (A5.1).

WS-Security ist ein relevanter Standard, wenn es um den Aspekt der Absicherung des Nachrichtenaustausches zwischen Webservices geht. Dies gilt insbesondere, wenn keine individuellen Nachrichten für jeden einzelnen Ziel-Webservice erstellt werden sollen, sondern eine Gesamtnachricht versendet wird, in der jeder Webservice nur die für ihn relevanten Teile sehen darf. Jedoch

besteht die Notwendigkeit, die für die Verschlüsselung notwendigen Zertifikate außerhalb und damit unabhängig von WS-Security vorab zu verteilen. Hierzu ist eine wesentliche Voraussetzung die Einführung einer unternehmensweiten *Public Key Infrastructure* (PKI) [AL99] lösen. Hierdurch lässt sich erreichen, dass Webservices als technische Kommunikationsendpunkte untereinander ihre öffentlichen Schlüssel (engl. *Public Key*) kennen, sich darauf aufbauend gegenseitig vertrauen und somit verschlüsselt miteinander kommunizieren können. Es fehlt jedoch eine passende Zugriffskontroll-Architektur, die eine unternehmensweise Ende-zu-Ende Authentifizierung von Benutzern gegenüber Webservices durchführt und nicht nur Punkt-zu-Punkt-basiert zwischen einzelnen Komponenten auf dem Nachrichtenweg arbeitet. Dies macht die Authentifizierungsfunktionalität von WS-Security zu einem ungeeigneten Kandidaten für die eigentliche Benutzerauthentifizierung. Sie sollte lediglich für Schutz des Dienstzugangspunktes eines Webservices eingesetzt werden, in dem Sinne, dass nur Nachrichten von durch die PKI als vertrauenswürdig eingestuften Webservices entgegen genommen werden.

3.2.2.2 WS-Policy und WS-SecurityPolicy

WS-Policy [W3C-WSP1.5] stellt ein Rahmenwerk dar, um generelle Anforderungen für die Interaktion mit einem Webservice strukturiert darzustellen. Ziel der Spezifikation ist es, Bedingungen für das Zusammenspiel zweier unmittelbar kommunizierender Webservices zu definieren. Als Beispiele für eine Interaktionsbedingung kann die Zusicherung gewisser Performanz im Sinne der Definition von Dienstqualität oder die Notwendigkeit des Einsatzes spezieller Transportprotokolle genannt werden. Für den Bereich der Sicherheit ist die darauf aufbauende Spezifikation WS-SecurityPolicy [WSSP1.1] relevanter. Diese ist im Gegensatz zu WS-Policy noch nicht von einer Standardisierungsorganisation wie dem W3C angenommen worden. Mit WS-SecurityPolicy besteht die Möglichkeit, neben den fachlichen Beschreibungen eines Diensts, die durch die WSDL vorgegeben werden, sicherheitstechnische Anforderungen zu definieren. Beispiele dafür sind die Verpflichtung, Nachrichten oder Teile davon zu signieren beziehungsweise zu verschlüsseln. Ebenfalls kann die Verwendung bestimmter Sicherheitstoken erzwungen werden oder ein Höchstalter für die Nachricht definiert werden. WS-SecurityPolicy ist dadurch unmittelbar mit dem vorgenannten Standards WS-Security verknüpft.

Bewertung

WS-Policy und WS-SecurityPolicy weisen keine über den Webservice-Kontext hinausgehenden Einschränkungen auf (A1.1). Die Einbindung des Fachentwicklers wird nicht thematisiert und würde mangels Werkzeugunterstützung zu Aufwand führen (A1.2). Die Schnittstellen lassen sich entsprechend der Webservice-Spezifikationen erweitern (A2.1). Eine Portabilität ist nicht bewertbar, da es sich im Wesentlichen um Konfigurationsparameter handelt (A2.2). Die Integration in bestehende Produkte wird angestrebt, wurde jedoch derzeit noch nicht umgesetzt (A2.3). Während WS-Policy Möglichkeiten bietet, allgemeine Policies zu formulieren, bleibt die konkrete Sprache zur Formulierung von Zugriffskontroll-Policies nicht spezifiziert (A3.1). Daher ist auch der Bereich der modellgetriebenen Policy-Entwicklung (A4.1) nicht bewertbar. Benutzerdatenintegration liegt ebenfalls nicht im Fokus von WS-(Security)Policy.

3.2.3 Das kommerzielle Umfeld

Auch im kommerziellen Umfeld gibt es Produkte, die Zugriffskontrolle auf Webanwendungen ermöglichen. Stellvertretend werden nachfolgend mit CA eTrust SiteMinder (vormals Netegrity SiteMinder) und IBM Tivoli Access Manager die Architekturen zweier führenden Sicherheitsprodukte vorgestellt.

3.2.3.1 CA eTrust SiteMinder

CA eTrust SiteMinder, im Folgenden kurz SiteMinder genannt, bildet eine Sicherheits- und Managementplattform für den Schutz unternehmensweiter Webanwendungen mittels einer zentralen Sicherheitsinfrastruktur zur Durchführung von Benutzerauthentifizierung und Zugriffskontrolle [CA-SM-DB, CA-SM-TWP]. SiteMinder wurde für den unternehmensweiten Schutz von Webanwendungen entwickelt und setzt im Kern auf einen zentralisierten *Policy Decision Point* (PDP), den SiteMinder Policy Server und eine Vielzahl an *Policy Enforcement Points* (PEP), die in jedem *Web Server* mit Zugriffskontroll-Funktion installiert werden. SiteMinder setzt auf ein zusätzlich einzusetzendes *Firewall*-Konzept, um externe Vorabfilterungen zu ermöglichen. SiteMinder gibt zwei Wege vor, um bestehende Webanwendungen mit Zugriffskontrolle zu versehen. Bei Anwendungen, die direkt auf einem *Web Server* ablaufen, wird diesem *Web Server* ein sogenannter SiteMinder Webagent als Modul hinzugefügt. Dieses

Modul klinkt sich in die internen Kommunikationswege des *Web Server* ein und führt eine Anfrage an den SiteMinder Policy Server durch. Wenn von diesem eine Zugriffsbestätigung übermittelt wird, wird die eigentliche Fachfunktionalität auf dem *Web Server* ausgeführt.

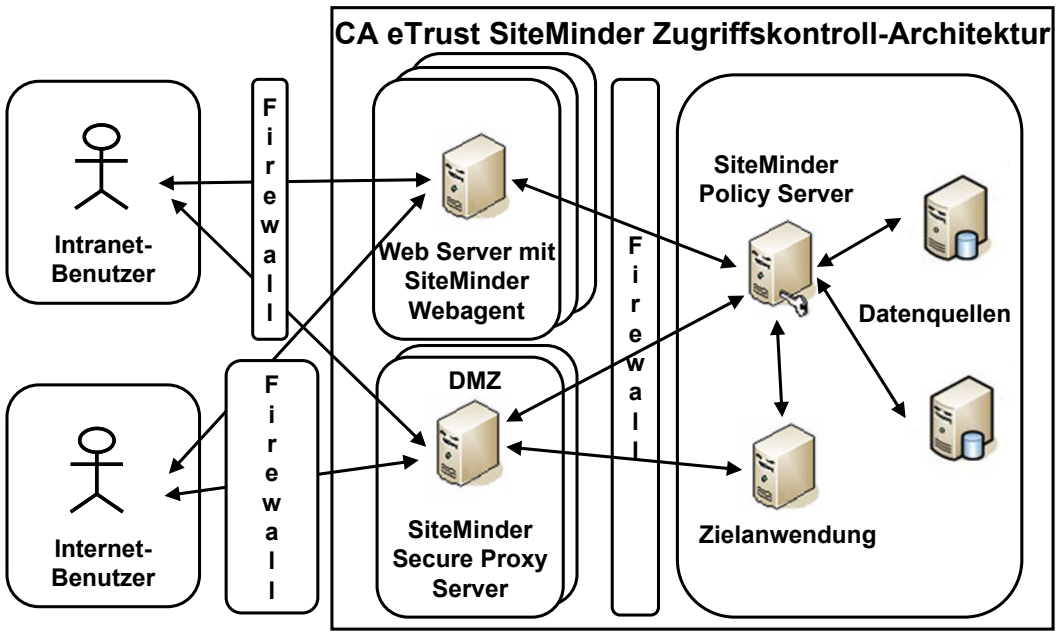


Abbildung 20: Architektur von CA eTrust SiteMinder

Die zweite Alternative ermöglicht es, mit dem SiteMinder Secure Proxy Server lediglich eine Stellvertreterkomponente in einer demilitarisierten Zone (DMZ) aufzustellen. Diese führt ebenfalls eine Berechtigungsanfrage an den SiteMinder Policy Server durch, führt bei positiver Rückmeldung dann jedoch die Fachfunktionalität nicht lokal aus, sondern übergibt die Anfrage in einen geschützteren, internen Sicherheitsbereich. Abbildung 20 zeigt im oberen Teil die Realisierungsform mittels SiteMinder Webagent, im unteren Teil die mittels des SiteMinder Secure Proxy Server. Die Zugriffskontroll-Anfrage ist mehrstufig: Zunächst wird überprüft, ob der Benutzer authentifiziert ist. Falls dies der Fall ist, wurde dem Benutzer ein sitzungsbezogenes und damit zeitlich limitiertes *Cookie* übermittelt, welches er über seinen Browser automatisch und damit transparent für ihn überträgt. Im nächsten Schritt wird überprüft, ob der (authentifizierte) Eigentümer des *Cookies* berechtigt ist, auf die durch eine URI charakterisierte Webressource zuzugreifen. Die Antwort vom SiteMinder Policy Server an den SiteMinder Webagent ist ein boolescher Wert.

Bewertung

Die SiteMinder-Plattform ist eine häufig genutzte Zugriffskontroll-Lösung im industriellen Umfeld und wurde zur Absicherung klassischer Webanwendungen entwickelt. Sie erfüllt im Grundsatz die Anforderung nach Allgemeingültigkeit (A1.1), adressiert jedoch von ihrem Zugriffskontroll-Modell traditionelle Webanwendungen. Das wird unter anderem auch daran deutlich, dass die zu schützenden Objekte oder Verzeichnisbäume nur durch ihre URI bestimmt werden. Während dieser Ansatz für traditionelle Webanwendungen genügt, ist für Zugriffskontrolle im Webservice-Kontext die reine Betrachtung der URI nicht immer ausreichend. Es ist beispielsweise mit SiteMinder nicht möglich, auf Parameter eines Dienstaufrufes zuzugreifen, was im Kontext grobgranularerer Webservices als zu schützende Objekte wichtig ist. SiteMinder unterstützt derzeit den Schutz von Webservices nur über *Workarounds*, jedoch ist eine Unterstützung für die Zukunft angekündigt. Der Fachentwickler wird üblicherweise in die Erstellung der Zugriffskontroll-Policies nur marginal eingebunden (A1.2). Diese werden in der Praxis nachgelagert durch Sicherheitsexperten entwickelt. Somit ist der Fachentwickler zwar nicht belastet, jedoch ist die fehlende Verbindung zwischen Fachentwickler, der die Zugriffsrechte seiner Anwendung kennt, und dem Sicherheitsexperten, der sie umsetzen muss, problematisch. Die Dienstschnittstelle, insbesondere die zum SiteMinder Policy Server ist herstellerspezifisch und nicht in WSDL ausgeprägt (A2.1). Dies ist für den Fall unproblematisch, dass für den zu schützenden *Application Server* durch den Hersteller passende SiteMinder Webagents als *Policy Enforcement Points* geliefert werden. Ist das nicht der Fall, entsteht ein überdimensionaler Entwicklungsaufwand, wie in [St06b] gezeigt wurde. Eine Portabilität wie in Anforderung A2.2 gefordert, ist daher nicht gegeben. Hinsichtlich der Integration bestehender Produkte hält sich CA, wie auch andere Produkthersteller, bedeckt. Es ist lediglich vorgesehen, Datenquellen von Fremdherstellern einzubinden, nicht jedoch Anwendungslogik. Dies schränkt die geforderte Integrationsfähigkeit (A2.3) stark ein. Hinsichtlich der Policy-Sprache ist problematisch, dass die Eigenheiten einer dienstorientierten Webservice-Architektur nicht adäquat abgebildet werden können. Eine Plattformunabhängigkeit der Sprache selbst, also konkret eine Herstellerunabhängigkeit, ist aus naheliegenden Gründen nicht erwünscht. Auch die Einbindung des Fachentwicklers ist nicht vorgesehen. Daher ist die Betrachtung der Anforderung A4.1 nach expliziten Policy-Transformationen bis in das konkrete Produkt hinein obsolet. Die Forderung nach expliziter Darstellung der Semantik der Benutzereigenschaften und -attribute wird durch SiteMinder nicht realisiert

(A5.1). Es findet eine harte Verdrahtung zwischen Attributen eines Benutzerverzeichnisses und einer Zugriffskontroll-Policy statt.

3.2.3.2 IBM Tivoli Access Manager

IBM bietet mit dem Produktportfolio „Sicherheit“ aus der Tivoli-Reihe aufeinander abgestimmte Werkzeuge. Durch die klare Auftrennung in einzelne Produkte für die Bereiche Identitätsmanagement und Zugriffskontrolle ist für die Betrachtung der Zugriffskontroll-Architektur der Tivoli Access Manager relevant.

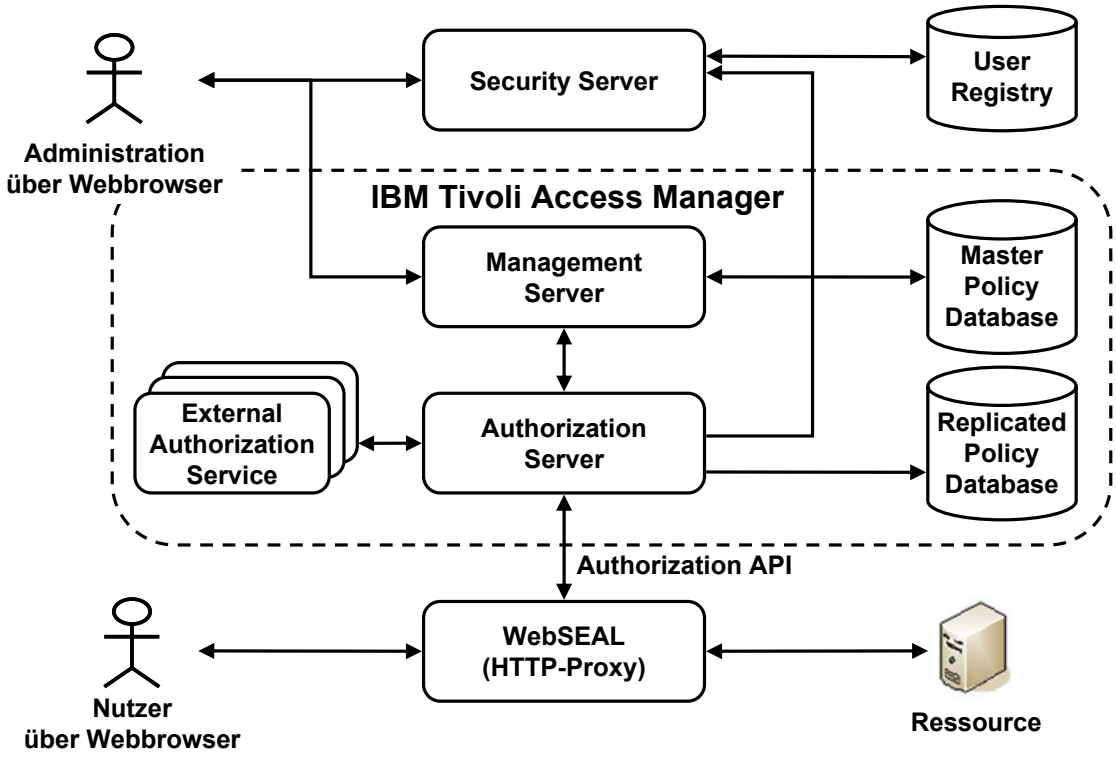


Abbildung 21: Architektur des IBM Tivoli Access Manager

In [Ka03] wird ein Überblick über die Architektur des IBM Tivoli Access Manager, die in Abbildung 21 dargestellt wird. Den zu schützenden Anwendungen wird mit WebSEAL ein HTTP-*Proxy* vorgeschaltet, der für die Aspekte Lastverteilung und Zugriffskontrolle zuständig ist. Er stellt für den Benutzer den Zugangspunkt für die jeweiligen Anwendungen dar und wird in einer demilitarisierten Zone (DMZ) positioniert. Die zu schützenden Anwendungen werden dann in einem geschützten Bereich aufgestellt, der von außen nur über die DMZ erreichbar ist. Der WebSEAL stellt bei jedem Aufruf eine Anfrage an den Authorization Server, der anhand der angeschlossenen Policy-Database und

gegebenenfalls weiteren, externen Autorisierungsquellen eine Zugriffsentscheidung trifft. Die Anfrage wird über eine öffentlich verfügbare Authorizion API abgewickelt, was einen Austausch des WebSEAL als PEP prinzipiell ermöglicht. Die Policies werden über ein Webportal durch den Management Server in die zentrale Master Policy Database geschrieben und regelmäßig repliziert. Die Benutzerdaten werden über gesonderte Produkte wie dem Tivoli Directory Server und dem Tivoli Identity Manager verwaltet. Diese Produkte werden mit dem Begriff „Security Server“ zusammengefasst, über den auch der Tivoli Access Manager auf die Benutzerdaten zugreift. Zur Erhöhung der Performanz erlaubt es der Tivoli Access Manager, den Authorization Server zu replizieren, und ihn beispielsweise direkt auf dem *Application Server*, auf dem auch der WebSEAL läuft, zu installieren. Die Policy-Datenbank wird dann ebenfalls repliziert und auf den für diese Anwendung notwendigen Regelsatz begrenzt. Wie auch SiteMinder stammt der Tivoli Access Manager aus der Zeit der traditionellen Webanwendungen. Die Policies lassen sich ebenfalls nur in URI-Mustern definieren, wobei Tivoli versucht, auch die Parameter eines auf HTTP-GET basierten Formularaufrufes mit zu berücksichtigen.

Bewertung

Analog wie für die in 3.2.3.1 gegebene Bewertung des direkten Konkurrenzprodukts SiteMinder gilt auch für den Tivoli Access Manager, dass er zwar allgemeingültig einsetzbar ist, jedoch nicht für den Einsatz in Webservice-Architekturen entwickelt wurde (A1.1). Dadurch ergeben sich Probleme aus architektureller Sicht und hinsichtlich des Zugriffskontroll-Modells. Auch hier findet keine explizite Einbindung des Fachentwicklers im Software-Entwicklungsprozess statt (A1.2, A3.2). Die Dienstschnittstellen zum Authorization Server sind in einer API-Spezifikation veröffentlicht; sie sind jedoch herstellerspezifisch. Sie sind nicht in WSDL beschrieben (A2.1). Der WebSEAL als PEP kann durch seine *Proxy*-Eigenschaft gut ausgetauscht werden, ist damit also hinreichend portabel (A2.2). Hinsichtlich der Integrationsfähigkeit von Fremdprodukten gilt ähnliches wie zu SiteMinder: Es sind lediglich Ankopplungen von Fremdprodukten auf Ebene der Datenquellen vorgesehen. Die konkreten Policies sind Tivoli-spezifisch (A3.1) und ein modellgetriebener Ansatz wird nicht thematisiert (A4.1). Die explizite Behandlung von Semantik hinsichtlich der Benutzerverzeichnisse wird ebenfalls nicht thematisiert (A5.1).

3.3 Zugriffskontroll-Modelle und Policy-Sprachen

Um eine Zugriffskontroll-Entscheidung treffen zu können, wird neben einer geeigneten Zugriffskontroll-Architektur auch ein Zugriffskontroll-Modell benötigt, auf dessen Basis entsprechende Zugriffskontroll-Policies erstellt werden können. Dieses Zugriffskontroll-Modell muss die Spezifika der zu schützenden Objekte angemessen berücksichtigen können. Nachfolgend wird der Stand der Forschung in diesem Bereich vorgestellt, wobei drei Modelle betrachtet werden, die eine Nähe zu Webservice-orientierten Architekturen aufweisen.

3.3.1 RBAC-basierte Zugriffskontroll-Modelle

Während die Urform der rollenbasierten Zugriffskontrolle (engl. *Role-Based Access Control*, RBAC), die in Kapitel 2.2.2 eingeführt wurde, unabhängig von der Architektur der zu schützenden Software-Umgebung ist, finden sich in der Literatur Weiterentwicklungen und Ausprägungen für konkrete Plattformen. Nachfolgend wird mit dem Zugriffskontroll-Modell aus [Lo03] ein wissenschaftlicher Ansatz vorgestellt, der auch im weiteren Verlauf dieser Arbeit für den Bereich der modellgetriebenen Entwicklung von Zugriffskontroll-Policies von Relevanz ist. Mit der Entwicklung von SecureUML aus [Lo03] wird das Ziel verfolgt, ein Zugriffskontroll-Modell für aktuelle Software-Architekturen zu erstellen. Es ist auf dem RBAC-Basismodell aus [SC+96] aufgebaut, das eine Verschattung einzelner Subjekte und damit handelnder Benutzer hinter einer Rolle durchführt. Abbildung 22 greift das dort vorgestellte Modell auf, das zunächst als konzeptionelles Metamodell dargestellt wird, das die Mengen und Relationen einführt. Den Kern dieses Modells bildet das Element „Berechtigung“, das eine Rolle mit einer Aktion auf einer Ressource verbindet. Durch die definierten Kardinalitäten ist sichergestellt, dass eine Berechtigung wiederverwendbar ist, jedoch stets mindestens eine Rolle mit einer Aktion verbindet. Abgesehen von der Darstellung in Form eines formalisierten UML-Klassenmodells stellt dieser Zusammenhang ebenso wie der linke Teil des Modells, der die Zusammenhänge zwischen Rollen, Subjekten, Benutzern und Gruppen darstellt, keine echte Erweiterung der bekannten RBAC-Modelle dar. Interessant ist der Aspekt, eine Berechtigung mit weiteren Einschränkungen zu versehen. Als mögliche Einschränkung werden Umgebungszustände wie Tageszeit oder Systemauslastung genannt. Dieser Bereich wird jedoch nicht weitergehend vertieft.

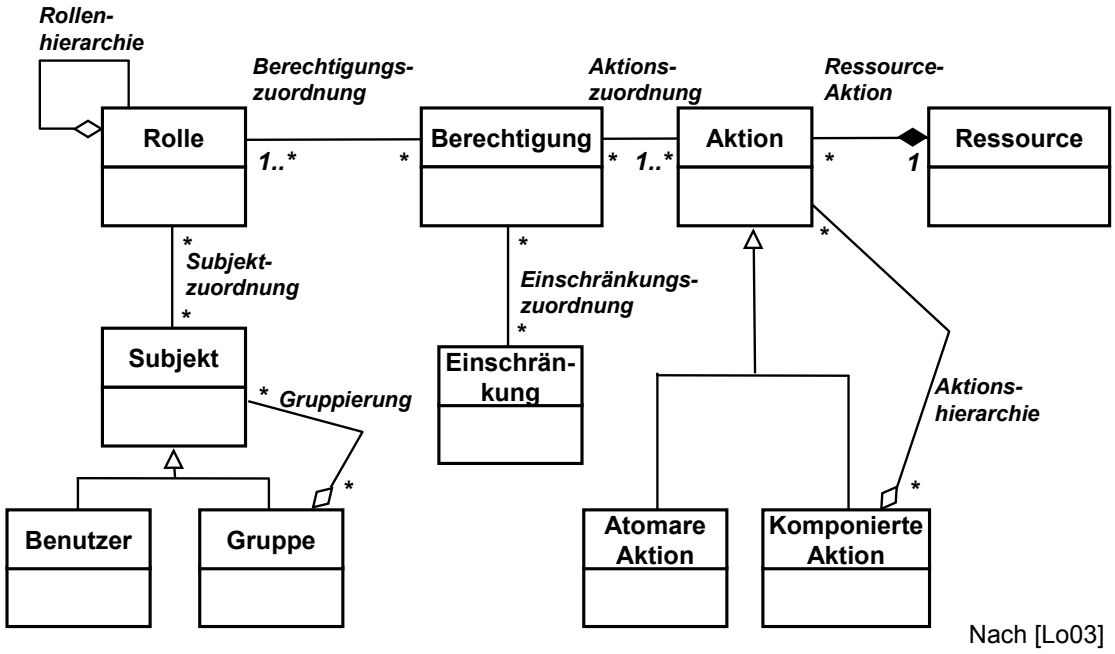


Abbildung 22: Metamodell von SecureUML

Während die Aktion zwar immer noch den klassischen Aktionsparadigmen wie „lesen“ oder „schreiben“ zugeordnet wird, wird jedoch die Möglichkeit gegeben, komponierte Aktionen durch die Einführung einer Aktionshierarchie zu beschreiben. Die Berechtigungen einer komponierten Aktion wirken sich dabei unmittelbar auf die Berechtigungen der zugrundeliegenden atomaren Aktionen aus. Eine weitere Eigenschaft, die nicht aus dem dargestellten Modell hervorgeht, sondern lediglich textuell aufgeführt wird, ist die Möglichkeit des Zugriffs auf den Namen des Aufrufers in den Einschränkungen, was eine Erweiterung von RBAC darstellt.

Bewertung

SecureUML ist hinreichend allgemeingültig, um es als Zugriffskontroll-Modell für Webservices einzusetzen (A1.1). Die Einbindung des Fachentwicklers wird explizit gefordert (A1.2). Die Zugriffskontroll-Architektur steht nicht im Vordergrund, es wird die Sicherheitsarchitektur des jeweiligen *Application Server* verwendet. Daraus ergeben sich Hürden hinsichtlich der im Webservice-Kontext üblichen, *Application Server*-übergreifenden Webservice-Kompositionen (A2.1). Die Portabilität der Software-Lösung wird dahingehend adressiert, dass eine jeweils passende Lösung aus den plattformunabhängigen Modellen erzeugt wird (A2.2). Die Integration bestehender Sicherheits-Architekturen steht im Fokus, da keine eigenen Sicherheitskomponenten entwickelt werden (A2.3), jedoch werden

nur die *Application Server*-internen Sicherheitskomponenten betrachtet. Die Policies von SecureUML sind zwar plattformunabhängig (A3.1) und eine Einbindung in den Software-Entwicklungsprozess wird ebenfalls gegeben (A3.2), jedoch setzt der propagierte Ansatz sehr spät im Software-Entwicklungsprozess an. Es werden lediglich die bereits spezifizierten Komponenten mit sehr technologienahen Zugriffskontroll-Informationen angereichert. Die Policy-Transformationen werden nur für *Application Server* eingeführt, nicht jedoch für die üblicherweise eingesetzten, kommerziellen Produktlösungen, deren Policy-Sprachen deutlich aufwendigere Transformationen benötigen. Die Anforderungen an die modellgetriebene Policy-Entwicklung werden in Kapitel 3.4.1 weitergehend diskutiert.

Positiv an SecureUML ist herauszustellen, dass die Erweiterung des Zugriffskontroll-Modells auf der Grundlage des bewährten RBAC-Basismodells stattfindet. Ebenso ist die formale Darstellung des Zugriffskontroll-Modells von Vorteil, insbesondere wenn sie die Grundlage einer modellgetriebenen Entwicklung von Zugriffskontroll-Policies bilden soll. Nachteilig erweist sich, dass die Kompositionsbildung, die in Webservice-orientierten Architekturen eine entscheidende Rolle spielt, auf zusammengesetzte Aktionen und nicht auf zusammengesetzte Ressourcen abgebildet wird. Aus der Perspektive der dienstorientierten Architekturen sind die einzelnen Aufrufe, also die Aktionen aus der Sicht von SecureUML, immer noch atomar. Die Komposition erzeugt eine neue Ressource, die über einzelne Aufrufe einen komplexeren Kontext abbildet. Dieser Zusammenhang lässt sich in SecureUML nicht darstellen. Ebenso ist es nur möglich, bei der Auswertung der Zugriffsentscheidung auf die Rollen eines handelnden Subjekts zuzugreifen. Auch die Erweiterung um den Namen des Aufrufers gibt noch nicht die Möglichkeit, feingranular einzelne Attribute seiner digitalen Identität gegen die Parameter eines Webservice-Aufrufs zu vergleichen.

3.3.2 ABAC-basierte Zugriffskontroll-Modelle

Während bei RBAC-basierten Zugriffskontroll-Modellen durch die Einführung des Rollenkonzepts lediglich eine Indirektionsstufe zwischen Subjekten und Objekten geschaffen wurde, werden die Zugriffskontroll-Policies üblicherweise noch immer in Form von Zugriffskontroll-Listen direkt an die zu schützenden Objekte gebunden. Das bedeutet, dass beim Zugriffsversuch auf ein geschütztes Objekt ein unmittelbar zugehöriges Regelwerk ausgewertet wird. Dies bedeutet wiederum, dass bei der Spezifikation der Zugriffskontroll-Policies die Kenntnis

über die möglichen Rollen der potenziell zugreifenden Subjekte vorhanden sein muss. Während im kommerziellen Produktumfeld RBAC-basierte Sicherheitslösungen dem Stand der Technik entsprechen, findet im wissenschaftlichen Kontext eine Diskussion statt, ob Veränderungen in der Software-Architektur, im Speziellen die Fokussierung auf dienstorientierte Architekturen, ebenso eine Veränderung in den Zugriffskontroll-Modellen notwendig macht. Durch die Auftrennung existierender Software-Systeme in gekapselte, fachlich relevante Dienste, die lose miteinander verbunden sind und von einer Vielzahl von Benutzern und anderen Diensten genutzt werden, stellen viele Autoren starke Einschränkungen bei der Nutzung rollenbasierter Modelle fest. Dadurch, dass die Zugriffsberechtigungen ausschließlich an den vordefinierten Rollen festgemacht werden können und die Nutzer vorab mit diesen Rollen assoziiert werden müssen, ergibt sich durch die Vervielfachung der zu schützenden Objekte das Problem der exponentiell zunehmenden Anzahl an Rollen. Zur Überwindung dieses Zustands wird in [YT05] die Erstellung eines attributbasierten Zugriffskontroll-Modells favorisiert. Als Handlungsbedarf wird hier explizit die steigende Anzahl an Subjekten und Objekten in einer dienstorientierten Architektur festgemacht. Neben der Erweiterung des RBAC-Modells um allgemeine Attribute bei der Berücksichtigung der Zugriffsentscheidung wird eine Abtrennung der Zugriffskontroll-Policies von den zu schützenden Objekten angestrebt. Die Policies sollen unabhängig von den konkreten Objekten formuliert werden und die Anwendbarkeit zum Zugriffszeitpunkt überprüft werden. Während bei RBAC die Zugriffskontroll-Policies üblicherweise direkt an die Objekte geknüpft sind, führt dieser Ansatz zu einer weiteren Indirektion und damit einer Unabhängigkeit der Elemente Subjekt, Objekt und Zugriffskontroll-Policy. Um eine attributbasierte Zugriffskontrolle für den Webservice-Kontext zu entwickeln, werden in [YT05] drei Arten von Attributen als Basismenge zur Berechnung der Zugriffsentscheidung eingeführt: Subjektattribute beschreiben die Eigenschaften eines Subjekts. Die Attribute sind dabei nicht auf zugriffsrelevante Eigenschaften eines Subjekts beschränkt. Die Subjektattribute lassen sich ideal mit dem Ansatz einer digitalen Identität, wie sie in Kapitel 2.2.3 eingeführt wurde, in Deckung bringen. Die digitale Identität stellt dabei die Summe aller Attribute im Sinne von Schlüssel/Wert-Paaren dar. Alle diese Wertpaare charakterisieren ein Subjekt und können zur Berechnung der Zugriffskontroll-Entscheidung herangezogen werden. Eine Verknüpfung mit rollenbasierten Ansätzen ist ebenfalls möglich, da eine Rolle auch im Sinne eines Schlüssel/Wert-Paars dargestellt werden kann. Des Weiteren werden Objektattribute eingeführt. Objektattribute stellen eine Form von Metainformation über ein Objekt dar, die es hinreichend charakteri-

siert, um eine Zugriffsentscheidung herbeizuführen. Die Beschreibung der Objekte findet dabei unabhängig von den Subjekten oder den Zugriffskontroll-Policies statt. Als dritte Kategorie von Attributen werden umgebungsbezogene Attribute eingeführt. Dieser Kategorie werden alle Attribute zugeordnet, die keinen direkten Bezug zu Subjekten und Objekten haben. Beispiele für solche Attribute sind die aktuelle Uhrzeit oder die Netzauslastung. Die Zugriffskontroll-Policies setzen nun auf diesen Attributen auf. Ziel ist es, eine Grundmenge an allgemeinen Policies zu entwickeln, die für die Durchführung der Zugriffsentscheidung auf einer Vielzahl an Objekten einsetzbar ist. Die Policies an sich lassen sich als boolesche Funktion verstehen, die ausschließlich auf Subjekt-, Objekt-, und Umgebungsattributen operiert. Gerade bei einer großen Zahl an Objekten mit ähnlichen Zugriffseinschränkungen lassen sich Vorteile erzielen, was die Menge an Zugriffskontroll-Policies betrifft. Vorteile gegenüber klassischen RBAC-basierten Modellen lassen sich bei der Kombination orthogonaler Eigenschaften eines Subjekts erkennen. Beispielhaft kann das Ausleihen von Filmen in einer Videothek mit Pauschaltarif betrachtet werden. Neben der Definition unterschiedlicher Altersklassen, die in einer Rollenzuweisung resultieren würde, kann es auch unterschiedliche Kundenklassen geben: Normalkunden und Premiumkunden, die gewisse Vorzüge genießen. Auch diese Eigenschaft würde im RBAC-Modell in einer Rolle abgebildet. Da die Kombination beider Eigenschaften beim Ausleihvorgang geprüft werden müssen, würde dies zur Erstellung neuer Rollen führen, die das Produkt aller bestehenden Einzelrollen darstellen. Dies führt bereits bei wenigen Kombinationen zu Skalierungsproblemen.

Bewertung

Das vorgestellte ABAC-Modell wurde explizit für den Einsatz in Webservice-basierten Architekturen geschaffen (A1.1), jedoch wurde die Einbindung des Fachentwicklers zur Policy-Erstellung nicht thematisiert (A1.2). Da keine Aussagen hinsichtlich der einzusetzenden Software-Komponenten getroffen werden, ist ein pauschales Urteil über die Interoperabilität der Schnittstellen (A2.1) und der Portabilität der Komponenten (A2.2) nicht möglich. Die Entwicklung WSDL-konformer Schnittstellen ist jedoch nicht grundsätzlich ausgeschlossen. Die Integration bestehender Sicherheitsprodukte (A2.3) steht nicht im Fokus, ebenso wenig wie modellgetriebene Policy-Entwicklung (A4.1) und die Benutzerdatenintegration (A5.1). Es steht das Ziel im Vordergrund, durch die starke Entkopplung von Subjekten und Objekten sehr allgemeine und damit

plattformunabhängige Zugriffskontroll-Policies formulieren zu können (A3.1). Diese sollen jedoch nicht transformiert und damit konkretisiert werden, sondern in allgemeiner Form zur Anwendung gebracht werden. Dies stellt große Herausforderungen an die konkret einzusetzende Policy-Sprache.

Der grundsätzliche Ansatz, neben einem starren Rollenkonzept alle Eigenschaften eines Subjekts und damit seiner digitalen Identität für die Zugriffskontroll-Entscheidung zu berücksichtigen, ist positiv zu bewerten. Insbesondere auch vor dem Hintergrund, dass die Rollen in den Anwendungssystemen oft überhand nehmen und das ursprüngliche Konzept einer direkten Verknüpfung von Rollen aus der Geschäftsprozessmodellierung und den systemtechnisch bedingten Rollen bei weitem nicht mehr gegeben ist. Die Berücksichtigung von Attributen, die den Umgebungszustand charakterisieren, schafft ebenfalls Erweiterungsmöglichkeiten. Der Ansatz, durch die Einführung von Objektattributen die Zugriffsberechtigungen weiter von den Objekten abzukoppeln, scheint vordergründig mehr Flexibilität zu schaffen, die Notwendigkeit muss jedoch in Frage gestellt werden. Eine wirklicher Mehrwert würde entstehen, wenn es gelingen würde, durch eine (geringe) Anzahl an objektübergreifenden Zugriffskontroll-Policies eine passende und feingranulare Zugriffskontrolle für alle zu schützenden Objekte zu ermöglichen. Policies würden anhand allgemeiner Unternehmensziele spezifiziert und dann allgemeingültig zur Umsetzung gebracht. Das kann jedoch höchstens eine grobgranulare Zugriffskontrolle ermöglichen wie beispielsweise „Zugriffe von extern sind nur für Vorstandsmitglieder möglich“. Feingranulare Zugriffskontrolle muss gerade in Webservice-orientierten Architekturen die einzelne zu schützende Webservice-Operation berücksichtigen. Diese wird unter anderem über ihre höchst individuelle Methodensignatur charakterisiert. In wie weit eine allgemein formulierte und damit übergreifend gültige Policy hier sinnvolle Zugriffskontroll-Entscheidungen ermöglichen können, ist in Frage zu stellen. Wenn man durch diese Betrachtungen an dem Punkt angelangt ist, dass eine Zugriffskontroll-Policy einen unmittelbaren Bezug zu einem zu schützenden Objekt haben sollte, kann die Komplexität der indirekt verknüpften Policies eingespart werden. Auch die Definition einfacher Objekt-Eigenschaften in Form von Attributen scheint nicht ausreichend, um eine wirkungsvolle Zugriffskontrolle zu ermöglichen. Durch die Definition und Berücksichtigung von Objektattribute wird eine rein statische Sicht auf ein Objekt eingenommen. Die dynamischen Aspekte wie die konkreten Parameter, die bei einem Webservice-Aufruf übermittelt werden, bleiben außerhalb der Betrachtung. Gerade durch die Fokussierung auf Webservice-Plattformen, bei denen eine Vergröberung der Ob-

jektgranularität stattfindet, stellt diese fehlende Betrachtung einen wichtigen Nachteil dar.

Zusammenfassend lässt sich festhalten, dass die Erweiterung von Rollen auf die Gesamtmenge der Subjektattribute sinnvoll ist, die Verknüpfung von Zugriffskontroll-Policies mit zu schützenden Objekten über Objektattribute eine unnötige Indirektion und damit Komplexität darstellt, insbesondere unter fehlender Berücksichtigung der dynamischen Aspekte wie der konkreten Webservice-Parameter.

3.3.3 Die Policy-Sprache XACML

Die durch die OASIS standardisierte *Extensible Access Control Markup Language* (XACML) [OASIS-XACML2.0] spezifiziert neben einer Zugriffskontroll-Architektur, die in Kapitel 3.2.1.2 behandelt wurde, eine flexible Sprache für die Darstellung von Zugriffskontroll-Policies. Die Policies sind dabei nicht auf eine bestimme Domäne beschränkt und werden auch für den Einsatz in Webservice-orientierten Architekturen diskutiert [Ta05, Ho07]. XACML ist eine XML-basierte Sprache, deren Elemente und ihre Zusammenhänge in Abbildung 23 aufgezeigt und nachfolgend erläutert werden. Das Wurzelelement der Sprache ist die `Policy`. Im Zuge einer Hierarchiebildung kann es einem `PolicySet` zugeordnet werden. Jede `Policy` besteht aus mindestens einer `Rule` (dt. Regel), die das Basiselement eines XACML-Artefakts darstellt. Jede `Rule` ist mit genau einem `Effect` verbunden, der definiert, welches Resultat bei positiver Auswertung der Regel gemeldet wird. Damit besteht die Möglichkeit, sowohl positive als auch negative Zugriffsberechtigungen zu formulieren. Die wichtigsten Effekte sind `Permit` und `Deny`. In jeder `Rule` wird üblicherweise ein `Target` (dt. Ziel) angegeben, auf das sich die Regel bezieht, sonst bezieht sich die Regel auf sämtliche Anfragen. Ein `Target` besteht aus einem 3-Tupel (`Subject`, `Action`, `Resource`), durch das spezifiziert wird, welche Operationen (`Action`) welcher Entitäten (`Subject`) auf welchen Objekten (`Resource`) durch die Regel kontrolliert werden sollen. Zusätzlich gibt es weitere Möglichkeiten, über die Sprachkonstrukte `Condition` und `Obligation` weitere Einschränkungen oder zusätzliche Ausführungsaktionen zu definieren. Für jede `Policy` muss ein `RuleCombiningAlgorithm` spezifiziert werden, der bestimmt, welcher `Effect` als Gesamtresultat ausgeliefert wird, für den Fall, dass mehr als eine `Rule` auswertbar ist. In der XACML-Spezifikation sind

bereits einige Algorithmen vordefiniert: Beispielsweise führt `deny-overrides` dazu, dass eine `Policy`, bei der mindestens eine zutreffende `Rule` als `Effect` den Wert `Deny` hat, als Gesamtresultat ebenfalls `Deny` liefert; beim Algorithmus `first-applicable` wird die Auswertung der `Policy` nach der ersten `passenden Rule` beendet und deren `Effect` als Ergebnis der Policy-Auswertung zurückgegeben.

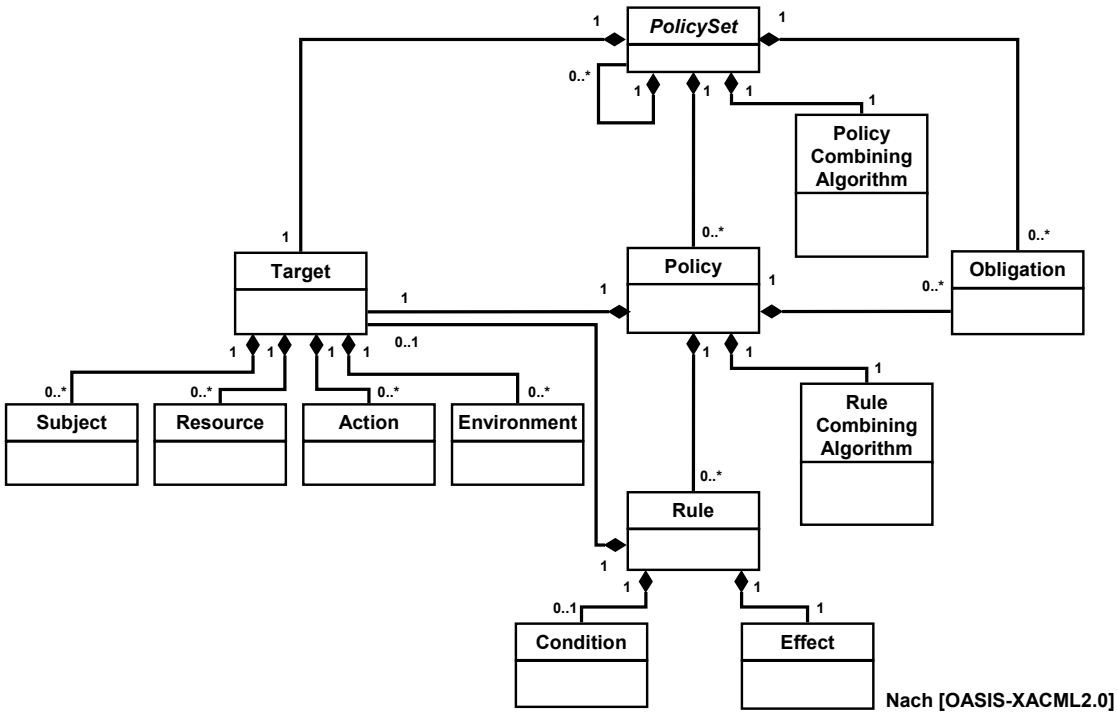


Abbildung 23: XACML-Zugriffskontroll-Modell

Die Vereinigungsmenge der `Targets` der `Rules` kann optional als `Target` der `Policy` angegeben werden; hierdurch kann effizienter entschieden werden, ob eine `Policy` überhaupt für eine Anfrage relevant ist. Um eine weitere Hierarchiebildung zu ermöglichen, wurde das Element `PolicySet` definiert, das mehrere `Policy`-Elemente zusammenfassen kann. Für die Steuerung der Auswertung der in einem `PolicySet` enthaltenen `Policies` muss analog zum `RuleCombiningAlgorithm` ein `PolicyCombiningAlgorithm` angegeben werden; hierfür stehen ebenfalls einige vorgefertigte Algorithmen zur Auswahl.

Bewertung

Die Bewertung der in der XACML-Spezifikation ebenfalls vorgestellten Zugriffskontroll-Architektur wurde in Kapitel 3.2.1.2 gegeben, weshalb hier nur auf das Zugriffskontroll-Modell und die Policy-Sprache eingegangen wird. XACML legt durch die Nutzung von XML eine Nähe zum Webservice-Umfeld nahe. Die Flexibilität und Plattformunabhängigkeit (A3.1) steht im Vordergrund, was sich unter anderem daran festmachen lässt, dass es viele Möglichkeiten der Strukturierung und Hierarchisierung von Zugriffskontroll-Informationen gibt. Die Policies als Kernelement werden nicht unmittelbar mit den zu schützenden Objekten verknüpft, sondern es wird ihre Anwendbarkeit individuell geprüft, was zu Skalierungsproblemen gerade in großen Webservice-Szenarien führen kann. Es gibt auch weitere Nachteile, die dadurch entstehen, dass XACML nicht unmittelbar als Zugriffskontroll-Modell für Webservice-orientierte Architekturen entwickelt wurde. So werden zwar keine Einschränkungen hinsichtlich der Granularität der zu schützenden Objekte getroffen, was es ermöglicht, auch Webservices beziehungsweise Webservice-Operationen zu betrachten. Allerdings stellt das Konzept der `Targets` mit den typischen Elementen Subjekt, Objekt und Aktion einen zu geringen Funktionsumfang bereit, wenn es um Webservices als konkret zu schützende Ressourcen geht. Insbesondere fehlt die Möglichkeit, das Modell so zu erweitern, dass neben der Aktion, die durch die Granularität im Webservice-Kontext üblicherweise ein einheitlicher Aufruf (engl. *Execute*) darstellt, auch die Parameter des Aufrufs berücksichtigt werden (A1.1). Auch der Aspekt der expliziten Dienstkomposition spiegelt sich nicht im XACML-Modell wider.

3.4 Modellgetriebene Entwicklung von Zugriffskontrolle

Zugriffskontrolle und insbesondere die Zugriffskontroll-Policies selbst stehen in unmittelbarem Bezug zu den fachlichen Komponenten, für die sie Zugriffskontrolle ermöglichen sollen. Analog zur strukturierten Entwicklung von fachlicher Software ist daher auch eine analoge Vorgehensweise zur Entwicklung der zugehörigen Zugriffskontroll-Policies zu empfehlen. Nachfolgend werden Ansätze vorgestellt und bewertet, die sich mit dem Themenkomplex der modellgetriebenen Entwicklung von Zugriffskontrolle der notwendigen Verzahnung mit dem Software-Entwicklungsprozess beschäftigen.

3.4.1 Modellgetriebene Sicherheit mit SecureUML

Neben den Anforderungen, die aus Geschäftssicht gestellt werden, existiert eine Vielzahl nichtfunktionaler Eigenschaften, die ein Software-System erfüllen muss. Als prominentestes Beispiel für eine nichtfunktionale Eigenschaft im Fokus dieser Arbeit ist die Zugriffskontrolle zu nennen, also das Verhindern von unautorisierten Zugriffen auf ein Software-System. Während mit der UML eine Sprache besteht, die sich gut zur Darstellung der fachlichen Anforderungen eignet, widmet sich mit [Lo03] eine Dissertation der Verzahnung von Sicherheitsanforderungen mit Software-Entwicklungsprozessen. Ausgangspunkt ist die Erkenntnis, dass die Anwendungsentwicklung einerseits und die Realisierung von Sicherheitsmechanismen andererseits üblicherweise getrennt voneinander ausgeführt werden. Dies führt jedoch dazu, dass die Sicherheit in IT-Systemen oftmals nur als nachrangiges Ziel behandelt wird und somit schlecht in die Gesamtarchitektur integriert wird. In vielen Fällen werden Sicherheitsanforderungen erst kurz vor oder während der Inbetriebnahme eines Software-Systems analysiert, definiert und implementiert. Aufgrund von Kosten- und Zeitdruck wird sogar häufig auf wesentliche Sicherheitsaspekte ganz verzichtet, sodass der Betrieb des Software-Systems oft mit erheblichen Risiken verbunden ist. Als Prämisse für die Entwicklung des fachlichen Teils werden in [Lo03] die Grundsätze der modellgetriebenen Architektur genommen (MDA, siehe Kapitel 2.3.4). Ziel der Arbeit ist es, einerseits die Sicherheitsaspekte wie allgemeine Zugriffskontroll-Informationen ebenso modellgetrieben zu effektiv ausführbaren Zugriffskontroll-Policies zu verfeinern und andererseits diese Arbeit unmittelbar mit der Entwicklung des fachlichen Teils der Software zu verzahnen. Hierfür wird in der Arbeit eine sogenannte schwergewichtige Erweiterung der UML vorgenommen (siehe Kapitel 0). Der vorgestellte Ansatz stützt sich auf eine MOF-basierte Erweiterung, das heißt, die entstehenden Sprachelemente sind zunächst unabhängig von den Metamodellen der UML, also der UML-*Infrastructure* und der UML-*Superstructure*. Es mit SecureUML ein Metamodell entwickelt, das das RBAC-Basismodell (siehe Kapitel 2.2.2) mit kleineren Erweiterungen in eine UML-konforme Darstellung überführt. Der Ausgangspunkt des modellgetriebenen Ansatzes stellt dann die Verknüpfung von Rollen aus dem SecureUML-Modell mit Elementen aus Komponenten- oder Klassendiagrammen dar, die über Assoziationsklassen mit erweiterter Semantik verknüpft werden. Diese Verknüpfung stellt dann den Ausgangspunkt für die Modelltransformation dar, die eine Abbildung der fachlichen Modelle in *Enterprise Java Beans* (EJB) und Microsoft .NET ermöglicht.

Bewertung

Die Bewertung von SecureUML wurde in Kapitel 3.3.1 vorgenommen, weshalb hier nur der Aspekt der modellgetriebenen Policy-Entwicklung (A4.1) betrachtet wird. Der sehr formale Ansatz der Entwicklung eines MOF-basierten Metamodells mit einer Anknüpfung an modellierte Klassen und Komponenten der fachlichen UML-Modelle sorgt zwar für eine direkte Bindung von Fachfunktionalität mit Zugriffskontroll-Information, erschwert jedoch eine Werkzeugunterstützung, da die Mehrzahl der UML-Modellierungswerkzeuge sich mit einer schwergewichtigen Erweiterung der UML schwer tun. Die Fixierung von SecureUML auf ein vergleichsweise gering erweitertes RBAC-Zugriffskontroll-Modell schränkt den Einsatz in Webservice-orientierten Architekturen ein. Ebenfalls ist festzuhalten, dass die Verknüpfung von fachlichen Modellen mit den Sicherheitsmodellen erst sehr spät im Software-Entwicklungsprozess stattfindet, nämlich auf Ebene der Klassen- bzw. Komponentenmodelle. Das führt zwar zu einer vergleichsweise guten Durchführbarkeit von modellgetriebenen Ansätzen, schafft jedoch nicht die Möglichkeit, dass bereits frühzeitig beispielsweise durch Geschäftsanalysten in der Analyse-Phase bereits erste Zugriffskontroll-Informationen formal festgehalten werden. Mit dem Ausgangspunkt der (erweiterten) Klassenmodelle werden Abbildungen in die JEE- und .NET-Plattform definiert, die allerdings ebenso fest mit den den Plattform mitgelieferten Zugriffskontroll-Strukturen wie beispielsweise dem *Java Authentication and Authorization Service* (JAAS) verknüpft sind. Flexibilität hinsichtlich externen Zugriffskontroll-Architekturen ist nicht vorgesehen. Der Ansatz greift also erst zum Zeitpunkt der bereits technologiespezifischen Komponentenbildung, in dem die passenden Zugriffskontroll-Policies an diese „angehängt" werden, um anschließend gemeinsam in die konkrete Technologie transformiert zu werden. Eine weitere Fragestellung, die dabei nicht berücksichtigt wird, ist, wie die Policies mit den Geschäftsprozessen zusammenhängen und sukzessive daraus abgeleitet werden können.

3.4.2 Einsatz von Prädikatenlogik zur Modellierung von Zugriffsregeln

In [AB+06] wird versucht, eine systematische Bearbeitung von Sicherheitsaspekten über den gesamten Software-Entwicklungszyklus hinweg durchzuführen. Im Besonderen wird auf die frühzeitige Erhebung von Sicherheitsanforderungen für Zugriffskontrolle bereits im Rahmen der Geschäftsprozessmodellierung abge-

stellt, die eine tatsächliche Ende-zu-Ende Zugriffskontrolle ermöglicht und nicht wie Standards wie WS-Security (siehe Kapitel 3.2.2.1) lediglich dies auf Punkt-zu-Punkt-Basis ermöglicht. Dieser Ansatz hat das Ziel, diese frühzeitig und sehr abstrakt definierten Zugriffskontroll-Informationen während des Software-Entwicklungsprozesses sukzessive zu verfeinern. Ausgangspunkt sollen plattformunabhängige Zugriffskontroll-Policies darstellen und es soll ein modellgetriebener Ansatz zum Einsatz kommen. Zur Verknüpfung der sicherheitsbezogenen Modelle mit denen des fachfunktionalen Entwicklungszyklus werden drei Modelle miteinander in Beziehung gebracht. Aus fachlicher Seite wird ein Modell der Schnittstellen als Dienstzugangspunkt genutzt und mit einer Rolle aus dem unternehmensweiten Rollenmodell verknüpft. Mit dieser Verknüpfung können weitere Bedingungen in Beziehung gebracht werden, hierzu wird ein sogenanntes Zugriffskontroll-Modell verwendet. Dieses Modell erlaubt, aufbauend auf der modellierten Rolle/Objekt-Relation aus dem RBAC-Basismodell, weitere Einschränkungen darzustellen. Diese Einschränkungen können individuelle Charakteristiken wie über die Rolle hinaus gehenden Attribute des aufrufenden Subjekts oder Umgebungszustände sein. Hierzu wird eine plattformunabhängige Sprache als PIM des MDA-Ansatzes entwickelt, die den Namen SECTET-PL trägt. SECTET-PL ist eine auf Prädikatenlogik basierende Sprache, die eine feingranluare Zugriffskontrolle auf Webservices ermöglichen soll. Vereinfacht dargestellt wird hierzu jede Webservice-Operation um Prädikate ergänzt, die die Zugriffskontroll-Policy darstellen. Dabei gibt es Prädikate mit positivem Effekt (engl. *Permission*) und Prädikate mit negativem Effekt (engl. *Prohibition*). Eine Webservice-Operation kann mit mehreren Prädikaten versehen werden, die jeweils unmittelbar mit einer Rolle als Auswahlkriterium des Prädikats verknüpft sind. Der eigentliche Inhalt der Prädikate wird nicht vorgegeben. In den angegebenen Beispielen sind boolesche Ausdrücke erkennbar, die neben einen Vergleich von übergebenen Parametern mit definierten Konstanten ermöglichen. Nach der Spezifizierung von Zugriffskontroll-Policies zu Webservice-Schnittstellen in der Phase des Entwurfs in der Software-Entwicklung muss eine Transformation aus den prädikatenlogischen Ausdrücken in effektiv auswertbare Zugriffskontroll-Policies durchgeführt werden. Als plattformspezifisches Modell im Sinne der MDA wird XACML vorgeschlagen und eine entsprechende Architektur präsentiert, die allerdings dicht an der XACML-Architektur (siehe Kapitel 3.2.1.2) liegt. Die Transformation wird in rein textueller Form dargestellt und ermöglicht daher keine automatische Transformation der Modelle ineinander.

Bewertung

Aufgrund der Spezifik wird nachfolgend im Wesentlichen auf die Anforderung A4.1 zur modellgetriebenen Entwicklung der Policies eingegangen. Die Entwicklung einer auf Prädikatenlogik aufgebauten Sprache zur abstrakten Darstellung von Zugriffskontroll-Informationen ist grundsätzlich begrüßenswert, da sie eine formale Darstellung ermöglicht. Für den Zweck der frühzeitigen Einbindung in den Software-Entwicklungsprozess und der Nutzung durch Geschäftsanalysten im Rahmen der Geschäftsprozessmodellierung ist sie jedoch nur bedingt geeignet, da die Komplexität der Sprache und ihrer korrekten Nutzung nicht minder gering ist als die von XACML. Positiv zu bewerten ist die Verknüpfung der Zugriffskontroll-Policies mit den Webservice-Operationen als den zu schützenden Objekten. Die Verknüpfung der Operationen mit rollenbezogenen Regeln ermöglicht zwar für die Transformation eine bessere Abbildung dieser Artefakte in XACML, führt jedoch unnötige Abhängigkeiten zwischen diesen beiden Sprachen ein. Des Weiteren ist eine Ausrichtung der Regeln an der Rolle des zugreifenden Subjekts unflexibel, was die Dienstwiederverwendung angeht. Wenn ein Dienst in einem erweiterten oder neuen Nutzungskontext zum Einsatz kommt, bei dem bestehende Rollen weitere Rechte bekommen sollen, müssen potenziell alle Regeln angepasst werden. Hier wäre die Einführung einer zusätzlichen Hierarchie sinnvoll, die die Ebene der Nutzungskontexte abbildet und dadurch eine Veränderung bestehender Zugriffskontroll-Policies beim Hinzufügen neuer Einsatzszenarien des Webservices unnötig macht. Die Nutzung von XACML als plattformspezifisches Modell erscheint für einen praxisorientierten Einsatz unrealistisch, da die derzeitige Produktunterstützung von XACML sehr dünn ist. Ziel sollte eher sein, als plattformspezifisches Modell dasjenige einer bestehenden Sicherheitslösung zu verwenden. Dies ermöglicht den tatsächlichen Einsatz, erschwert jedoch in nachvollziehbarer Weise die ohnehin schwierige Verknüpfung von PIM und PSM und damit die Transformation.

3.5 Synchronisation von Benutzerdaten

Benutzerdaten in ihrer Form als digitale Identitäten stellen ein zentrales Element der Subjekt/Objekt-Relation dar, wenn es um die Durchführung von Zugriffskontroll-Entscheidungen geht. Typischerweise liegen Benutzerdaten in Unternehmen verteilt vor. Verschiedene Anwendungssysteme mit integrierter Zugriffskontrolle halten sie jeweils intern vor, Personalabteilungen pflegen Stammdaten, die den Kern einer digitalen Identität bilden. Oft gibt es auch

anwendungsübergreifende Benutzerverzeichnisse, die von mehreren Systemen gemeinsam genutzt werden. Notwendige Voraussetzung für ein effizientes Identitätsmanagement und eine effektive Zugriffskontrolle ist die Synchronhaltung dieser verschiedenen Datenbestände und damit einhergehend die Schaffung einer homogenen Sicht auf den Gesamtbestand an Benutzerdaten. Nachfolgend werden zunächst allgemeine Integrationsvorgehen aufgezeigt. Es schließen sich Beispiele aus dem Hochschulkontext an. Weiter wird mit dem kommerziellen Sektor eine Bewertung des bestehenden Produktumfelds durchgeführt. Das Kapitel schließt mit der Anforderungsdefinition für ein interoperables Informationsmodell für Benutzerdaten.

3.5.1 Allgemeine Integrationsvorgehen für Benutzerdaten

Zwei Ansätze werden üblicherweise für die unternehmensinterne Integration von Benutzerdaten genutzt: Meta-Verzeichnisse und virtuelle Verzeichnisse.

3.5.1.1 Meta-Verzeichnisse

Meta-Verzeichnisse ermöglichen eine Aggregierung von Benutzerdaten von syntaktisch und technologisch verschiedenen Datenquellen dar. Ein Meta-Verzeichnis umfasst nach [Ni03] die folgenden Elemente: bestehende Datenquellen wie LDAP-basierte Verzeichnisse, relationale Datenbanken oder *Flat Files*. Diese werden über individuelle Konnektoren an eine Verbindungseinheit (engl. *Join Engine*) angeschlossen. An dieser Verbindungseinheit wird ein Meta-Verzeichnis als zentrales Verzeichnis angeschlossen, das eine Gesamtsicht aller Benutzer darstellt. Neben der Möglichkeit, alle Benutzer aus allen Verzeichnissen mit allen Attributen ihrer digitalen Identität in das zentrale Meta-Verzeichnis zu provisionieren und damit auch synchron halten zu müssen, können Filterungen vorgenommen werden. Für den Fall, dass manche Benutzer tatsächlich nur lokal für ein einzelnes Anwendungssystem Relevanz besitzen, kann eine Synchronisation mit dem Meta-Verzeichnis ausgespart bleiben. Ebenso sind einige Attribute einer digitalen Identität, insbesondere die in den äußeren Schalen des Kernmodells, nur für spezifische Anwendungssysteme relevant. Es kann sinnvoll sein, diese ebenfalls nicht mit dem Meta-Verzeichnis abzugleichen. Es können unterschiedliche Meta-Sichten (engl. *Meta View*) aufbauend auf den Daten des Meta-Verzeichnisses gebildet werden. Eine häufige Einsatzform von Meta-Verzeichnissen ist es, nur die für alle Anwendungssysteme relevanten Kerndaten

in ein gemeinsames Meta-Verzeichnis zu synchronisieren und darüber konsistent zu halten. Weitere Ansätze ermöglichen die Ankopplung an Personalmanagementsysteme aus den Personalabteilungen, die üblicherweise die Datenverantwortlichkeit für diese Kerndaten innehaben. Die Richtung der Datenpflege ist nicht festgelegt: Es gibt einerseits die Möglichkeit, dass alle Datenänderungen über das Meta-Verzeichnis initiiert werden müssen, oder aber, dass die Daten weiterhin über die jeweiligen Anwendungssysteme mit ihren lokalen Datenhaltungen gepflegt werden. Wichtig ist eine Absprache, welche an ein Meta-Verzeichnis angeschlossene Einheit für die Pflege welche Datenbereiche einer digitalen Identität verbindlich zuständig ist.

Bewertung

Aufgrund der Spezifik von Meta-Verzeichnissen als Mittel zur Benutzerintegration sind diese nicht hinsichtlich des gesamten Anforderungskatalogs zu bewerten. Daher wird nur die nach plattformunabhängiger und expliziter Darstellung der Semantik der Benutzerdaten in einem gemeinsamen Modell untersucht (A5.1). Die bestehenden Lösungen sind üblicherweise Erweiterungen von LDAP-basierten Verzeichnissen um Verbindungseinheiten (engl. *Join Engine*). Das durch das Meta-Verzeichnis genutzte gemeinsame Datenmodell ist daher ein LDAP-Schema, das eine ausgeprägte syntaktische Prüfung erlaubt, die Semantik der Attribute jedoch nur über „sprechende“ Namen definiert. Der Entwickler, der ein neues Verzeichnis an das Meta-Verzeichnis anbinden soll, muss sich daher die Abbildungsregeln daraus ableiten, was üblicherweise zu Problem führt. Meta-Verzeichnisse fördern die redundante Vorhaltung von Benutzerdaten über verschiedene Systeme hinweg. Hier ist ein besonderes Augenmerk darauf zu richten, wer für welche Daten die Autorität übernimmt. Gerade bei Änderungen an derselben Identität in verschiedenen Systemen kann es sonst in der Verbindungseinheit zu Konflikten kommen. Problematisch an Produkten, die Meta-Verzeichnisse realisieren, ist, dass die Abbildungsregeln, die in der Verbindungseinheit genutzt werden, produktspezifisch sind. Des Weiteren kann die Erstellung der individuellen Konnektoren für die technische Anbindung an die Verbindungseinheit beliebig aufwendig sein. Oftmals werden daher entsprechende Konnektoren für die handelsüblichen Anwendungssysteme bereits mitgeliefert. Der Aufwand entsteht dann nur für diejenigen Anwendungen, für die der Hersteller der Meta-Verzeichnis-Lösung keine Konnektoren bereitstellt.

3.5.1.2 Virtuelle Verzeichnisse

Der Unterschied zwischen einem „normalen“ Verzeichnis und einem virtuellen Verzeichnis liegt vereinfacht gesagt darin, dass ein virtuelles Verzeichnis keine lokale Datenhaltung beinhaltet. Die Anfrage wird stattdessen auf ein oder mehrere dahinterliegende Datenquellen wie andere Verzeichnisse oder Datenbanken umgelenkt. Das virtuelle Verzeichnis stellt damit eine Art Stellvertreter (engl. *Proxy*) dar, wenn auch nicht notwendigerweise in 1:1-Beziehung. Aus Sicht des Anfragenden ist dieser Umstand jedoch nicht ersichtlich. Ein virtuelles Verzeichnis hat das Ziel, bestehende Datenquellen eines Unternehmens zusammenzufassen, die sowohl hinsichtlich ihres internen Aufbaus als auch hinsichtlich der eingesetzten Protokolle heterogen sein können. Es besteht nach [Ge05] aus drei logischen Schichten: Es bietet seine Dienste gegenüber potenziellen Dienstnehmern an definierten Außenschnittstellen an, wobei üblicherweise mehrere Protokolle nutzbar sind. Typischerweise wird LDAP genutzt, aber es können auch weniger häufig eingesetzte Protokolle wie die *Directory Service Markup Language* (DSML) oder Schnittstellen sein, die im Kontext relationaler Datenbanken genutzt werden. Zur Anbindung der bestehenden Datenquellen wie LDAP-Verzeichnisse, relationale Datenbanken oder sonstigen Quellen wie beispielsweise flache XML-Dateien werden sogenannte Konnektoren eingesetzt. Diese Konnektoren führen eine rein syntaktische Ankopplung der Datenquellen an das virtuelle Verzeichnis vor und sorgen für den prinzipiellen Zugriff. Für den Zugriff des virtuellen Verzeichnisses auf die dadurch verschatteten Datenquellen werden üblicherweise technische Benutzer eingesetzt. Die Entwicklung von individuellen Konnektoren ist durch ein solches Konzept ebenfalls möglich, kann sich jedoch beliebig aufwendig gestalten. Der Kern eines virtuellen Verzeichnisses liegt in seiner internen Prozesslogikschicht. Hier wird die eingehende Anfrage von der Schnittstelle des Dienstzugangspunktes ausgewertet. Im einfachsten Fall muss die Anfrage lediglich unbehandelt an das passende Zielverzeichnis durchgeleitet und die resultierende Antwort zurückgegeben werden. Komplexere Szenarien setzen auf eine Lastverteilung der Anfragen durch das virtuelle Verzeichnis. Hierzu können bewusst dem virtuellen Verzeichnis nachgelagerte Datenquellen redundant vorgehalten werden, was neben der Lastverteilung und damit Performanzsteigerung auch eine bessere Ausfallsicherheit ermöglicht. Auch Protokolltransformationen können durch das virtuelle Verzeichnis durchgeführt werden. Das ist im Besonderen dann interessant, wenn ein Datensatz auf mehrere dahinterliegende Datenquellen verteilt ist, die unterschiedliche Protokolle nutzen. Im Bereich der digitalen Identitäten können die Kerndaten beispiels-

weise in einem LDAP-basierten Verzeichnis vorgehalten werden, während ergänzende Informationen in einer relationalen Datenbank der Personalabteilung gepflegt werden. Ein virtuelles Verzeichnis ermöglicht eine gesamtheitliche Sicht auf diese Daten, und dies ohne eine explizite Provisionierung und Synchronisation erforderlich zu machen.

Bewertung

Virtuelle Verzeichnisse stellen einen ersten Schritt in die Richtung eines Identitätsdienstes dar und zwar dadurch, dass technische Abhängigkeiten durch das Anbieten einer ausreichenden Menge an standardisierten Dienstzugangspunkten für den Dienstnehmer reduziert werden. Dies ermöglicht, notwendige Protokolltransformationen außerhalb der fachlichen Anwendungen durchzuführen. Auch die Notwendigkeit, sich Daten aus unterschiedlichen Quellen selbst zusammenzustellen, wird durch virtuelle Verzeichnisse reduziert. In diesem Sinne stellen virtuelle Verzeichnisse eine Indirektionsstufe zwischen Dienstnutzer und den Datenquellen für digitale Identitäten dar. Neben der Erhöhung der Verfügbarkeit durch Lastverteilungsmechanismen kann jedoch die Performanz eines Zugriffes durch diese Indirektion etwas gebremst werden. Im Gegensatz zu Meta-Verzeichnissen wird die Redundanz der Datenhaltung minimiert, da die Daten an dem Ort gehalten und gepflegt werden, der ursprünglich dafür vorgesehen war. Virtuelle Verzeichnisse ermöglichen es, unterschiedliche und damit individuelle Sichten für unterschiedliche Dienstnutzer zu erstellen. Ein zentraler Punkt ist jedoch die Behandlung der Semantik der Daten, die durch das virtuelle Verzeichnis zusammengeführt werden. Diese wird in der Regel hart verdrahtet der Schicht der internen Prozesslogik eines virtuellen Verzeichnisses zugeführt. Beim Hinzufügen einer zusätzlichen Datenquelle kann daher nicht auf ein explizit definiertes, gemeinsames Semantik-Verständnis zurückgegriffen werden, sondern die Verknüpfungsregeln sind individuell zusammengestellt (A5.1).

3.5.2 Benutzerintegration im Hochschulkontext

Im Hochschulkontext findet man über Jahre gewachsene Anwendungslandschaften, die oft durch Individualität und stetige Anpassungen an neue Anforderungen gekennzeichnet sind. Projekte an unterschiedlichen Hochschulen nehmen als Teilaufgabe die Benutzerintegration in Angriff. Im integraTUM-Projekt an der TU München [Ho07] wird eine Rezentralisierung von Diensten angestrebt, was einen gemeinsamen Benutzerdatenbestand voraussetzt. Das Projekt „Werkstatt

Unternehmenssoftware Karlsruhe“ (WUSKAR) [EA04], an dem der Autor maßgeblich beteiligt war, hat in mehreren Fallstudien die Benutzerdatensynchronisation im Hochschulkontext untersucht und prototypische Lösungen implementiert [ME+05, EM+06]. Mit dem an der Universität Karlsruhe (TH) beheimateten Projekt „Karlsruher Integriertes InformationsManagement“ (KIM) findet eine Geschäftsprozess-orientierte Anwendungsintegration auf Basis von Webservice-Architekturen statt, die ebenfalls eine Benutzerdatenintegration voraussetzt. Dieses Projekt wird stellvertretend nachfolgend vorgestellt.

3.5.2.1 Karlsruher Integriertes InformationsManagement (KIM)

Im Zuge einer Neustrukturierung der bestehenden Anwendungslandschaft an der Universität Karlsruhe (TH) wurde im Jahr 2005 das Projekt „Karlsruher Integriertes InformationsManagement“ (KIM) [UKA-KIM] ins Leben gerufen. Es handelt sich dabei um ein Integrations- und Reorganisationsprojekt, das über verschiedene Organisationseinheiten an der Universität verteilt bearbeitet wird. Ziel des Projekts ist die Verbesserung der Lehre durch besser verzahnte Geschäftsprozesse mit optimaler Rechnerunterstützung. Durch die bestehende technische Heterogenität ist die von verschiedenen Organisationseinheiten angebotene, rechnergestützte Fachfunktionalität zwar in den meisten Fällen ausreichend, aber die Geschäftsprozess-abbildende Integration dieser Funktionalität ist aufgrund technischer Inkompatibilität nicht effizient möglich. Durch das KIM-Projekt sollen diese Schwierigkeiten durch die Entwicklung und den Betrieb einer organisatorischen und technischen Infrastruktur auf Basis einer dienstorientierten Architektur gelöst werden. Diese Software-Architektur soll die Integration und das Angebot von Diensten verschiedener Systeme und Organisationseinheiten in einem gesamtuniversitären Kontext ermöglichen beziehungsweise vereinfachen. Während der ersten Phase des Projekts, die von Prozessmodellierung und fachlicher Integration geprägt war und an der der Autor beteiligt war, wurde schnell deutlich, dass ein integriertes Identitätsmanagement das Fundament bilden muss, auf dem diese dienstorientierte Architektur aufzubauen ist. Ebenfalls wurde erkannt, dass es wenig sinnvoll ist, das Identitätsmanagement als Teil des Sicherheitskonzepts dem Integrationsprojekt nachträglich „unterzuschieben“. Dies führte während des ersten Projektjahres zur Schaffung eines weiteren, zugehörigen Projekts namens KIM-IDM, das sich ausschließlich um die Konzeption und Entwicklung eines zur dienstorientierten Facharchitektur von KIM passenden Identitätsmanagements kümmert. Dies ist wichtig, da

durchweg personenbezogene Informationen durch die zu integrierenden Anwendungssysteme verarbeitet werden.

In [HM+06] werden die Prämissen und die Vorgehensweise des KIM-IDM-Projekts vorgestellt. Grundlage der Architektur ist die Betrachtung der Universität als föderierter Verbund ihrer organisatorischen Einheiten, die als Satelliten bezeichnet werden. Ziel ist es, nicht notwendigerweise eine zentrale Datenbasis für die Benutzerdaten der Hochschule etablieren zu müssen. Dabei wird sowohl ein Meta-Verzeichnis-Ansatz ausgeschlossen, der sämtliche Benutzerdaten aus den jeweiligen Organisationseinheiten vollständig zusammenführt, als auch ein Ansatz, der lediglich die Kerndaten der Benutzer in einem führenden und damit autoritativen System vorhält. Stattdessen soll die notwendige integrierte Gesamtsicht der Daten durch dynamische und föderierte Prozesse herbeigeführt werden, wobei sich Analogien zu einem virtuellen Verzeichnis, wie in Kapitel 3.5.1.2 eingeführt, bilden lassen. Dies hängt mit der Prämisse zusammen, dass die jeweiligen Organisationseinheiten weiterhin vollständige Souveränität über ihre lokalen Daten behalten sollen. Die Ankopplung soll über explizit definierte Dienstschnittstellen erfolgen und die Anpassung der Datenschemata über die jeweiligen Kopplungselemente erfolgen.

Bewertung

Das Projekt KIM-IDM ist noch nicht abgeschlossen und einige Entwurfsentscheidungen sind daher noch nicht finalisiert. Eine Bewertung des Vorgehens kann daher nur auf Basis des aktuellen Standes (Dezember 2007) vorgenommen werden. Die angestrebte Lösung muss gemäß der Projektanforderung allgemeingültig einsetzbar sein und wird nur auf Webservice-basierte, dienstorientierte Architekturen eingeschränkt (A1.1). Die frühe Verzahnung des fachlichen Teils einer dienstorientierten Architektur mit den digitalen Identitäten als Baustein einer passenden Zugriffskontroll-Architektur ist grundsätzlich positiv zu bewerten. Gerade durch die starke technische Heterogenität der Rechnerunterstützung in den einzelnen Organisationseinheiten und die Notwendigkeit, sich dem Kunden (also dem Studierenden) gegenüber dienstorientiert aufzustellen, machen die Migration in eine dienstorientierte Software-Architektur sinnvoll. Zum derzeitigen Zeitpunkt wird eine integrierte Entwicklung von Fachfunktionalität und zugehöriger Zugriffskontrolle nicht unterstützt (A1.2). Dies wird jedoch für den weiteren Projektverlauf angestrebt. Ebenso sind die Dienstschnittstellen noch nicht spezifiziert, die ein Fachentwickler zum Zugriff auf die Identitätsdienste

nutzen kann. Daher sind die Anforderungen A2.1 bis A2.3 nicht bewertbar. Auch das Zugriffskontroll-Modell und die Policy-Sprache sind noch Gegenstand zukünftiger Entwicklung (A3.1 und A3.2), genauso wie die Einbindung in einen modellgetriebenen Software-Entwicklungsprozess (A4.1). Der logische erste Schritt, die Integration der bestehenden Nutzer, ist ein richtiger Ausgangspunkt als Vorbedingung zur Entwicklung einer Zugriffskontroll-Architektur. Die Frage, auf welche Art und Weise die Semantik der bestehenden Schemata der digitalen Identitäten in den einzelnen Organisationseinheiten aufeinander abgebildet werden, wird wiederum durch eine statische und produktspezifische Abbildung durchgeführt (A5.1). Gerade bei direkten Synchronsationsvorgängen zwischen einzelnen Organisationseinheiten führt das zu einem quadratischen Aufwand, wenn neue Datenquellen in den Verbund aufgenommen werden sollen. Diesbezüglich wird derzeit an einer zentralen Datenhaltung für Identifikator-Abbildungen zwischen den digitalen Identitäten der jeweiligen Organisationseinheiten gearbeitet. Dieser notwendige Schritt geht jedoch wieder in die Richtung eines Kerndaten-basierten Meta-Verzeichnisses. Eine ebenfalls noch zu betrachtende Aufgabe ist die Nutzung dieser Identitätsdaten zur Durchführung von Zugriffskontrolle auf Dienstebene, insbesondere bei Organisationseinheitsübergreifenden, fachlichen Dienstkompositionen in der dienstorientierten Architektur. Vorbedingung hierzu ist die Entwicklung eines Zugriffskontroll-Modells, das analog zur Autarkie der Organisationseinheiten eine verteilte Zugriffskontroll-Entscheidung bei übergreifenden Geschäftsprozessen ermöglicht. Dieser Sachverhalt muss noch untersucht werden, um die Zugriffskontroll-Architektur als Baustein einer effektiven Zugriffskontrolle zu gestalten.

3.5.3 Kommerzielle Produkte zur Benutzerintegration

Es gibt auch eine Vielzahl an kommerziellen Produkten, die als Ziel die Integration und Synchronisation von bestehenden Identitätsdatenquellen haben. Nachfolgend werden mit dem IBM Tivoli Directory Integrator eine zentrale Komponente aus der Tivoli-Serie von IBM vorgestellt.

3.5.3.1 Tivoli Directory Integrator (TDI)

Bei IBM Tivoli Directory Integrator (TDI) handelt es sich um eine Meta-Verzeichnis-Lösung mit offener Architektur, die dem Austausch und der Synchronisierung von Informationen über Anwendungs- und Plattformgrenzen

hinweg dient. Ziel ist die unternehmensweite Integration von Benutzerdaten über alle Verzeichnisse hinweg [IBM-TDI]. Der TDI stellt dabei nur ein einzelnes Element einer gesamtheitlichen Architektur dar, die aus den zu benennenden autoritativen Datenquellen einzelner Anwendungssysteme besteht, die durch sogenannte Meta-Verzeichnisdienste auf Änderungen überwacht werden. Diese Änderungen werden an ein logisch getrenntes Provisionierungssystem übermittelt, das das unternehmensweite Meta-Verzeichnis aktualisiert und die Synchronisation der Zielsysteme ansteuert. Ein Beispiel für eine Integrationsarchitektur auf Basis des TDI wird ausführlich in [JS03] beschrieben. Hauptaugenmerk des TDI liegt auf den individuellen Konnektoren zur Anbindung der Quell- und Zielanwendungen und in einer grafischen Benutzeroberfläche zur Definition, Steuerung und Überwachung der Synchronisationsvorgänge.

Bewertung

Gut ausgearbeitete Architekturvorschläge und ein ausgereiftes Produkt machen den TDI zu einer sinnvollen Lösung, wenn es um die Integration von Benutzerdaten über verschiedene Verzeichnisse hinweg geht. Auch hier ist jedoch die Frage, wie die Semantik der Beziehungen zwischen einzelnen Attributen der digitalen Identitäten unterschiedlicher Systeme plattformunabhängig dargestellt werden kann (A5.1). Auch hier findet durch die mitgelieferte GUI eine feste Verdrahtung zwischen den einzelnen Datenquellen statt, die einen möglichen quadratischen Aufwand bei der Einbindung neuer Verzeichnisse darstellt. Die Tivoli-spezifische Darstellung der Abbildungsregeln erschwert einen Austausch des Produkts. Ebenso liegt viel Komplexität in den Konnektoren, die die bestehenden Datenhaltungen anbinden. Für den Fall, dass ein passender Konnektor nicht besteht, kann die Entwicklung eines solchen aufwendig und damit kostenintensiv sein.

3.6 Zusammenfassung und Handlungsbedarf

Tabelle 2 fasst die Ergebnisse der Bewertung der betrachteten Ansätze des Stands der Forschung und Technik im Bereich der Zugriffskontroll-Architekturen tabellarisch zusammenfassen. In der Tabelle bedeutet ein „+“, dass eine Anforderung erfüllt ist, ein „o“, dass eine Anforderung nur teilweise erfüllt ist und ein „-“ zeigt an, dass eine Anforderung gar nicht erfüllt ist oder nicht betrachtet wird. Nachfolgend wird für jede Anforderung eine Feinspezifikation der Bewertungskriterien gegeben.

	Zugriffskontroll-Architekturen								
	Firewall-basiert (Nedgty-Firewall)	PDP/PEP-basiert (XACML-Architektur)	Referenzmonitor	Secure Service Proxy (SSP)	Intercepting Web Agent (IWA)	WS-Security	WS-Policy WS-SecurityPolicy	CA eTrust SiteMinder	IBM Tivoli Access Manager
A1.1 - Allgemeingültigkeit	+	+	+	+	+	+	+	o	o
A1.2 - Einbindung des Fachentwicklers ohne Aufwandssteigerung	-	-	-	-	o	o	-	o	o
A2.1 - Interoperable Dienstschnittstellen	-	o	o	o	o	+	+	-	-
A2.2 - Portabilität	+	o	-	+	-	-	-	-	+
A2.3 - Integrationsfähigkeit	-	o	-	o	o	o	o	+[1]	+[1]

[1] Integration nur von externen Datenhaltungsquellen wie Benutzerverzeichnissen

Tabelle 2: Bewertung bestehender Ansätze (1)

Für die Anforderung A1.1 wurde die direkte Einsetzbarkeit im Webservice-Kontext bewertet. Ein „o“ zeigt an, dass noch Anpassungen als Vorbedingung für den Einsatz in Webservice-Umgebungen notwendig sind. Hinsichtlich der Bewertung der Aufwandsreduktion für den Fachentwickler (A1.2) stellt seine Einbindung eine Voraussetzung dar. Für den Fall, dass ein Sicherheitsexperte die Umsetzung der Zugriffskontrolle (fast) vollständig übernimmt, wurde ein „o“ vergeben. Wenn für den Fachentwickler ein zusätzlicher Programmieraufwand entsteht, wurde ein „-“ vergeben. Hinsichtlich der interoperablen Dienstschnittstellen (A3.1) wurde ein „+“ vergeben, wenn eine WSDL-konforme Spezifikation gegeben ist. Ein „o“ bedeutet, dass die Erstellung einer WSDL-konformen Schnittstelle theoretisch möglich wäre, sie jedoch bisher nicht umgesetzt wurde. Ein „-“ bedeutet, dass WSDL-konforme Schnittstellen nicht bestehen oder nur mit erheblichem Aufwand entwickelt werden können. Ein „+“ bei der Portabilität (A2.2) wurde vergeben, wenn die Lösung mit geringem Aufwand in eine andere Systemumgebung (beispielsweise einen anderen *Application Server*) überführt werden kann. Ein „o“ wurde vergeben, wenn man die Lösung theoretisch so implementieren könnte, dass sie portierbar wäre. Bezüglich der Integrationsfähigkeit bestehender Sicherheitsprodukte (A2.3) wurde ein „+“ vergeben, wenn Fremdprodukte sich in die Zugriffskontroll-Architektur integrieren lassen. Ein „o“ wurde vergeben, wenn die Integrationsfähigkeit geplant ist.

In Tabelle 3 werden die Zugriffskontroll-Modelle sowie die Ansätze zur modellgetriebenen Entwicklung von Zugriffskontrolle und zur Benutzerintegration zusammenfassend bewertet. Für die Anforderung A1.1 und A1.2 gelten die vorgenannten Bewertungsschemata. Hinsichtlich der Plattformunabhängigkeit der Policies (A3.1) wurde ein „+“ vergeben, wenn keine Produktspezifika in den Policies erkennbar sind. Wenn die Einbindung in den Software-Entwicklungsprozess durchgeführt wird (A3.2), wurde ein „+“ vergeben. Um plattformunabhängige Zugriffskontroll-Policies in effektiv auswertbare Policies zu transformieren, müssen Transformationen explizit definiert werden (A4.1), was mit einem „+“ bewertet wurde. Hinsichtlich der expliziten Definition der Semantik der Benutzerdaten (A5.1) wurde ein „+“ vergeben, wenn die Daten plattformunabhängig und explizit vorliegen.

	Zugriffskontroll-Modelle			Modellgetriebene Policy-Entwicklung		Benutzerintegration		
	RBAC-basierte Modelle (SecureUML)	ABAC-basierte Modelle	XACML	SecureUML	Prädikatenlogik	Meta-Verzeichnisse / virtuelle Verz.	KIM	IBM Tivoli Directory Integrator
A1.1 - Allgemeingültigkeit	+	+	o	o	o	o	+	o
A1.2 - Aufwandsreduktion für den Fachentwickler	+	o	o	+	o	-	o	-
A3.1 - Plattformunabhängigkeit und WS-Bezug	o	+	+	+	+	-	-	-
A3.2 - Einbindung in den Entwicklungsprozess	o	-	-	+	+	-	-	-
A4.1 - Explizite Definition von Transformationen	-	-	-	+	+	-	-	-
A5.1 - Explizite Definition der Semantik	-	-	-	-	-	o[1]	-	o[1]

[1] Produktspezifische Darstellung

Tabelle 3: Bewertung bestehender Ansätze (2)

Die Analyse der Bewertungen zeigt, dass keiner der untersuchten Ansätze in der Lage ist, alle gestellten Anforderungen in zufrieden stellender Weise zu erfüllen. Defizite zeigen sich insbesondere in der Spezifikation der Dienstschnittstellen einer Zugriffskontroll-Architektur und in der Anbindung an die Facharchitektur. Ebenso sind die Zugriffskontroll-Modelle und Policy-Sprachen nicht unmittelbar für den Webservice-Kontext anwendbar. Nachgelagerte Policy-Entwicklung durch Sicherheitsexperten und die fehlende Benutzerintegration als Datenbasis

der handelnden Subjekte erschweren eine formalisierte Software-Entwicklung. Das Ziel dieser Arbeit ist es, eine integrierte Lösung zu entwickeln, die die gestellten Anforderungen erfüllt. In den Kapiteln 4 bis 7 werden die erarbeiteten Konzepte im Detail vorgestellt.

4 Dienstorientierte Zugriffskontroll-Architektur

Im Kontext Webservice-basierter, dienstorientierter Architekturen werden Webservices entwickelt und zur Unterstützung von Geschäftsprozessen komponiert. Diese fachlichen Webservices sollen um Zugriffskontrolle erweitert werden, wobei die Verknüpfung aus Webservice und Zugriffskontroll-Dienst zu einer betriebenen Einheit werden soll. Hierzu wird eine querschnittlich nutzbare Zugriffskontroll-Architektur benötigt. Sie soll den in Kapitel 3.1 definierten Anforderungen, insbesondere nach interoperablen Dienstschnittstellen (A2.1), nach Portabilität (A2.2) und Integrationsfähigkeit (A2.3) genügen. Eine solche Zugriffskontroll-Architektur ermöglicht die Ausprägung des Paradigmas der Komposition auch für den Bereich der Zugriffskontrolle. Ein Webservice wird durch eine solche Komposition mit dem lokalen *Policy Enforcement Point* und dem durch den *Policy Decision Point* bereitgestellten Zugriffskontroll-Dienst zu einer betriebenen Einheit, die um Zugriffskontrolle beim Dienstaufruf ergänzt wurde. Nachfolgend wird in Kapitel 4.1 die Entwicklung der Zugriffskontroll-Architektur als Ganzes betrachtet. Daran schließt sich in Kapitel 4.2 die Vertiefung der Verknüpfung mit der Facharchitektur an.

4.1 Entwicklung einer Zugriffskontroll-Architektur für WSOA

4.1.1 Motivation und Anforderungen

Vor dem eigentlichen Entwurf der Zugriffskontroll-Architektur muss Klarheit hinsichtlich der zu schützenden Facharchitektur geschaffen werden. Nach der formalen Darstellung der Struktur einer Webservice-orientierten Architektur werden die in Kapitel 3.1 vorgegebenen Anforderungen weiter präzisiert. Anschließend werden die durch die Zugriffskontroll-Architektur abzubildenden Prozesse vorgegeben.

4.1.1.1 Aufbau einer Webservice-orientierten Architektur

Um eine fachlich geprägte Webservice-orientierte Architektur (WSOA) um Zugriffskontrolle zu erweitern, müssen zunächst ihre Zusammensetzung aus Einzelbausteinen und das zugehörige Interaktionsverhalten der Bausteine reflek-

tiert und modelliert werden. Ansatzpunkte für den Aufbau einer WSOA wurden in Kapitel 2.1 vorgestellt. Der für die Erweiterung um Zugriffskontrolle relevante Kern einer WSOA wird durch Abbildung 24 zusammengefasst und nachfolgend diskutiert.

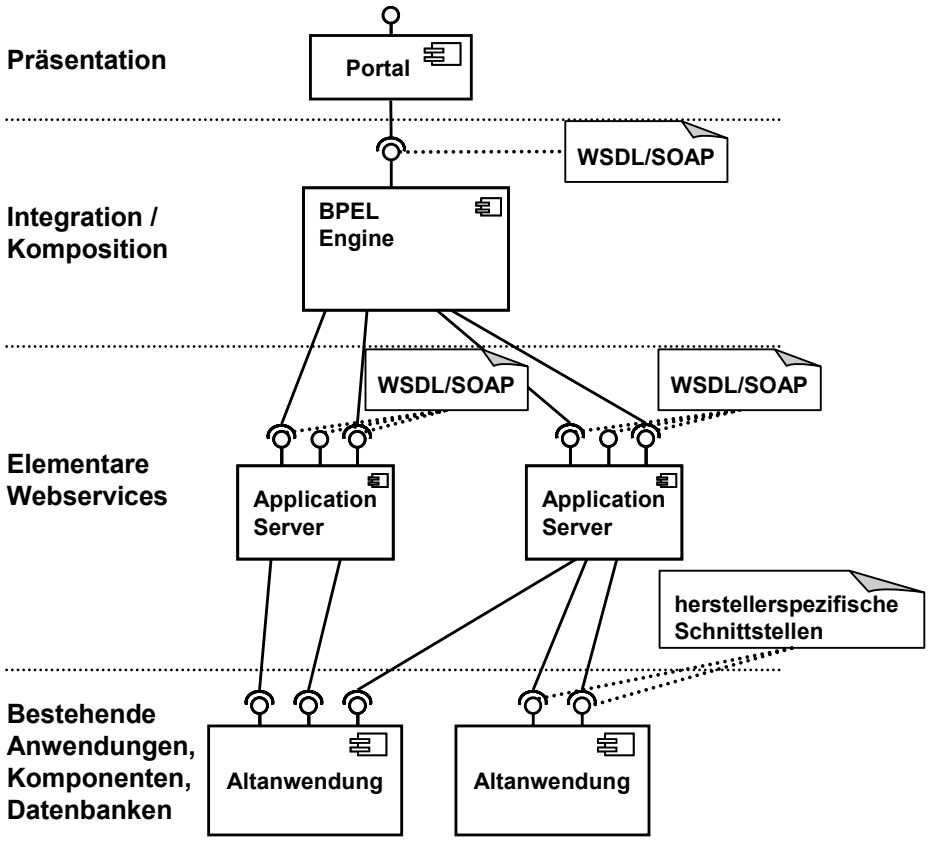


Abbildung 24: Aufbau einer Webservice-orientierten Architektur

Wie bereits beschrieben, bedeutet Dienstorientierung in der Software-Entwicklung keineswegs die Ablösung der bestehenden Anwendungssysteme durch Neuentwicklungen. Es ist ein erklärtes Ziel, bestehende Fachfunktionalität weiterzuverwenden. Daher bilden die bestehenden Anwendungen, Komponenten und Datenbanken die Grundlage und damit die unterste Schicht einer Webservice-orientierten Architektur. Diese bestehenden Anwendungssysteme (auch Altanwendung, engl. *Legacy Application*) stammen typischerweise von unterschiedlichen Herstellern und sind in unterschiedlichen Programmiersprachen implementiert. Falls Schnittstellen angeboten werden, sind diese üblicherweise herstellerspezifisch ausgeprägt. Dies erschwert die geschäftsprozessorientierte Anwendungsintegration über Systemgrenzen hinweg. Die darüberliegende Schicht der elementaren Webservices bildet den Einstiegspunkt in die Webservice-Technologien. Bestehende Funktionalität der Altanwendungen oder auch neu

entwickelte Fachfunktionalität wird über *Application Server* als wiederverwendbare Dienste an interoperablen Dienstschnittstellen zur Verfügung gestellt. Die Interoperabilität impliziert die Beschreibung der Schnittstelle mittels der herstellerunabhängigen *Web Services Description Language* (WSDL) und die Nutzung von SOAP (früher: *Simple Object Access Protocol*) als Kommunikationsprotokoll. Die WSOA ermöglicht, wie in Kapitel 2.1 vorgestellt, eine schrittweise Migration zu Webservices. In der darüberliegenden Integrations-/-Kompositionsschicht wird die klassische Anwendungsintegration durchgeführt, nun jedoch mit der Vereinfachung der gemeinsam genutzten Webservice-Technologien. Diese Schicht wird oft in zwei Teilschichten aufgetrennt: Die untere der beiden führt die aus technischer Sicht motivierte Anwendungsintegration durch, während die virtuell darüberliegende Schicht die Abbildung von Geschäftsprozessen auf unmittelbar ausführbare Dienstkompositionen explizit macht. Während diese explizite Abbildung von Geschäftsprozessen in unmittelbar ausführbare Dienstkompositionen eine Errungenschaft dienstorientierter Architekturen darstellt, ist aus technischer Sicht zwischen den beiden Teilschichten kein Unterschied erkennbar. Sowohl die aus Geschäftssicht umgesetzte Dienstkomposition als auch die aus technischer Sicht notwendige Anwendungsintegration werden in WSOA mit Hilfe der *Business Process Execution Language* (BPEL) durchgeführt. BPEL-*Engines* ermöglichen die unmittelbare Ausführung der XML-basierten BPEL-Artefakte. Für den Anwender wird die Komplexität der WSOA hinter Portalen als Dienstzugangspunkten verschattet. Dies erlaubt, bestehende Technologien wie klassische Webbrowser weiterzuverwenden. Im Portal findet dann die Umsetzung der HTTP-basierten Kommunikationsbeziehung mit dem Benutzer in die Technologien der WSOA statt. Die in Abbildung 24 vorgestellte Struktur erzwingt jedoch die strikte Schichtung nicht. Webservices können entweder direkt oder über Intermediäre wie BPEL-Kompositionen oder über Nachrichtenverteiler wie einem *Enterprise Service Bus* (ESB) aufgerufen werden. Auch die BPEL-Kompositionen selbst können rekursiv zu größeren Einheiten zusammengesetzt werden. Aus technischer Sicht besteht kein Unterschied zwischen der Schnittstelle einer Webservice-Komposition als BPEL-Artefakt oder einem elementaren Webservice. Beide werden mit WSDL beschrieben. Die Umsetzung solcher Webservice-Architekturen findet in der Praxis bereits statt und wurde auch durch den Autor in [EM+05] und [EW+06] beschrieben. Dies stellt den Ausgangspunkt für die Verknüpfung der Fachfunktionalität mit Zugriffskontrolle dar.

4.1.1.2 Anforderungen an die Zugriffskontroll-Architektur

Aufbauend auf der Spezifikation der zu schützenden Objekte werden nachfolgend die Anforderungen an eine Erweiterung um Zugriffskontrolle motiviert.

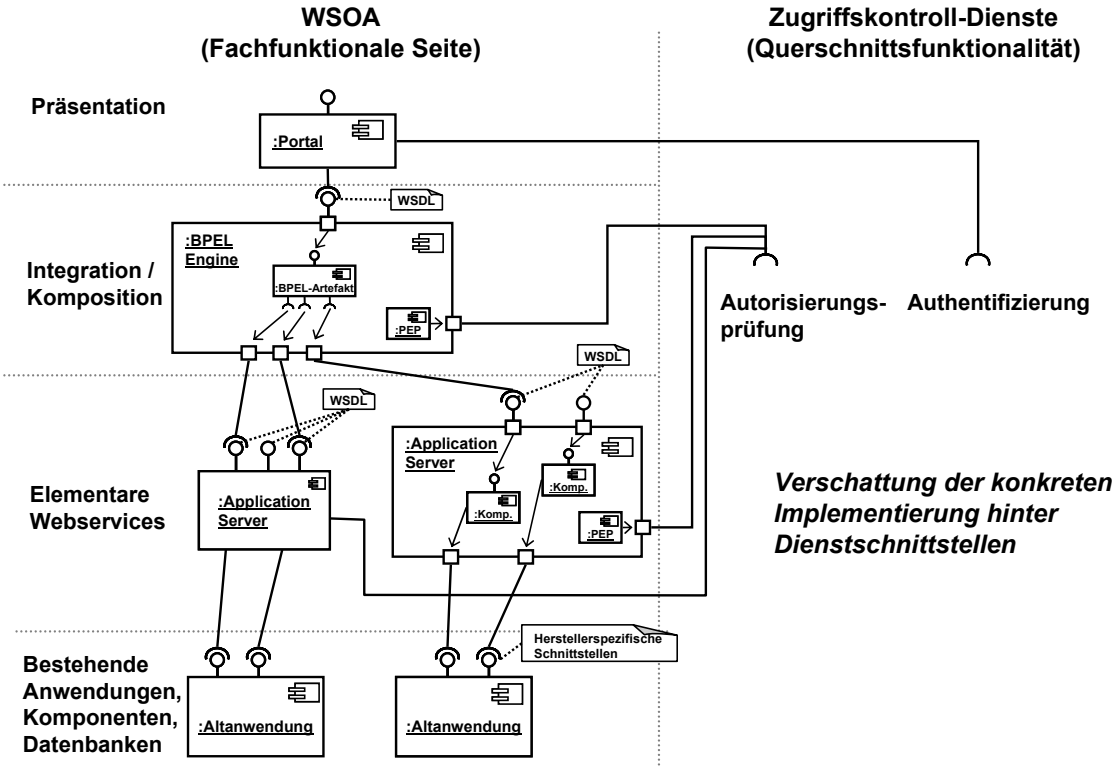


Abbildung 25: Verknüpfung einer WSOA mit Zugriffskontroll-Diensten

Abbildung 25 zeigt links auf der fachfunktionalen Seite eine Instantiierung der vorgenannten Struktur, modelliert mit der UML. Dargestellt sind zwei Altanwendungen, deren Fachfunktionalität über herstellerspezifische Schnittstellen nutzbar ist. Komponenten im jeweiligen *Application Server* bilden diese Schnittstellen ab, wozu sich das Entwurfsmuster „Stellvertreter“ (engl. *Proxy*) eignet, wie in Kapitel 3.2.1.2 beschrieben. Jeder *Application Server* hält neben diesen Komponenten auch übergreifende Basiskomponenten bereit, die von allen betriebenen Komponenten genutzt werden können. Ein Beispiel hierfür ist der integrierte Webservice-*Wrapper*, der die Komponenten mit WSDL-konformen Dienstschnittstellen versieht. Ohne den nachfolgenden Ergebnissen zu stark vorgreifen zu wollen, gilt es festzuhalten, dass im *Application Server* die Möglichkeit zur Durchführung einer Autorisierungsprüfung vorgesehen sein muss. Dieser Sachverhalt ist in Abbildung 25 durch die PEP-Komponente modelliert, die in jeden *Application Server* „eingesteckt“ wird. Dies soll die Portabilität des

PEP ermöglichen (vgl. Anforderung A2.2). Dies gilt analog auch für die BPEL-*Engines*, die für die Dienstkomposition eingesetzt werden. Wie diese PEP-Komponente in den *Application Server* eingebunden und wie sie mit den fachlichen Webservices verbunden ist, wird an dieser Stelle bewusst offen gelassen und in Kapitel 4.2 behandelt. Wichtig für die lose Kopplung zwischen fachfunktionaler Seite und den Zugriffskontroll-Diensten ist, dass diese über interoperable Dienstschnittstellen verfügen (vgl. Anforderung A2.1).

Um eine effektive Zugriffskontrolle für die durch die *Application Server* und BPEL-*Engines* betriebenen Webservices zu ermöglichen, sind Erweiterungen notwendig. Wie in Kapitel 2.2.5 dargestellt, besteht der Prozess der Zugriffskontrolle aus den Teilschritten Identifizierung, Authentifizierung und der eigentlichen Zugriffsentscheidung, also der Autorisierungsprüfung. Identifizierung und Authentifizierung werden häufig als zusammengefasster Prozess durchgeführt, der der Autorisierungsprüfung vorgelagert ist. Dies lässt sich auch auf WSOA übertragen, wobei ein Webservice selbst weder Kenntnis hat, noch Kenntnis haben soll, in welchem Nutzungskontext er aufgerufen wird. Unter dem Nutzungskontext kann der Geschäftsprozess verstanden werden, während dessen Ausführung der Webservice genutzt wird. Aus Sicht eines elementaren wie auch eines komponierten Webservice findet ein Dienstaufruf statt, ohne dass der konkrete und übergeordnete Nutzungskontext bekannt ist. Dies ist Teil des Konzepts der losen Kopplung, wie in Kapitel 2.1.3 beschrieben. Die notwendigerweise durchzuführende Zugriffskontrolle sollte daher aus dem Webservice ausgelagert werden. Eine Auslagerung im Sinne einer dienstorientierten Zugriffskontrolle bedeutet aus Sicht des Webservices, eine Anfrage an einen Zugriffskontroll-Dienst zu stellen mit dem informell spezifizierten Inhalt: „Darf ich dem vorliegenden Zugriffsversuch stattgeben?“ Als Ergebnis aus Sicht des anfragenden, fachfunktionalen Webservices ist eine boolesche Antwort ausreichend.

Aus Sicht der WSOA sind zwei Dienste zu spezifizieren, um eine querschnittlich nutzbare Zugriffskontroll-Architektur für WSOA bereitzustellen: ein Authentifizierungs- und ein Autorisierungsprüfungsdienst. Wenn der Authentifizierungsdienst zeitlich vorgelagert eingesetzt wird, beispielsweise durch eine Portal-Integration, kann nach erfolgreicher Authentifizierung dem Benutzer ein temporär gültiges Sicherheitstoken ausgestellt werden. Dieses kann für alle Dienstaufrufe innerhalb eines definierten Zeitfensters übertragen werden und stellt so die Grundlage für eine Ende-zu-Ende-Authentifizierung des aufrufenden Benutzers

dar, wie in Abbildung 19 in Kapitel 3.2.2.1 dargestellt. Um die in Kapitel 3.1 definierten Anforderungen zu erfüllen, sind folgende Sachverhalte zu beachten: Die Zugriffskontroll-Architektur muss allgemeingültig für WSOA einsetzbar sein (Anforderung A1.1) und muss die Einbindung des Fachentwicklers ermöglichen (Anforderung A1.2). Die vorgenannten Dienste müssen an stabil zu haltenden Schnittstellen ausgeprägt werden, die mit WSDL beschrieben werden und müssen SOAP als Basisprotokoll für den Nachrichtenaustausch unterstützen (A2.1). Dabei gilt anzumerken, dass die zu übermittelnden Daten für eine Autorisierungsprüfung im Webservice-Kontext noch nicht im Stand der Forschung und Technik spezifiziert wurden. In Kapitel 4.1.2.1 wird ein geeigneter Schnittstellenentwurf vorgestellt. Hinsichtlich der geforderten Portabilität (A2.2) stellt der *Policy Enforcement Point* die Komponente mit dem maßgeblichen Einfluss dar. In dieser Arbeit wurde dafür mit dem *Secure Service Agent* ein Entwurfsmuster entwickelt, das in Kapitel 4.2 vorgestellt wird. Die Dienstschnittstellen der Zugriffskontroll-Architektur müssen so spezifiziert werden, dass sich bestehende, kommerzielle Produkte integrieren lassen (Anforderung A2.3).

4.1.1.3 Abzubildende Prozesse für Zugriffskontrolle

Nachfolgend wird in UML-Aktivitätsdiagrammen der Analyse-Phase das Ablauf- und Objektflussverhalten für die Prozesse „Benutzerauthentifizierung“ und „Autorisierungsprüfung“ vorgestellt. Ziel dabei ist es, die vorhandene Komplexität vor den Benutzern zu verschatten, der bestehende Technologien wie beispielsweise Webbrowser weiterverwenden können sollte. Ausgangspunkt des in Abbildung 26 vorgestellten Ablaufs zur Benutzerauthentifizierung ist, dass sich auf der fachfunktionalen Seite der WSOA (links dargestellt) ein Benutzer authentisieren möchte. Hierzu übermittelt er seinen Identifikator und den zugehörigen Berechtigungsnachweis (engl. *Credentials*). Dies könnten beispielsweise ein Benutzername als Identifikator und das zugehörige Passwort als Berechtigungsnachweis darstellen (vgl. Kapitel 2.2.3). Durch die strikte Trennung der Zuständigkeiten (engl. *Separation of Concerns*) in der Architektur werden Identifikator und Berechtigungsnachweis von der Fachseite nach rein syntaktischer Überprüfung an die Zugriffskontroll-Architektur übergeben. Dort wird eine Authentifizierung gegen die lokal vorgehaltenen Benutzerdaten vorgenommen. Im Fehlerfall wird eine entsprechende Nachricht an die Fachseite zurückgegeben. Bei erfolgreicher Authentifizierung wird ein temporär gültiges Sicherheitstoken erzeugt und die Abbildung zwischen Sicherheitstoken und dem eigentlichen

Identifikator innerhalb der Zugriffskontroll-Architektur vermerkt. Das Token an sich soll keinerlei Daten tragen, sondern soll lediglich eine temporär gültige Referenz auf einen Identifikator sein. Dies genügt, damit die Dienste der Zugriffskontroll-Architektur zu jeder Zeit die Abbildung auf den eigentlichen Identifikator durchführen können.

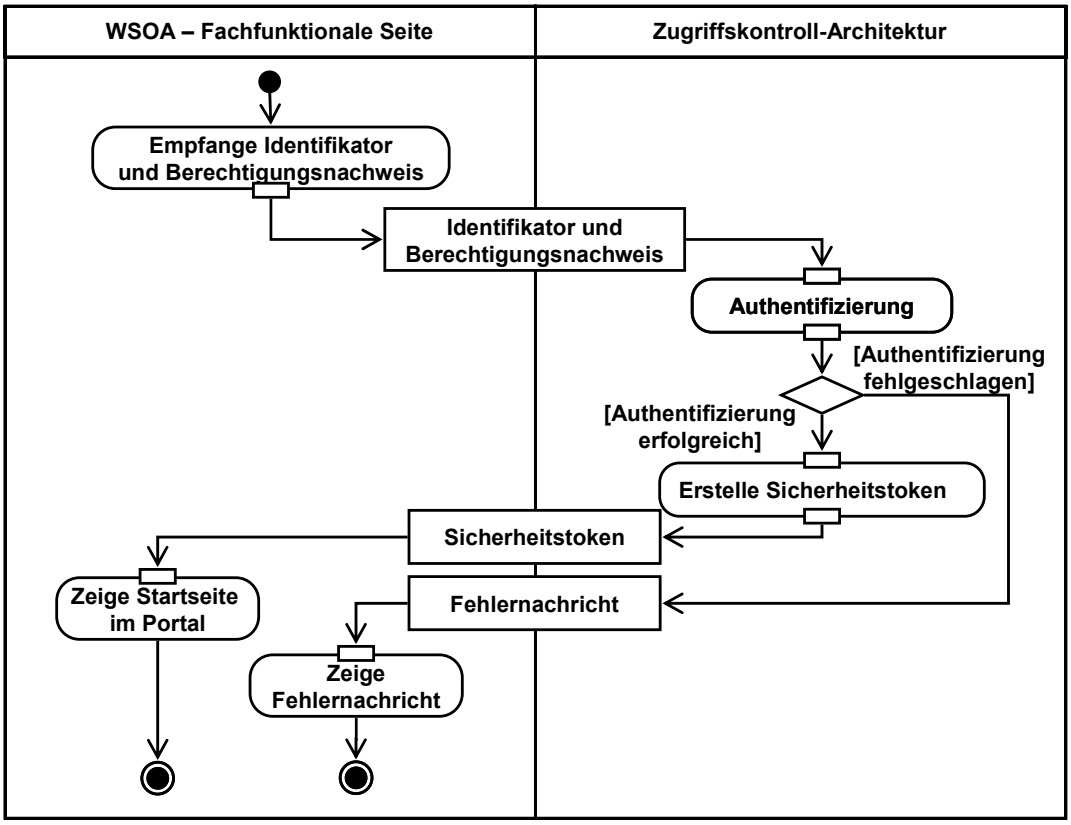


Abbildung 26: Aktivitätsdiagramm für die Benutzerauthentifizierung

Dieses Sicherheitstoken wird dann an die Fachseite zurückgegeben, die es über bestehende Technologien wie *Cookies* dem Benutzer transparent übermitteln kann. Durch die Nutzung von *Cookies* besteht kein Änderungsbedarf hinsichtlich der eingesetzten Produkte und Technologien beim Benutzer und es ist zudem sichergestellt, dass das Sicherheitstoken bei jedem Aufruf automatisch übermittelt wird. Die Nutzung eines solchen Tokens hat verschiedene Vorteile: Zunächst wird dadurch ein WSOA-weites *Single Sign-On* (SSO) ermöglicht. Der Benutzer muss nicht bei jedem Dienstaufruf erneut die sensiblen Authentifizierungsdaten seiner digitalen Identität übermitteln. Durch die zeitliche Begrenzung der Gültigkeit des Sicherheitstokens, die in der Abbildungstabelle innerhalb der Zugriffskontroll-Architektur gespeichert werden sollte und nicht im Token selbst, ist sichergestellt, dass für den Fall des Aufdeckens oder des Verlusts eines

Tokens inhärent eine zeitliche Begrenzung des Missbrauchpotenzials gegeben ist. Auch aus Datenschutzsicht ergeben sich Vorteile: Weil bei jeder erfolgreichen Authentifizierung ein neues Sicherheitstoken erstellt wird, hat ein fachfunktionaler Dienst weder die Möglichkeit zu erkennen, von welchem Benutzer er aufgerufen wird, noch ob er immer vom selben oder von unterschiedlichen Benutzern angefragt wird. Eine erfolgreiche Authentifizierung und damit der Besitz eines gültigen Sicherheitstokens ist die Vorbedingung, dass ein Benutzer einen Webservice aufrufen kann. Dabei ist es unerheblich, ob es sich um einen elementaren oder komponierten Webservice handelt. Während die Authentifizierung als vorgelagerter Prozess einmalig für den Gültigkeitszeitraum des Sicherheitstokens durchgeführt werden muss, muss durch die Atomizität und bewusst geforderte Unabhängigkeit der einzelnen Webservices bei jedem Aufruf eine Autorisierungsprüfung durchgeführt werden. Es genügt insbesondere nicht, diese auf der Ebene der komponierten Dienste oder gar auf Portalebene durchzuführen, da, wie bereits beschrieben, eine strikte Schichtung innerhalb der WSOA nicht gegeben ist und die explizite Definition von Vertrauensbeziehungen zwischen den Webservices untereinander den Grad der losen Kopplung zwischen den Diensten verschlechtert.

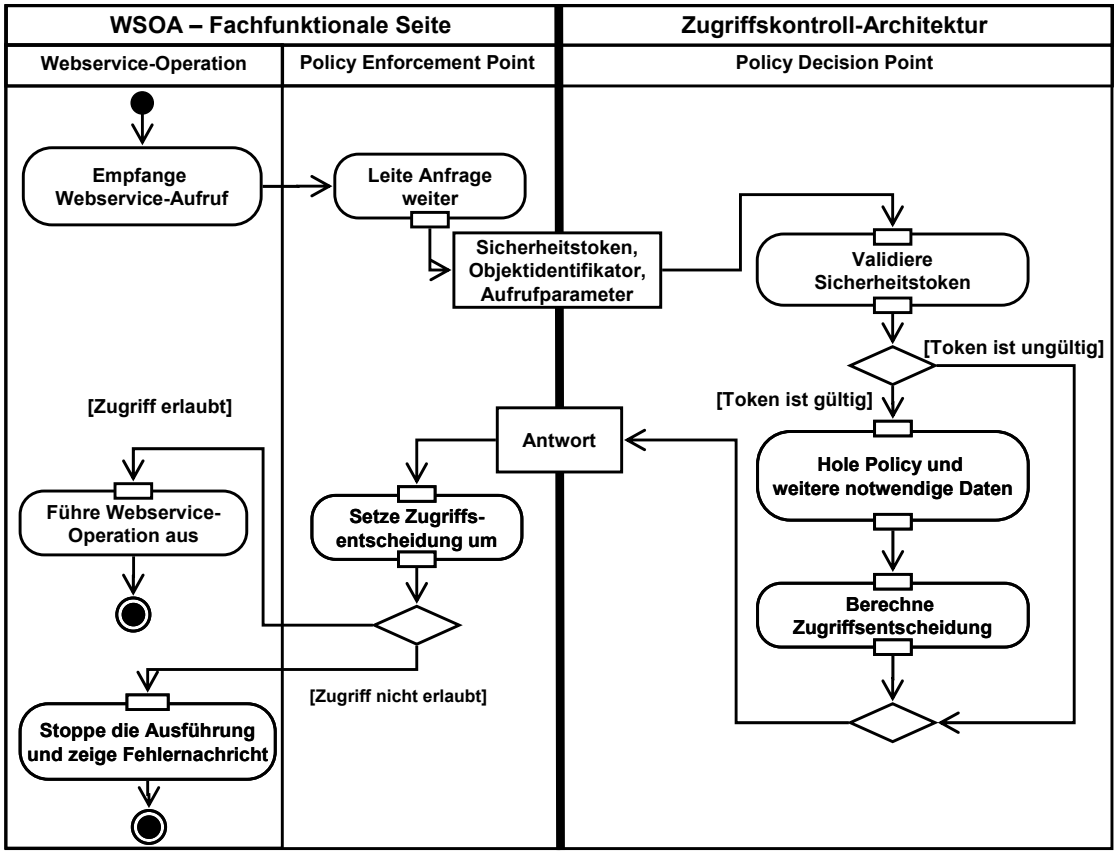


Abbildung 27: Aktivitätsdiagramm für die Autorisierungsprüfung

Die in Abbildung 27 vorgestellte Autorisierungsprüfung beginnt mit dem Aufruf eines Webservices. Beim Aufruf übermittelt der Benutzer sein Sicherheitstoken und entsprechende Aufrufparameter des Webservices (vgl. Kapitel 5). Diese Daten werden an den *Policy Enforcement Point* (PEP) weitergegeben, der in jedem *Application Server* zu platzieren ist. Die genaue Spezifikation des PEP wird in Kapitel 4.2 vorgestellt. Der PEP führt die Kommunikation mit dem extern platzierten Autorisierungsprüfungsdienst durch, der durch die Komponente *Policy Decision Point* (PDP) realisiert wird. Dieser überprüft zunächst das Sicherheitstoken auf zeitliche Gültigkeit. Für den Fall eines positiven Validierungsergebnisses wird die passende Zugriffskontroll-Policy gesucht sowie weitere notwendige Daten des aufrufenden Benutzers geladen. Die Struktur der Policies stellt einen wesentlichen Baustein einer Zugriffskontroll-Architektur dar und wird in Kapitel 5 behandelt. Aufgrund dieser Daten wird die Zugriffsentscheidung berechnet und als Ergebnis ein boolescher Wert an den PEP übermittelt. Dieser gibt die Entscheidung an die Fachkomponente weiter, die die angefragte Operation dann entsprechend ausführt oder eine Fehlernachricht zurückgibt.

4.1.2 Entwurf

Die im vorangegangenen Kapitel motivierten Anforderungen an die Architektur und die Definition der umzusetzenden Prozesse werden nun für den Entwurf einer geeigneten Zugriffskontroll-Architektur genutzt, die der Autor in [EB+07b] vorgestellt hat.

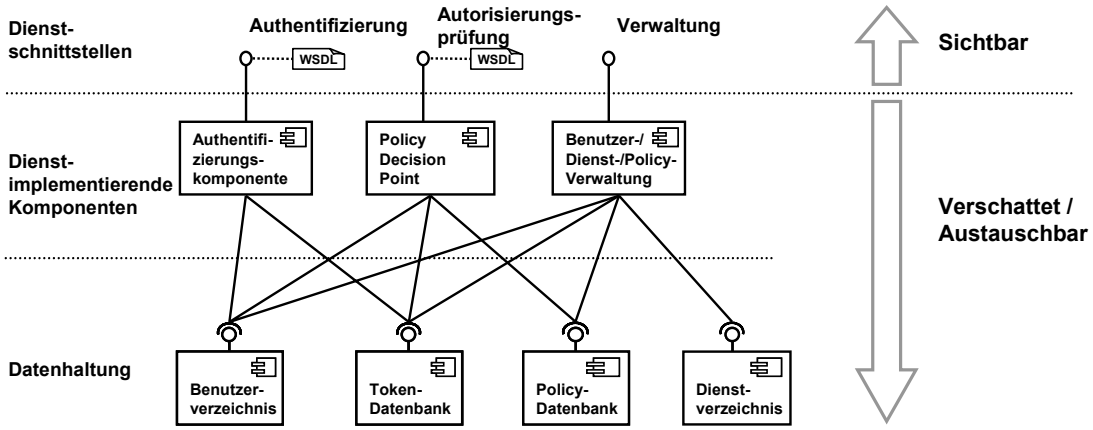


Abbildung 28: Entwurf einer dienstorientierten Zugriffskontroll-Architektur

Die Zugriffskontroll-Architektur lässt sich in drei Schichten einteilen, wie in Abbildung 28 dargestellt. Die oberste und gegenüber der Facharchitektur der WSOA einzige sichtbare Schicht stellt die Dienstschnittstellen bereit. Neben den in der Analyse geforderten Diensten für Authentifizierung und Autorisierungsprüfung ist für Administrationszwecke eine Verwaltungsschnittstelle bereitzustellen. Unterhalb der Dienstschnittstellen und damit für die Facharchitektur der WSOA verschattet befinden sich die Dienst-implementierenden Komponenten. Diese arbeiten auf den Daten, die durch die Datenhaltungsschicht bereitgestellt werden. Die Fokussierung des Fachentwicklers auf die reine Nutzung der Dienstschnittstellen, insbesondere die Autorisierungsprüfung, reduziert die entstehende Komplexität (Anforderung A1.2) und ermöglicht die grundsätzliche Integrationsfähigkeit bestehender Sicherheitsprodukte (A 2.3). Nachfolgend wird eine Spezifikation der einzelnen Bausteine gegeben.

4.1.2.1 Schnittstellen und Dienst-implementierende Komponenten

Die Aufgabe des Authentifizierungsdienstes ist es, nach erfolgreicher Benutzerauthentifizierung ein Sicherheitstoken auszustellen. Zur Authentifizierung wird die Operation `authenticateUser` an der Dienstschnittstelle angeboten. Diese kann je nach Authentifizierungsmöglichkeiten vielfach überladen sein.

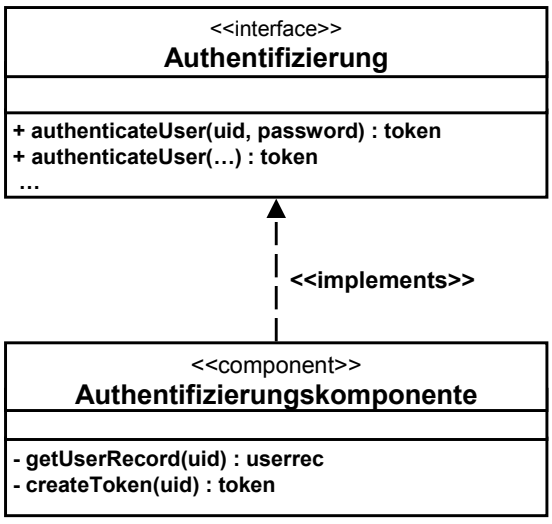


Abbildung 29: Klassendiagramm der Authentifizierungskomponente

Abbildung 29 zeigt die Dienstschnittstelle des Authentifizierungsdienstes sowie die Dienst-implementierende Authentifizierungskomponente, dargestellt als UML-Klassendiagramm. Nur die öffentliche Methode `authenticateUser`

wird an der Dienstschnittstelle als Webservice-Operation bereitgestellt. Die erste hier spezifizierte Methode stellt die Authentifizierung mittels des Benutzernamens (Identifikator) und des zugehörigen Passworts (Berechtigungsnachweis) dar. Zur Realisierung weiterer Authentifizierungsmechanismen ist diese Methode wie angedeutet überladbar. Das Ergebnis der Methode `authenticateUser` bei erfolgreicher Authentifizierung ist jedoch in allen Fällen ein Sicherheitstoken; bei nicht erfolgreicher Authentifizierung wird eine Fehlermeldung zurückgegeben. Die weiteren Methoden sind Hilfsmethoden und nur durch die Komponente selbst nutzbar. Sie müssen daher nicht zwingend umgesetzt werden. Der Eintrag des Benutzers mit der eindeutigen Kennung `uid` wird durch den Aufruf der Methode `getUserRecord` aus dem Benutzerverzeichnis abgerufen und im lokalen Variablen-Vektor `userrec` gespeichert. Die Methode `createToken` erzeugt ein neues, temporäres Sicherheitstoken für den Benutzer und liefert es zurück.

Die Autorisierungsprüfung wird durch den `PolicyDecisionPoint` durchgeführt. Er prüft, ob ein Benutzer auf eine Webservice-Operation zugreifen darf. Abbildung 30 zeigt das UML-Klassendiagramm des Autorisierungsprüfungsdienstes und der Dienst-implementierenden Komponente `PolicyDecisionPoint`. Es wird wieder nur die öffentliche Methode (`validateAuthorization`) an der Dienstschnittstelle als Webservice-Operation bereitgestellt. Die Methode `validateAuthorization` nimmt die Daten des `PolicyEnforcementPoint` entgegen und ruft die nachfolgend beschriebenen Hilfsmethoden auf. Das Ergebnis der durch die Hilfsmethode `evaluateAccessControlDecision` getroffenen Zugriffskontroll-Entscheidung wird an den PEP zurückgesendet. Die Methode `evaluateAccessControlDecision` überprüft an Hand einer vorgegebenen Policy, ob der Benutzer auf die angegebene Webservice-Operation zugreifen darf. Zur Berechnung der Zugriffskontroll-Entscheidung können die vom PEP übergebenen Aufrufparameter, die aus dem Benutzerverzeichnis abgerufenen Attribute des Benutzers und der Umgebungszustand berücksichtigt werden. Diese Daten entsprechen dem Zugriffskontroll-Modell, das in Kapitel 5 vorgestellt wird.

Die Methode `getTokenInformation` ruft die Daten des Sicherheitstoken aus der Token-Datenbank ab. Unter anderem ist hier der Identifikator des Benutzers enthalten. Die Methode `isTokenValid` überprüft die Gültigkeit der mit `getTokenInformation` abgerufenen Daten und liefert als Ergebnis einen

booleschen Ausdruck zurück. Ein Token ist gültig, wenn es syntaktisch korrekt ist und wenn seine zeitliche Gültigkeit noch nicht abgelaufen ist. Der Eintrag des Benutzers mit dem Identifikator `uid` wird durch den Aufruf von `getUser-Record` aus dem Benutzerverzeichnis abgerufen. Die mit der Webservice-Operation verbundene Zugriffskontroll-Policy wird mit Hilfe der Methode `getPolicy` abgerufen. `getEnvironmentState` ruft Informationen zum aktuellen Umgebungszustand ab. Dazu können zum Beispiel das aktuelle Datum und die Uhrzeit gehören.

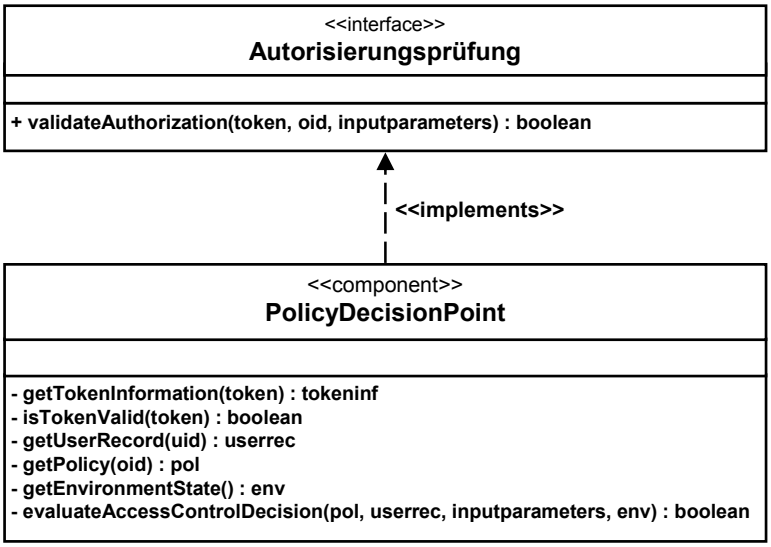


Abbildung 30: Klassendiagramm des PolicyDecisionPoint

Die Ausprägung der Authentifizierung und Autorisierungsprüfung als Dienst wird unter dem Begriff *Identity as a Service* zusammengefasst [Em07]. Die beiden vorgenannten Dienste sollten als Webservices betrieben und damit mit WSDL-konformen Schnittstellen beschrieben werden. Dies bildet die Grundlage für ihre interoperable Nutzung (Anforderung A2.1). Zur Absicherung der Kommunikationsbeziehung zwischen Fach- und Zugriffskontroll-Architektur bietet sich der Einsatz von WS-Security an (vgl. Kapitel 3.2.2.1). WS-Security sollte dabei einen Punkt-zu-Punkt-Sicherheitskontext zwischen aufrufendem *Application Server* und den Diensten der Zugriffskontroll-Architektur ermöglichen. Die begrenzte Anzahl an *Application Server* innerhalb eines Unternehmens erlaubt den Einsatz einer *Public Key Infrastructure* (PKI), um jeden *Application Server* mit einem Schlüsselpaar zu versehen. Die PKI wird dabei nur zur Erstellung des Punkt-zu-Punkt-Sicherheitskontext genutzt, nicht jedoch für die Ende-zu-Ende-Sicherheit vom Benutzer zum Webservice. Durch die Nutzung von WS-SecurityPolicy ist es möglich, die Notwendigkeit der Verwendung des zertifi-

katsbasiertem Einsatzes von WS-Security für die Authentifizierung und Autorisierungsprüfung zu spezifizieren. Zusätzlich ermöglicht dies die Reduktion der Dienstverfügbarkeit auf durch die PKI zertifizierte *Application Server*. Dies kann *Denial of Service*-Angriffe erschweren.

Die beiden vorgenannten Dienste sollten als Webservices mit WSDL-konformen Schnittstellen beschrieben werden. Die bereitzustellende Verwaltungsschnittstelle bietet Administratoren die Möglichkeit, Einträge in den Datenquellen anzulegen, zu verändern und zu löschen. Zur Fehlerbehebung oder zu Testzwecken sollte einem Administrator insbesondere die Möglichkeit eingeräumt werden, Einträge in der Token-Datenbank anzulegen, zu bearbeiten oder zu löschen. Neben diesen von Administratoren manuell ausgeführten Tätigkeiten sollten auch automatisierte Prozesse ablaufen, wie beispielsweise die regelmäßige Bereinigung der Token-Datenbank oder die Synchronisation von Benutzerverzeichnissen übernehmen.

4.1.2.2 Datenhaltung

Benutzerverzeichnis

Das Benutzerverzeichnis repräsentiert die Datenhaltung für die digitalen Identitäten der Benutzer. Diese bestehen, wie in Kapitel 2.2.3 beschrieben, aus dem Identifikator, den Berechtigungsnachweisen und den zugewiesenen Zugriffsberechtigungen. Das Benutzerverzeichnis muss gegenüber den Dienstimplementierenden Komponenten der Zugriffskontroll-Architektur eine einheitliche Schnittstelle aufweisen, kann jedoch durchaus intern ebenfalls als Verzeichnisverbund wie beispielsweise einem Meta- oder virtuellen Verzeichnis aufgebaut sein. Für die Speicherung von Benutzerdaten bieten sich klassische LDAP-basierte Verzeichnisse an, wobei für die Einbeziehung der zur Policy-Auswertung relevanten Zugriffsberechtigungen ein gemeinsames Datenschema und Vokabular zu definieren ist. Dies ist insbesondere bei der Umsetzung einer modellgetriebenen Entwicklung von Zugriffkontroll-Policies von Bedeutung und wird in Kapitel 6 beschrieben. Es ist anzuraten, die Verbindung mit dem LDAP-Verzeichnis zu verschlüsseln, beispielsweise mittels *Transport Layer Security* (TLS). Die Authentifizierungskomponente, der *Policy Decision Point* sowie auch die weiteren Verwaltungskomponenten dürfen nur auf das Benutzerverzeichnis zugreifen, wenn sie sich mit Hilfe eines technischen Benutzers und des zugehöri-

gen Passworts authentifiziert haben. Dies ermöglicht eine lose Kopplung bei gleichzeitiger, intern durchgeführter Zugriffskontrolle.

Token-Datenbank

Nach erfolgreicher Authentifizierung eines Benutzers durch die Authentifizierungskomponente wird ein neuer Eintrag in der Token-Datenbank erzeugt. Er enthält Informationen über die aktuelle Sitzung des Benutzers. Jeder Eintrag in der Datenbank besteht aus drei Werten. Als Primärschlüssel wird eine eindeutige und hinreichend kollisionsresistente Token-ID. Diese kann beispielsweise mit gängigen Hash-Funktionen erzeugt werden. Des Weiteren wird der Identifikator des authentifizierten Benutzers gespeichert. Würde es sich beim Benutzerverzeichnis um eine relationale Datenbank handeln, könnte man den Identifikator des Benutzers als Fremdschlüssel bezeichnen. Als dritter Wert wird der Erstellungszeitpunkt des Eintrags gespeichert. Dies ermöglicht ein automatisches Invalidieren des Tokens nach vorgegebener Zeit. Es besteht auch die Möglichkeit, bei jeder Token-Validierung den Erstellungszeitpunkt zu aktualisieren, was einen fixen Ablauf des Tokens verhindert und eine Invalidierung nach einer spezifizierten Zeit der Untätigkeit des Benutzers (engl. *Time-out*) ermöglicht. Jede der Dienst-implementierenden Komponenten greift auf die Token-Datenbank mit einem unterschiedlichen technischen Benutzer zu. Dies ermöglicht die Vergabe feingranularer Zugriffsberechtigungen. Die Authentifizierungskomponente darf nur schreibend zugreifen (SQL-Befehl `INSERT`), während der *Policy Decision Point* ausschließlich lesenden Zugriff hat (SQL-Befehl `SELECT`). Er muss bestehende Einträge in der Datenbank nur dann aktualisieren können (SQL-Befehl `UPDATE`), wenn die Gültigkeit des Tokens des Benutzers nach einer erfolgreich durchgeführten Autorisierungsprüfung um einen bestimmten Zeitraum verlängert werden soll.

Policy-Datenbank

Der Einsatz von Webservices soll die Wiederverwendbarkeit von Fachfunktionalität in verschiedenen Nutzungskontexten (Geschäftsprozessen) vereinfachen. In der Policy-Datenbank sind für jeden Webservice, genauer gesagt für jede Webservice-Operation, die gültigen Zugriffskontroll-Policies hinterlegt. Grundlage der Policies ist ein geeignetes Zugriffskontroll-Modell sowie eine zugehörige

Policy-Sprache. Beides wird in Kapitel 5 vorgestellt, weshalb an dieser Stelle auf weitergehende Ausführungen verzichtet wird.

Dienstverzeichnis

Das Dienstverzeichnis stellt eine zentrale, querschnittlich genutzte Komponente einer dienstorientierten Architektur dar (siehe Kapitel 2.1.3). Hier werden Metainformationen zu den elementaren Webservices und auch den komponierten BPEL-Prozessen gespeichert. Das Dienstverzeichnis sollte unmittelbar mit der Zugriffskontroll-Architektur verknüpft sein, da die Zugriffskontroll-Policies ebenfalls Metainformationen zu einem Dienst darstellen. Eine Verknüpfung ist insbesondere für das in Kapitel 6 vorgestellte, modellgetriebene Vorgehen zur Entwicklung von Zugriffskontroll-Policies wichtig. Das Ziel ist die Erweiterung der Daten im Dienstverzeichnis um Informationen über die eindeutige Kennung (engl. *Object Identifier*, OID) jeder Webservice-Operation. Mit Hilfe dieser Kennung wird eine in der Policy-Datenbank hinterlegte Zugriffskontroll-Policy mit einer Webservice-Operation oder einer Webservice-basierten Dienstkomposition verknüpft.

4.1.2.3 Die Ablauflogik für die Authentifizierung und Autorisierungsprüfung

Nachfolgend wird in UML-Sequenzdiagrammen der dynamische Teil der Architektur spezifiziert. Es werden die Interaktionen zwischen den beteiligten Komponenten durch den Austausch von Nachrichten beschrieben. Abbildung 31 stellt die Interaktionssequenz für die Benutzerauthentifizierung dar. Sowohl die Verwendung von klassischen Benutzername/Passwort-basierten Authentifizierungsverfahren als auch der Einsatz von zertifikatsbasierter Authentifizierung erlaubt den Einsatz traditioneller Webbrowser auf der Seite des Benutzers. Das erzeugte Sicherheitstoken wird in der SOAP-Nachricht transportiert und erst auf dem Kommukationspfad zwischen Portal und Webbrowser des Benutzers über ein *Cookie* transportiert. Dies erlaubt den Einsatz bestehender Software insbesondere auf der Seite des Benutzers. Mittels eines (unmodifizierten) Webbrowsers verbindet sich der Benutzer mit dem Portal, das den zentralen Dienstzugangspunkt darstellt. Hier werden die Identität des Benutzers sowie seine Berechtigungsnachweise entgegen genommen. Das Portal stellt dann eine

Authentifizierungsanfrage an den Authentifizierungsdienst, implementiert durch die Authentifizierungskomponente der Zugriffskontroll-Architektur.

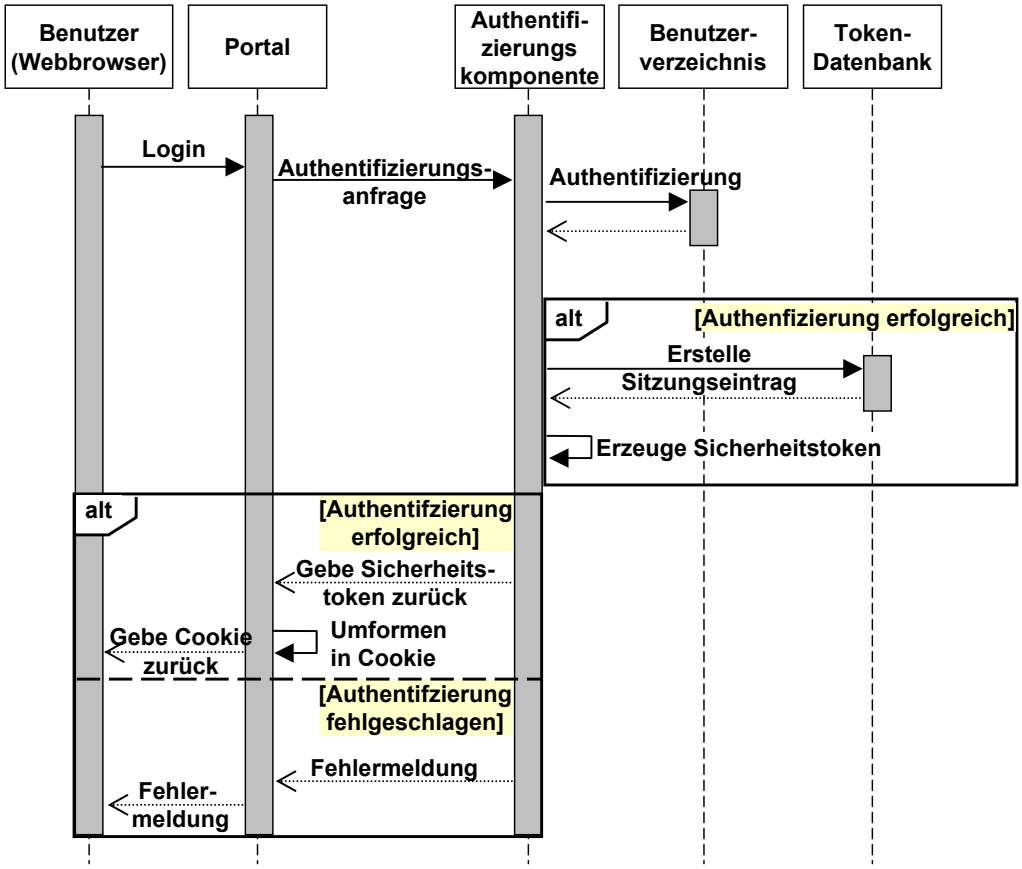


Abbildung 31: Sequenzdiagramm für die Benutzerauthentifizierung

Bei positiver Authentifizierung wird ein Sicherheitstoken erstellt, das als temporäres Authentifizierungselement verwendet wird. Dieses wird über das Portal an den Benutzer zurückgemeldet, wobei sich als Transportmedium der Einsatz von *Cookies* empfiehlt. Zur Verschlüsselung der Kommunikationsbeziehung zwischen Benutzer und Portal können klassische Transportverschlüsselungsmechanismen mit SSL/TLS eingesetzt werden, die üblicherweise bereits sowohl in den Webbrowsern als auch der gängigen Portal-Software implementiert ist. Aus Sicht des Benutzers wird die Migration in dienstorientierte Architekturen dadurch verschattet.

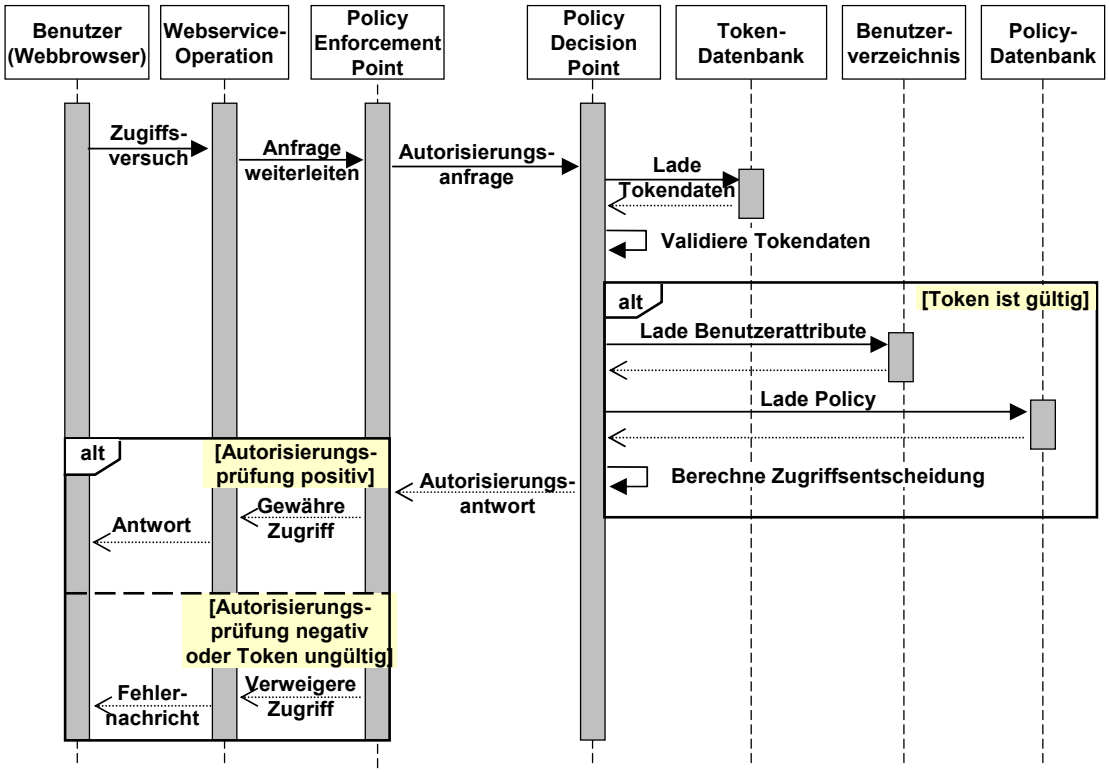


Abbildung 32: Sequenzdiagramm für die Autorisierungsprüfung

Abbildung 32 zeigt das Interaktionsverhalten für die bei jedem Dienstaufruf durchzuführende Autorisierungsprüfung. Die durch den *Policy Decision Point* durchgeführte Zugriffsentscheidung hängt maßgeblich vom eingesetzten Zugriffskontroll-Modell ab, das in Kapitel 5 eingeführt wird.

4.2 Das Entwurfsmuster Secure Service Agent

Die Verknüpfung der Zugriffskontroll-Architektur mit der Fachseite der WSOA wird durch den *Policy Enforcement Point* (PEP) durchgeführt. Bestehende Realisierungsformen für *Policy Enforcement Points* wie der *Intercepting Web Agent* (IWA) und der *Secure Service Proxy* (SSP) wurden in Kapitel 3.2.1.3 vorgestellt und bewertet. Es wurde gezeigt, dass weder IWA noch SSP eine ideale Lösung als PEP in WSOA-Umgebungen darstellen und insbesondere den in Kapitel 3.1 aufgestellten Anforderungen nicht genügen. Daher wurde durch den Autor mit dem *Secure Service Agent* (SSA) ein Muster entwickelt, das sich besser in die WSOA einfügt [ES+06].

4.2.1 Motivation und Anforderung

Wie bereits im Entwurf der Zugriffskontroll-Architektur spezifiziert wurde, ist eine wichtige Eigenschaft die klare Trennung von PEP und PDP. Dies bedeutet, dass die Entscheidung von Autorisierungsanfragen und ihre nachgelagerte Umsetzung an verschiedenen Stellen durchgeführt werden. Innerhalb einer Sicherheitsdomäne, beispielsweise einem Unternehmen, genügt daher der Betrieb eines einzelnen Autorisierungsprüfungsdiensts. Dieser wird durch einen gemeinsamen *Policy Decision Point* implementiert, der die Anfragen aller PEPs bearbeitet. Der *Secure Service Agent* soll dabei die positiven Eigenschaften der Entwurfsmuster *Secure Service Proxy* (SSP) und *Intercepting Web Agent* (IWA) aufgreifen. Folgende Ziele lassen sich daraus ableiten:

- Grundsätzlich soll es ermöglicht werden, bestehende Anwendungssysteme ohne Änderungen an deren Fachfunktionalität in eine WSOA zu integrieren (Anforderung A2.3). Hierzu bietet sich das Stellvertreter-Muster (engl. *Proxy*) an, wie es im SSP umgesetzt wird.

- Ebenso soll es möglich sein, die implementierten Komponenten so portabel wie möglich zu gestalten, was der Anforderung A2.2 entspricht. Die Portabilität ist hinsichtlich des Kopplungsgrades zwischen Fachkomponente und *Application Server* zu messen. Dies wird ebenfalls durch das Stellvertreter-Entwurfsmuster ermöglicht.

- Andererseits soll Zugriffskontroll-Funktionalität kapselbar sein, wie es durch den IWA ermöglicht wird. Der IWA erlaubt es im Gegensatz zum SSP, lediglich an einer Stelle im *Application Server* Zugriffskontroll-Funktionalität zu platzieren. Es besteht nicht die Notwendigkeit, jede einzelne Fachkomponente mit redundantem Sicherheitscode zu versehen.

- Ein weiteres Ziel ist es, eine größtmögliche Trennung zwischen Fachfunktionalität und Sicherheitsfunktionalität zu erreichen, wie es beispielsweise durch das Konzept des IWA ermöglicht wird.

- Ein WSOA-weite gültiges *Single Sign-On* soll ermöglicht werden. Hierzu muss ein WSOA-Benutzerkontext erstellt werden, der dann nach der Autorisierungsprüfung auf die jeweiligen Altsysteme abgebildet wird.

4.2.2 Entwurf

Die vorgenannten Anforderungen werden nachfolgend präzisiert. Zunächst wird mit einem Kommunikationsdiagramm der Ablauf der Autorisierungsanfrage modelliert, das mittels eines Sequenzdiagramms weiter verfeinert wird. Es schließt sich ein Verteilungsdiagramm an, das die Aufteilung der Funktionalität auf die verschiedenen Knoten hervorhebt.

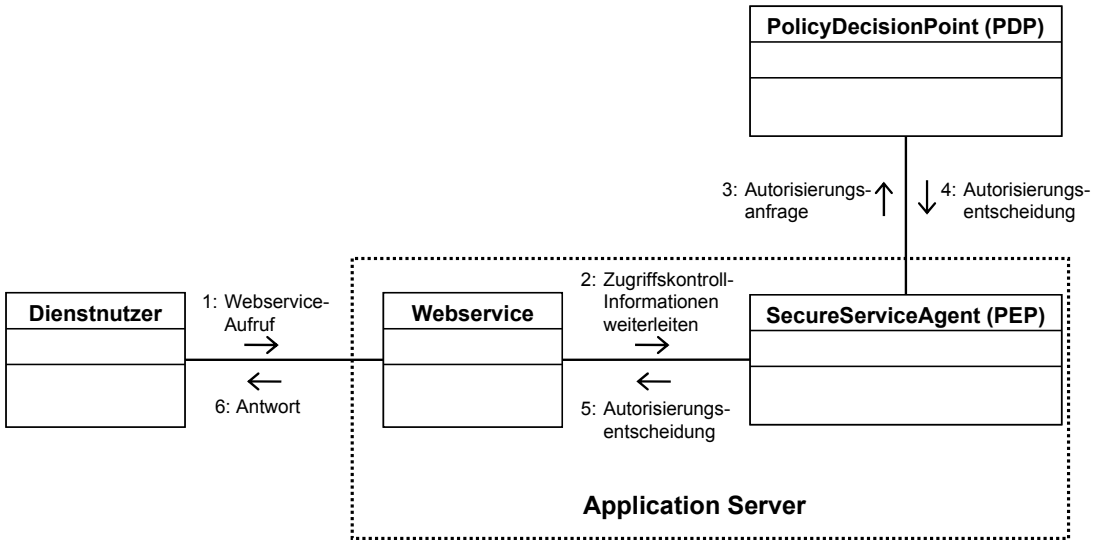


Abbildung 33: Secure Service Agent – Kommunikationsdiagramm

Abbildung 33 veranschaulicht den Aufbau des Entwurfsmusters anhand eines Kommunikationsdiagramms. Im Kern stehen der *Secure Service Agent* als PEP und der *Policy Decision Point* als PDP, die beide physisch voneinander getrennt sind. Auf jedem *Application Server* wird neben einer Anzahl fachfunktionaler Webservices auch jeweils ein *Secure Service Agent* als *Policy Enforcement Point* betrieben, während der *Policy Decision Point* die unternehmensweit verfügbare Dienstschnittstelle zur Autorisierungsprüfung bereitstellt. Der Kommunikationsablauf eines um Zugriffskontrolle erweiterten Webservice-Aufrufs ist wie folgt spezifiziert:

1. Der Dienstnutzer führt einen Webservice-Aufruf durch. Ob hierfür eine direkte Kommunikation stattfindet oder ob Intermediäre wie Portale oder Dienstkompositionen genutzt werden, ist für den weiteren Verlauf unerheblich.

2. Der Webservice erhält diese Anfrage und leitet zunächst alle für die Durchführung einer Autorisierungsprüfung relevanten Daten durch einen generischen Funktionsaufruf innerhalb des *Application Server* an den *Secure Service Agent* weiter.

3. Der *Secure Service Agent* führt eine Vorprüfung hinsichtlich syntaktischer Korrektheit sowie Vollständigkeit durch. Im Erfolgsfall sendet er dann die übermittelten Daten an den Autorisierungsprüfungsdienst, instantiiert durch den *Policy Decision Point*.

4. Der PDP trifft anhand der Anfrage und der ihm vorliegenden Zugriffskontroll-Policies eine Autorisierungsentscheidung.

5. Diese Entscheidung wird dem *Secure Service Agent* mitgeteilt, der diese über interne Kommunikationswege an den zu schützenden Webservice weitergibt. Dieser führt in Abhängigkeit des Ergebnisses den Aufruf durch oder bricht die Abwicklung mit einer Fehlernachricht ab.

6. Für den Fall einer vorliegenden Autorisierung gibt der Webservice das gewünschte Ergebnis zurück. Ansonsten wird eine Fehlernachricht übermittelt.

Bei diesem Aufbau wird der Webservice direkt und damit ohne vorgelagerte Autorisierungsprüfungen vom Dienstnutzer angesprochen. Wenn der Webservice mit Zugriffskontrolle versehen werden soll, muss durch den Fachentwickler die Nutzung des *Secure Service Agent* zum Implementierungszeitpunkt sichergestellt werden, was durch einen generischen Aufruf als ersten Schritt im Fachcode des Webservices durchgeführt wird. Somit wird dem Entwickler ein Werkzeug an die Hand gegeben, mit dem er leicht und unkompliziert fachfunktionale Webservices um Zugriffskontrolle ergänzen kann, ohne sich mit Details der Zugriffskontroll-Architektur sowie der Kommunikation zwischen *Secure Service Agent* als PEP und PDP näher beschäftigen zu müssen.

Das Kommunikationsdiagramm aus Abbildung 33 wird nun durch das Sequenzdiagramm in Abbildung 34 weiter verfeinert. Das zentrale Element in diesem Diagramm ist der Secure Service Agent. Er stellt innerhalb des *Application Server* die Dienste der Zugriffskontroll-Architektur für die dort betriebenen Webservices zur Verfügung.

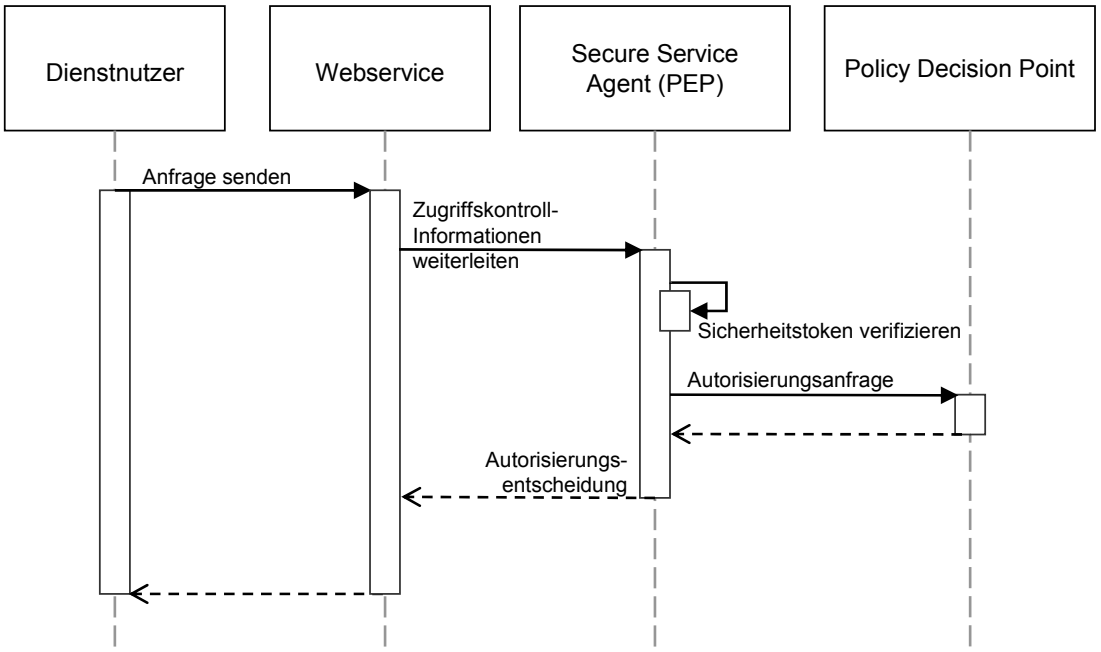


Abbildung 34: Secure Service Agent – Sequenzdiagramm

Um die Dienste des SSA in Anspruch zu nehmen, muss der Webservice einerseits das Sicherheitstoken des aufrufenden Benutzers, seinen Objekt-Identifikator und die übermittelten Aufrufparameter übergeben. Die Übermittlung der Aufrufparameter ermöglicht die Spezifikation feingranularer Zugriffskontroll-Policies. Die Notwendigkeit der Übermittlung wird bei der Einführung des Zugriffskontroll-Modells in Kapitel 5 diskutiert. Bevor der SSA den Aufruf an den extern liegenden Autorisierungsdienst durchführt, führt der SSA zunächst eine syntaktische Überprüfung der übermittelten Daten durch, um vorab mögliche ungültige Anfragen ausfiltern zu können. Wie bereits in Kapitel 4.1 spezifiziert, sollte die Kommunikationsbeziehung zwischen PEP und PDP mittels WS-Security verschlüsselt werden.

Die Schnittstelle des PEP gegenüber den auf demselben *Application Server* betriebenen Webservices ist wie folgt spezifiziert:

```
module SecureServiceAgent
{
interface Webservice_To_SSA
{
  void PEP-Aufruf
    (
    in char[] Sicherheitstoken
```

```
    in string Objekt-Identifikator
    in string[] Aufrufparameter
    out boolean Zugriffsentscheidung
    )
};
```

Der SSA übergibt das durch den Autorisierungsdienst ermittelte Ergebnis an den fachfunktionalen Webservice. Bei positiver Rückmeldung kann der Webservice davon ausgehen, dass der Aufrufer Zugriff auf die entsprechende Funktionalität auch unter Berücksichtigung der übermittelten Aufrufparameter hat. Dies erlaubt ihm, die gewünschte Fachfunktionalität des vertretenen Altsystems ohne weitere Prüfungen aufzurufen. Die nach wie vor vorhandenen Zugriffskontroll-Strukturen des Altsystems können mittels eines technischen Benutzers übergangen werden, da die eigentliche Zugriffskontrolle aus dem Altsystem in die WSOA herausgelöst wurde. Der technische Benutzer kann daher weitreichende Zugriffsberechtigungen besitzen. Eine Abbildung des übergreifenden WSOA-Benutzerkontexts auf Anwendungssystem-spezifische digitale Identitäten ist damit nicht erforderlich, was wiederum Komplexität verringert. Somit realisiert das vorgestellte Entwurfsmuster eine vollständige Entkopplung von Fachfunktionalität und Sicherheitsinfrastruktur in Form der Zugriffskontroll-Architektur, da die Sicherheitsinfrastruktur des Altsystems wie beschrieben vollständig umgangen wird.

Da das Kommunikationsdiagramm und das Sequenzdiagramm nicht alle Aspekte eines solchen Entwurfsmusters vollständig darstellen können, zeigt Abbildung 35 das zugehörige Verteilungsdiagramm. Exemplarisch wird eine fachfunktionale Komponente, die in einem *Application Server* betrieben wird und einen Webservice realisiert, gezeigt. Sie wird über den integrierten Webservice-*Wrapper* an der Außenschnittstelle des *Application Server* als Webservice bereitgestellt. Die Komponente stellt Funktionalität eines Altsystems zur Verfügung. Des Weiteren ist der SSA eingezeichnet, der ebenfalls als Komponente innerhalb des *Application Server* betrieben wird. Seine Funktionalität kann jedoch nur innerhalb des *Server* genutzt werden.

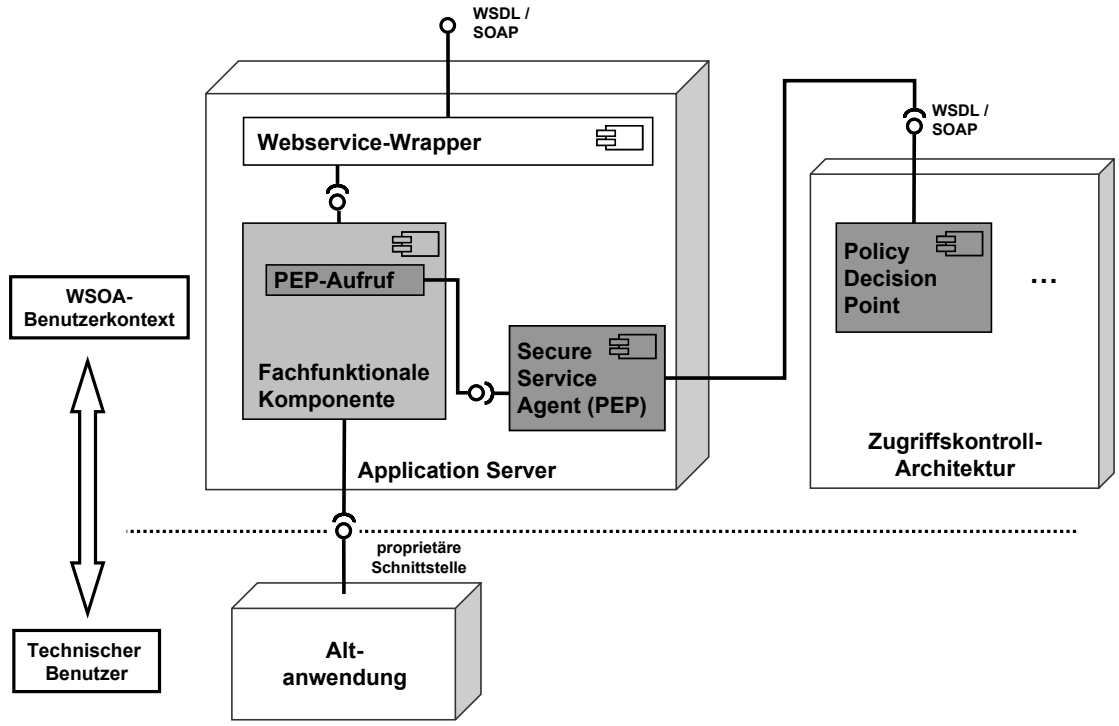


Abbildung 35: Secure Service Agent – Verteilungsdiagramm

Innerhalb der fachfunktionalen Komponenten ist ein Element mit der Bezeichnung „PEP-Aufruf“ abgebildet. Das zeigt an, dass der *Secure Service Agent* aus der fachfunktionalen Komponente heraus aufgerufen wird. Dieser Aufruf ist generisch und kann daher werkzeugbasiert und damit ohne Komplexitätssteigerung durch den Fachentwickler in den Quellcode aufgenommen werden. Wie ein solcher Aufruf aussehen kann, zeigt der folgende Pseudocode:

```
if (
        PEP-Aufruf ( Sicherheitstoken, Objekt-
        Identifikator, Aufrufparameter[] ) == FALSE
    )
then
    {
        // Beende alle Operationen, verlasse Methode
        // und melde Fehler;
    }
```

Der PEP benötigt von der fachfunktionalen Komponente ein Sicherheitstoken, mit dessen Hilfe der PDP später den Aufrufer identifizieren kann, und einen Dienst-Identifikator (Objekt-Identifikator), der die angesprochene fachfunktiona-

le Komponente symbolisiert. Außerdem werden alle Parameter des Methodenaufrufs (`Aufrufparameter[]`) an den PEP übergeben, um eine feingranulare Policy-Auswertung im PDP durchführen zu können. All diese Informationen werden also wie im Pseudocode oben angedeutet durch die fachfunktionale Komponente an den PEP weitergegeben. Dieser führt zunächst Plausibilitätsprüfungen durch und stellt dann die eigentliche Autorisierungsanfrage an den PDP.

Die Realisierung des *Secure Service Agent* als eigenständige Komponente im Gegensatz zur Einhängung in die Nachrichten-*Queues* des *Application Server* bringt entscheidende Vorteile hinsichtlich der Portabilität (Anforderung A2.2) mit sich. Die Entwicklung von *Queue Handler* wie beim *Intercepting Web Agent* ist aufwendig und muss für jeden *Application Server* individuell durchgeführt werden. Selbst bei einem Versionssprung innerhalb desselben Produkts sind Weiterentwicklungen bis hin zur Neuerstellung notwendig. Durch die Realisierung als „reguläre“ Komponente wird der *Application Server* zum reinen *Container*-Element. Dies ermöglicht in der gleichen Art und Weise wie bei fachfunktionalen Webservices, diese auf unterschiedlichen *Application Server* derselben Plattform (JEE, .NET, …) zu betreiben, was die Portabilität erhöht.

4.3 Resümee

Die Migration von Fachfunktionalität bestehender Anwendungssysteme in Webservice-orientierte Architekturen (WSOA) erfordert gleichzeitig die Anwendungssystem-übergreifende Behandlung von Zugriffskontrolle. Die WSOA integriert die Fachfunktionalität und bietet sie damit Anwendungssystemübergreifend an. Genau dasselbe Prinzip sollte bei der Entwicklung einer dienstbasiert nutzbaren Zugriffskontrolle angewendet werden. In Kapitel 4.1 wurde dazu eine Zugriffskontroll-Architektur entworfen. Sie bietet der Facharchitektur zwei Dienste an: einen Authentifizierungsdienst, der ein WSOA-weites *Single Sign-On* ermöglicht, sowie einen Autorisierungsprüfungsdienst, der von jedem Webservice in Anspruch genommen werden kann. Neben dem Entwurf der Dienstschnittstellen wurden die Dienst-implementierenden Komponenten spezifiziert. Hinsichtlich der genutzten Datenquellen findet eine Verknüpfung mit den bestehenden Datenbanken und Verzeichnissen der WSOA statt, insbesondere mit dem Dienstverzeichnis, deren Einträge mit Zugriffskontroll-Policies verknüpft werden. Die Zugriffskontroll-Architektur ist fachübergreifend einsetzbar und nur in technischer Hinsicht auf die Einsetzbarkeit in Webservice-Architekturen beschränkt (Anforderung A1.1). Dabei findet eine explizite

Auftrennung zwischen dem Punkt-zu-Punkt- und dem Ende-zu-Ende-Sicherheitskontext statt. Gerade im Kontext Webservice-basierter, dienstorientierter Architekturen findet die Kommunikation zwischen einem Benutzer und einer in Anspruch genommenen Ressource über viele Zwischenstationen (engl. *Intermediaries*) statt. Beispiele hierfür sind Portale, Dienstkompositionen oder weiterleitende Komponenten wie der *Secure Service Agent*. Hinsichtlich eines Dienstaufrufes ist daher zunächst zu unterscheiden, ob der technische Aufruf (Punkt-zu-Punkt) zulässig ist. Im Falle der Kommunikation zwischen PEP und PDP wird dies implizit durch den Einsatz von WS-Security sichergestellt. Über dieser technischen Kommunikation findet jedoch der eigentliche Dienstaufruf statt, bei dem ein Benutzer (Subjekt) auf eine Ressource (Objekt) zugreifen möchte. Dieser Ende-zu-Ende-Kontext steht im eigentlichen Fokus der Betrachtung. Dies wird durch das zusätzlich transportierte Sicherheitstoken des Benutzers realisiert. Auch die Einbeziehung des Fachentwicklers gestaltet sich mit dem hier vorgestellten Vorgehen einfach (A1.2). Er kann nach wie vor ohne Veränderungen seinen individuellen Software-Entwicklungsprozess durchführen. Zur Nutzung des Zugriffskontroll-Diensts muss er lediglich den in Kapitel 4.2.2 vorgestellten PEP-Aufruf einbinden. Dieser ist bewusst so generisch aufgebaut, dass die Einbindung durch Code-Generatoren problemlos durchführbar ist. Die beiden Schnittstellen zur Authentifizierung und Autorisierungsprüfung wurden bewusst als Webservices ausgeprägt. Dies erleichtert die Einbindung in unterschiedlichen Programmiersprachen und ermöglicht, auch andere als den vorgestellten *Policy Enforcement Point* einzusetzen, was eine größtmögliche Interoperabilität herbeiführt (Anforderung A2.1). Die Verschattung der Komplexität der Zugriffskontroll-Architektur hinter den Dienstschnittstellen (vgl. Abbildung 28) ermöglicht es, sowohl kommerzielle Produkte als Implementierung zu nutzen, als auch Eigenentwicklungen einzusetzen. Dies schafft eine Integrationsfähigkeit (A2.3) und erlaubt es, sich eine individuelle Architektur im Sinne eines Baukasten-Prinzips zusammenzustellen. In Kapitel 4.2 wird mit dem *Secure Service Agent* ein Entwurfsmuster für die Verbindung von Facharchitektur mit Zugriffskontroll-Diensten entwickelt. Es zeichnet sich durch seine hohe Portabilität (A2.2) aus. Damit ist die Flexibilität hinsichtlich der Nutzung unterschiedlicher *Application Server* gegeben. Ein zentrales Element zur Durchführung von Zugriffsentscheidungen stellen jedoch die Zugriffskontroll-Policies dar. Nach der Einführung eines Zugriffskontroll-Modells für Webservice-orientierte Architekturen im nächsten Kapitel wird daraus eine Policy-Sprache abgeleitet, die durch die Zugriffskontroll-Architektur genutzt werden kann.

5 Zugriffskontroll-Modell und Policy-Sprache

Neben einer Zugriffskontroll-Architektur als Komponenten-basierte Erweiterung der mit Webservice-Technologie ausgeprägten Facharchitektur stellen Zugriffskontroll-Policies einen zentralen Baustein bei der Durchführung von Zugriffskontrolle dar. Im Gegensatz zur Architektur selbst, die in weiten Teilen statisch ist und als kompilierter und betriebener Programmcode betrachtet werden kann, handelt es sich bei Policies um Konfigurationsdaten, die schnelle und flexible Änderungen der Zugriffsberechtigungen ermöglichen sollen. Bevor effektiv auswertbare Policies spezifiziert werden können, ist zunächst das einzusetzende Zugriffskontroll-Modell festzulegen. Es soll gemäß der Anforderung A3.1 plattformunabhängig hinsichtlich der eingesetzten Sicherheitsprodukte gestaltet und lediglich auf den Einsatz von Webservice-Architekturen eingeschränkt sein. Darauf aufbauend ist eine Policy-Sprache zu definieren, die die Mengen und Relationen aus dem Zugriffskontroll-Modell aufgreift und in eine konkrete, aber plattformunabhängige Sprache abbildet. Dies ist eine Vorbedingung für die in Anforderung A3.2 postulierte frühzeitige Einbindung in den Software-Entwicklungsprozess.

5.1 Zugriffskontroll-Modell

Zugriffskontroll-Modelle stellen, wie in Kapitel 2.2 eingeführt, eine Verfeinerung der klassischen Subjekt-/Objekt-Relation dar. Der Grundgedanke ist, den Zugriff eines aktiv handelnden Subjekts auf ein geschütztes Objekt formal abzubilden.

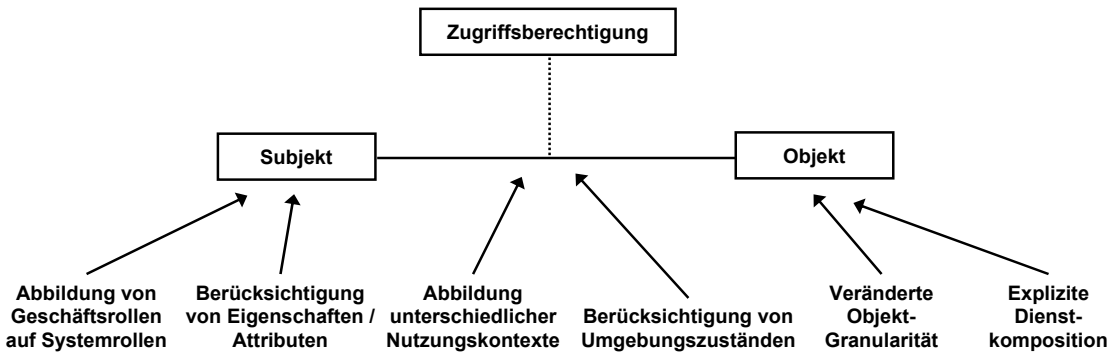


Abbildung 36: Die Subjekt/Objekt-Relation

Abbildung 36 zeigt den grundsätzlichen Aufbau einer Subjekt-/Objekt-Relation mit der Zugriffsberechtigung, die eine explizite Verbindung dieser beiden vorgenannten Mengen abbildet. In der Abbildung sind auch Fragestellungen an den entsprechenden Elementen skizziert, die durch die Einführung von dienstorientierten Architekturen aufgeworfen werden. Bei der Entwicklung eines Zugriffskontroll-Modells für den Einsatz in Webservice-basierten, dienstorientierten Architekturen stellen sich verschiedene Anforderungen, die sich auf die drei Basiselemente „Subjekt“, „Objekt“ und „Zugriffsberechtigung“ beziehen. Wie hängen die in der Organisations- bzw. Geschäftsmodellierung definierten Geschäftsrollen mit den durch Software-Systeme definierten Systemrollen zusammen? Bisherige Anwendungssysteme bringen ihre eigenen Rollen mit, die sich üblicherweise nicht unmittelbar auf die aus der Geschäftsmodellierung definierten Geschäftsrollen abbilden lassen. Ein Subjekt wird als digitale Identität repräsentiert, also einer Menge von Schlüssel/Wert-Paaren, auch Attribute genannt (vgl. Kapitel 2.2.3). Bestehende Anwendungssysteme nutzen diese nicht oder nicht vollständig bei der Berechnung einer Zugriffskontroll-Entscheidung. Gerade im Kontext feingranularer Zugriffskontrolle, wie sie beim Einsatz von Webservices benötigt wird, ist diese Einschränkung zu überdenken. Ebenfalls zu berücksichtigen ist die Veränderung der Granularität der zu schützenden Objekte. Während es sich bisher bei den atomaren Elementen um feingranulare Dateioperationen oder Ähnliches handelte, ist die kleinste, mit Zugriffskontrolle zu versehende Einheit im Webservice-Kontext die an der Webservice-Schnittstelle bereitgestellte Fachfunktionalität. Diese stellt ein geschäftsorientiertes und dadurch per se deutlich grobgranulareres Objekt dar. Ebenfalls zu thematisieren ist, wie die durch dienstorientierte Architekturen explizit gemachten Dienstkompositionen zu behandeln sind. In welcher Weise hängen ihre Zugriffsberechtigungen von denen der genutzten Basisdienste ab? Jenseits der Subjekte und der Objekte muss auch die Zugriffsberechtigung als Relation betrachtet werden. Dabei ist zu klären, wie die durch dienstorientierte Architekturen geforderte Wiederverwendung von Diensten (Objekten) in unterschiedlichen Nutzungskontexten sich bei der Spezifikation von Zugriffsberechtigungen abbilden lässt. Neben Daten, die unmittelbar Subjekten und Objekten zuordenbar sind, gibt es auch solche, die sich aus dem Umgebungszustand ergeben. Als Beispiele sind zu nennen die Auslastung eines *Servers* oder das Datum beziehungsweise die Uhrzeit des Zugriffs. Nachfolgend werden die einzelnen Bereiche „Subjekt“, „Objekt“ und die Relation „Zugriffsberechtigung“ als Grundlage des Zugriffskontroll-Modells feinspezifiziert.

5.1.1 Der Subjekt-Bereich

Während Abbildung 36 im Grundsatz das Modell für identitätsbasierte Zugriffskontrolle (engl. *Identity-Based Access Control*, IBAC) darstellt, wurde mit der heute üblicherweise eingesetzten, rollenbasierten Zugriffskontrolle (engl. *Role-Based Access Control*, RBAC) eine Indirektionsstufe zwischen Subjekten und Objekten geschaffen (vgl. Kapitel 2.2.2). Ursprünglicher Gedanke war die Gruppierung von menschlichen Benutzern (Unternehmensmitarbeitern) entsprechend ihrer Aufgabe (Geschäftsrolle) im Unternehmen. Dieses Vorgehen ermöglicht ein Skalieren bei der Zugriffsberechtigungsvergabe auch in großen Organisationseinheiten. Eine Nutzung der organisationsbezogenen Daten, wie sie durch die Aufbauorganisation einer Unternehmung definiert sind, um Zugriffskontrolle zu ermöglichen, ist grundsätzlich zu begrüßen. Es ermöglicht eine präzisere Grundlage für die Rechtevergabe. Analog dem Grundgedanken von dienstorientierten Architekturen, dass die Rechnerunterstützung sich nach geschäftlichen Bedürfnissen (und damit letztendlich den Geschäftsprozessen) richten muss, müssen sich die Zugriffsberechtigungen ebenfalls aus den geschäftlichen Prozessen und ihrer Randbedingungen ableiten lassen. Die derzeit notwendige Abbildung von geschäftlich definierten Rollen (Geschäftsrollen) auf durch die IT-Systeme bereitgestellten Rollen (Systemrollen) ist aufwendig. Es entsteht üblicherweise die Problematik, dass eine Rollenabbildung nicht ohne weiteres möglich ist, was zur Bildung von kartesischen Produkten der Systemrollen der unterschiedlichen Anwendungssysteme führt. Bei stetig anwachsender Zahl von klassischen IT-Systemen und einer dynamischen Anpassung beziehungsweise Feinspezifizierung von Geschäftsprozessmodellen führt das zu einer Explosion von Rollen, die kaum noch effizient administriert werden können [Sa01].

In Abbildung 37 wird ein Ansatz vorgestellt, wie die formale Beschreibung des Subjekts im Rahmen der Subjekt/Objekt-Relation verfeinert werden kann, insbesondere unter Berücksichtigung der bestehenden Konzepte der Geschäftsmodellierung und der rollenbasierten Zugriffskontrolle. Die Mengen und Relationen werden dazu in der Notation eines UML-Klassendiagramms dargestellt. Ausgangspunkt ist das auf der linken Seite notierte Subjekt. In Webservice-Umgebungen kann es sich dabei einerseits um einen menschlichen Benutzer handeln; es gibt jedoch auch autonom handelnde Systeme. Selbstverständlicherweise greift ein menschlicher Benutzer niemals selbst unmittelbar auf einen geschützten Webservice zu, sondern nutzt dafür immer technische Systeme wie beispielsweise einen Rechner mit installiertem Webbrowser. Diese Betrach-

tungsweise ist jedoch nur für die Behandlung des Punkt-zu-Punkt-Zugriffes relevant, nicht jedoch für die des Ende-zu-Ende-Zugriffs (vgl. Abbildung 19). Unter einem System als handelndes Subjekt wird eine autonom handelnde Einheit verstanden. Ein Beispiel hierfür wäre ein regelmäßig laufender Dienst, der Statusüberwachungen durchführt. Grundsätzlich ist festzuhalten, dass eine klare Unterscheidung zwischen einem menschlichen Benutzer und einem System anhand der vorgenannten Kriterien möglich ist, daher die Markierung der Vererbungsbeziehung mit *disjoint.*

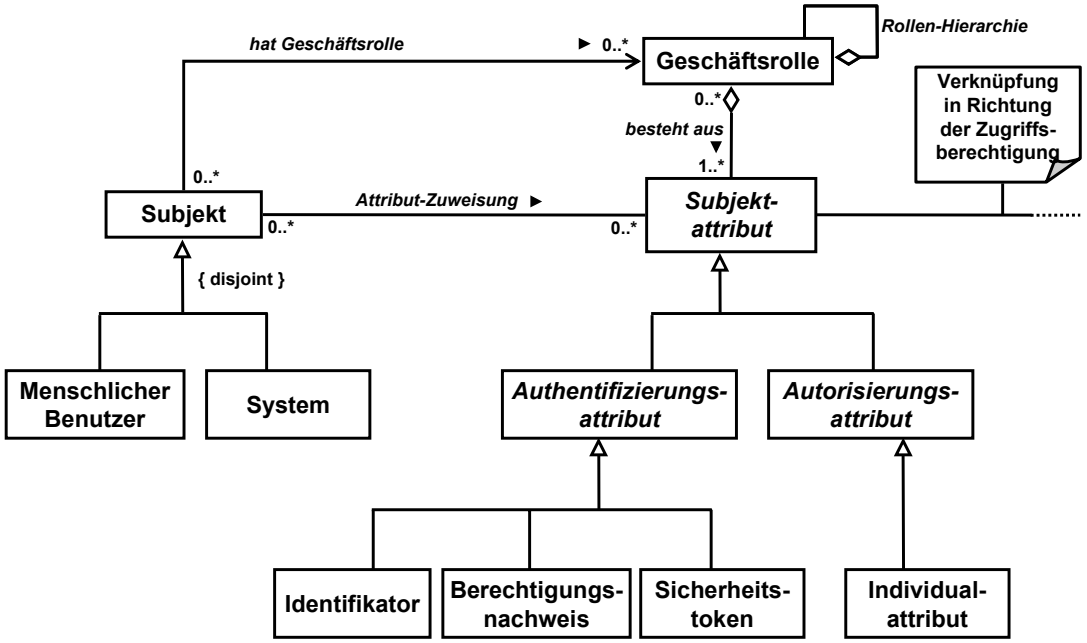


Abbildung 37: Der Subjekt-Bereich

Um den Bezug zur Geschäftsmodellierung herzustellen, wird das Konzept der Geschäftsrolle genutzt. Jedem Subjekt, insbesondere den menschlichen Benutzern, können Geschäftsrollen zugewiesen werden. Diese lassen sich unmittelbar aus der Geschäftsprozessmodellierung und damit einhergehend der Organisationsstruktur eines Unternehmens ableiten. Sie haben zunächst keinerlei Bezug zu den durch die Anwendungssysteme definierten Systemrollen. Analog der in [SC+96] vorgeschlagenen Möglichkeit zur Bildung von Rollen-Hierarchien wurde durch die selbstreflektierte Aggregationsbeziehung am Element „Geschäftsrolle“ diese Möglichkeit geschaffen. Im Gegensatz zu klassischen rollenbasierten Zugriffskontroll-Modellen wird die (Geschäfts-)Rolle jedoch nicht verwendet, um unmittelbar auf entsprechende Rollen der IT-Systeme abgebildet zu werden. Stattdessen wird analog der Konzepte der attributbasierten Zugriffskontrolle eine Abbildung zwischen Geschäftsrolle und subjektbasierten Attribu-

ten geschaffen. Das hier vorgestellte Konzept sieht vor, dass sich eine Geschäftsrolle in eine Menge von Subjektattribute abbilden lässt. Diese Subjektattribute sind unmittelbar mit den Subjekten verbunden und bilden in ihrer Gesamtheit die digitale Identität eines Subjekts. Dies stellt eine Erweiterung klassischer rollenbasierter Modelle dar. Für den einfachsten Fall, nämlich der ausschließlichen Nutzung von Geschäftsrollen zur Durchführung einer Zugriffskontroll-Entscheidung ohne den Einsatz weiterer Attribute, lassen sich die Geschäftsrollen auf ein einzelnes Subjektattribut abbilden. Beispielsweise kann das Subjektattribut mit dem Schlüssel (Attributnamen) „Rolle“ als Wert den Namen der Geschäftsrolle annehmen. Es ist aber ebenso möglich und sinnvoll, eine Geschäftsrolle auf mehrere Subjektattribute abzubilden. Nicht notwendigerweise müssen jedoch Subjektattribute unmittelbar aus Geschäftsrollen abgeleitet werden. Dies ermöglicht die Einbeziehung aller Subjekt-bezogenen Daten entsprechend des Schalenmodells der digitalen Identität (vgl. Kapitel 2.2.3) und gibt damit größtmögliche Flexibilität bei der Definition von Zugriffsberechtigungen, da dadurch prinzipiell sämtliche Daten einer digitalen Identität berücksichtigt werden können. Die Subjektattribute lassen sich in zwei Bereiche aufteilen, die jedoch nicht notwendigerweise disjunkt sind: Einerseits gibt es Attribute, die für die Authentifizierung des Subjekts benötigt werden, andererseits gibt es Attribute, die für die eigentliche Autorisierungsprüfung benötigt werden und damit Autorisierungen widerspiegeln. Als klassische Authentifizierungsattribute sind der Identifikator einer digitalen Identität sowie der zugehörige Berechtigungsnachweis (engl. *Credentials*) zu nennen. Für den Fall einer *Single Sign-On*-Lösung ist auch das entsprechende Surrogat, beispielsweise das ausgestellte Sicherheitstoken als Authentifizierungsattribut zu nennen, auch wenn es üblicherweise wegen seiner Transienz nicht unmittelbar dem Benutzerverzeichnis hinzugefügt wird, sondern gesondert gespeichert wird. Als Verknüpfungspunkt des Subjekt-Bereichs in Richtung des zu schützenden Objekts sollen ausschließlich Subjektattribute verwendet werden.

5.1.2 Der Objekt-Bereich

Während die Verfeinerung des Subjekt-Bereiches Objekt-neutral umgesetzt wurde, müssen zur weiteren Präzisierung der Subjekt/Objekt-Relation die zu schützenden Objekte näher betrachtet werden. Grundsätzlich handelt es sich bei den mit Zugriffskontrolle zu versehenden Objekten um Dienste, die im Rahmen von Rechnerunterstützung angeboten werden. In der Forschungsgruppe Coopera-

tion & Management wurde ein Dienstmodell für dienstorientierte Architekturen erarbeitet, an dem der Autor mitgewirkt hat [EK+07].

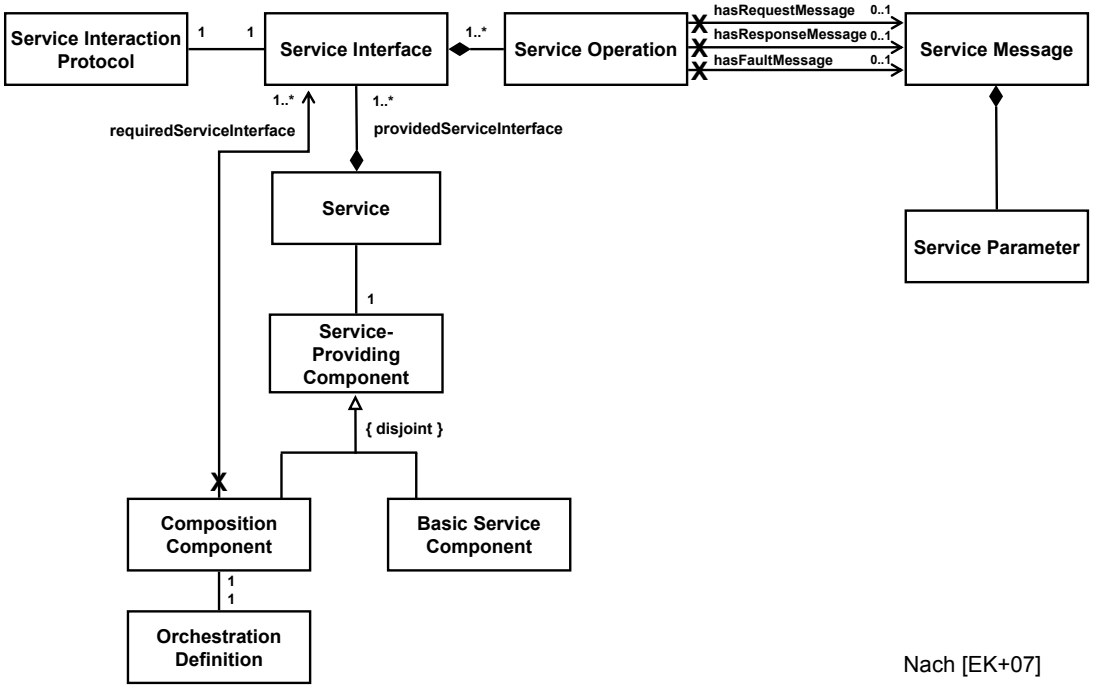
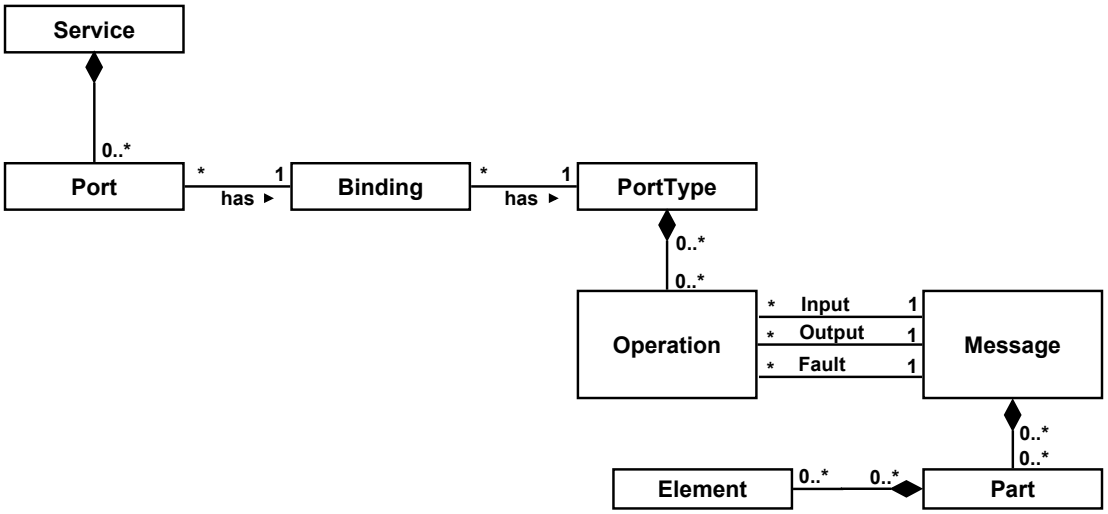


Abbildung 38: Der Kern des C&M-Dienstmodells

Abbildung 38 zeigt den konzeptionellen Kern des C&M-Dienstmodells für dienstorientierte Architekturen. Im Zentrum des Modells steht der Dienst (engl. *Service*), der seine Fachfunktionalität an einer Menge an Dienstschnittstellen (engl. *Service Interface*) bereitstellt. Jede Dienstschnittstelle besitzt ein definiertes Interaktionsprotokoll (engl. *Service Interaction Protocol*) und eine Menge an Dienstoperationen (engl. *Service Operation*). Eine Dienstoperation ist durch das Interaktionsprotokoll mit einem definierten Austausch von spezifizierten Nachrichten (engl. *Service Message*) assoziiert. Da eine Dienstoperation zwar nur jeweils maximal eine Anfrage- und Antwortnachricht besitzen, aber durchaus mehrere Parameter benötigen beziehungsweise liefern kann, lässt sich eine Nachricht aus einer beliebigen Anzahl von Parametern zusammensetzen. Um deutlich zu machen, dass diese Parameter im Kontext einer Nachricht auftreten, werden sie im Modell als *Service Parameter* bezeichnet. Ein Dienst wird durch eine Dienst-implementierende Komponente (engl. *Service-Providing Component*) erbracht. Bei dieser kann es sich entweder um einen Basisdienst (engl. *Atomic Service Component*) oder um eine Dienstkomposition (engl. *Composition Component*) handeln. Dienstkompositionen machen ihre Orchestrierung (engl. *Orchestration Definition*) explizit.

Während das C&M-Dienstmodell den weitestgehend technologieunabhängigen Aufbau von Diensten und Dienstkompositionen darstellt, muss für den Kontext dieser Arbeit eine Ausprägung des Dienstmodells hinsichtlich Webservice-orientierter Architekturen durchgeführt werden. Das bedeutet, dass die Mengen und Relationen aus dem C&M-Dienstmodell auf das Dienstverständnis der WSOA abgebildet werden müssen, das aus technischer Sicht in dem durch die WSDL-Schnittstellenspezifikation definierten Dienstverständnis geprägt wird.



Nach [Kl06]

Abbildung 39: Die Struktur der WSDL-Beschreibung

Abbildung 39 zeigt die Mengen und Relationen, aus denen die auf WSDL basierende Schnittstellenbeschreibung eines Webservice aufgebaut ist. Der logische Aufbau einer WSDL-basierten Dienstbeschreibung beginnt beim Beschreibungselement *Service* (dt. Dienst), der die umschließende Klammer bildet und eine Menge von *Ports* (dt. Kommunikationsendpunkt) darstellt. Die Endpunkte werden über ein *Binding* (dt. Bindung) an einen sogenannten *Port-Type* gebunden, der wiederum eine Menge von *Operations* (dt. Methoden) umfasst. Mit einer *Operation* können drei verschiedene Typen von Nachrichten (engl. *Message*) verbunden werden: eingehende (engl. *Input*) und ausgehende (engl. *Output*) Nachrichten sowie Fehlernachrichten (engl. *Fault*). Nachrichten bestehen aus einer Menge an *Parts* (dt. Teilstück), die wiederum aus *Elements* bestehen. Diese durch die WSDL definierten Strukturen bilden das stabile Grundgerüst zur Spezifikation einer Webservice-Schnittstelle.

Beim Vergleich mit der WSDL-Beschreibung (in der WSDL-Version 1.1) wird deutlich, dass alle relevanten Beschreibungselemente durch Modellierungskonstrukte im C&M-Dienstmodell abgebildet sind. Einzig der *Service Parameter* umfasst mehrere WSDL-Beschreibungselemente. Tabelle 4 stellt die Abbildung des C&M-Dienstmodells auf die WSDL dar. Die durch das C&M-Dienstmodell eingeführten Elemente *Service Interaction Protocol* werden in WSDL grundsätzlich nicht spezifiziert, ebenso werden die Informationen über die Dienst-erbringende Komponente nicht in der WSDL-konformen Schnittstellenbeschreibung offengelegt.

Modellierungselement des C&M-Dienstmodells	**WSDL-Beschreibungselement**
Service	Service
Service Interface	PortType
Service Operation	Operation
Service Message hasRequestMessage hasResponseMessage hasFaultMessage	Message Input Output Fault
Service Parameter	Part
	Element
	Types

Tabelle 4: Abbildung des C&M-Dienstmodells auf WSDL

Nachdem eine Übersicht über die zur Verfügung stehenden Daten gegeben wurde, ist jetzt die Frage zu klären, auf welcher Grundlage eine Zugriffskontroll-Entscheidung getroffen werden soll. Der Webservice, repräsentiert durch seine WSDL-konforme Dienstschnittstellenbeschreibung, stellt an sich nur eine lose Sammlung von unterschiedlichen *Operations* dar, die fachlich sehr unterschiedlich ausgeprägt sein können. Ebenso wie der *Port* und der *PortType* ist daher das *Service*-Element ein ungeeigneter Einstiegspunkt zur Fixierung eines zu schützenden Objekts. Das kleinste zu schützende Objekt stellt im Kontext Web-

service-orientierter Facharchitekturen daher die Webservice-Operation dar. Das bedeutet, dass auf dieser Ebene die Zugriffskontroll-Policies fixiert werden müssen. Eine Webservice-Operation wird ihrerseits charakterisiert durch den Nachrichtenaustausch in Form von eingehenden und ausgehenden Nachrichten. Sie ist im weitesten Sinne damit vergleichbar mit einem entfernten Methodenaufruf, wie beispielsweise dem *Remote Procedure Call* (RPC). Während bisherige Zugriffskontroll-Modelle eine Ressource sowie eine entsprechende Zugriffsaktion auf der Ressource als Tupel mit Zugriffskontrolle definieren [SC+96], ist dieser traditionelle Ansatz im Webservice-Kontext nicht sinnvoll. Dies hängt unmittelbar damit zusammen, dass einerseits im Bereich der Webservices grundsätzlich ein Aufruf (engl. *Execute*) anzunehmen ist und andererseits, dass die Granularität der zu schützenden Objekte gröber ist. Um die Semantik des Zugriffsversuchs hinreichend erfassen zu können, müssen daher Dienstaufrufparameter (Eingabeparameter) ebenfalls bei der Zugriffskontroll-Entscheidung berücksichtigt werden können.

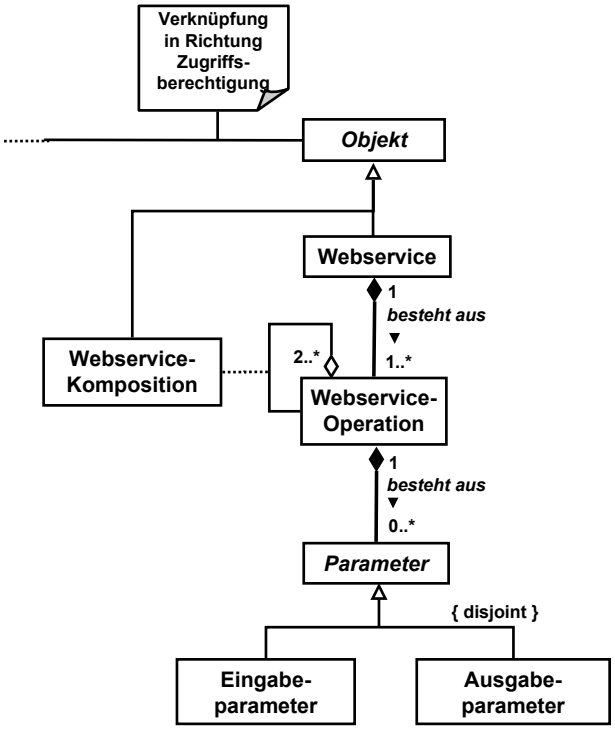


Abbildung 40: Der Objekt-Bereich

Eine Verfeinerung des Objekt-Bereichs für den Einsatz in Webservice-orientierten Architekturen wird in Abbildung 40 gegeben. Das abstrakte „Objekt“ wird instantiiert durch (elementare) Webservices. Diese können wiederum zu heterogen sein, um bereits auf dieser Ebene Zugriffskontroll-Policies zu definie-

ren. Daher wird der Webservice wiederum verfeinert durch die von ihm bereitgestellten Webservice-Operationen. Diese bilden im Zugriffskontroll-Modell dann den Verknüpfungspunkt zur eigentlichen Subjekt/Objekt-Relation. Webservice-Kompositionen stellen eine Aggregation einzelner Webservice-Operationen dar, dies wird durch eine Aggregationsfunktion dargestellt. Dies soll bei der Policy-Spezifikation berücksichtigt werden können. Das Konzept „Webservice-Komposition“ stellt daher ebenfalls ein Bindeglied zur später aufgezeigten Subjekt/Objekt-Relation dar.

Jede Webservice-Operation kann sowohl eingehende als auch ausgehende Nachrichten spezifizieren. Diese bestehen im Grundsatz aus einer Menge von spezifizierten Parametern, die ein wesentliches Element bei der Zugriffskontroll-Entscheidung darstellen. Daher wird zur Verfeinerung die abstrakte Klasse „Parameter“ genutzt, die sich disjunkt in die Klassen „Eingabeparameter“ und „Ausgabeparameter“ auftrennen lässt. Für die Zugriffskontrolle ist die Berücksichtigung der Eingabeparameter hinreichend unter der Prämisse, dass die Funktionsweise der die Webservice-Operation implementierenden Komponente bekannt ist. Dies wird dadurch sichergestellt, dass der Webservice-Entwickler parallel zur Entwicklung der Fachfunktionalität auch die entsprechenden Zugriffskontroll-Informationen als Vorstufe effektiv auswertbarer Zugriffskontroll-Policies spezifiziert. Die Berücksichtigung der Eingabeparameter erlaubt es, weniger und dafür gröber geschnittene Basisdienste anzubieten. Ein Beispiel hierfür ist die Bereitstellung personenbezogener Daten wie die eines Notenauszuges im universitären Kontext. Im Grundsatz genügt eine Webservice-Operation, die zu einer als Eingabeparameter übergebenen Matrikelnummer das zugehörige Resultat übermittelt. Wie im weiteren Verlauf des Kapitels gezeigt wird, ist es durch die Berücksichtigung der Eingabeparameter bei der Zugriffskontroll-Entscheidung möglich, Zugriffsberechtigungen so zu formulieren, dass jeder Anfragende nur Zugriff auf seine eigenen, personenbezogenen Daten bekommt. Der Verwaltung, deren Mitarbeiter durch entsprechende Subjektattribute ausgezeichnet werden können, können weiter reichende Berechtigungen auf fremden Datensätzen eingeräumt werden. In der traditionellen Software-Entwicklung wären zur Abbildung der unterschiedlichen Nutzungskontexte üblicherweise getrennte Fachfunktionalitäten bereitgestellt worden. Dies ist jedoch nicht im Sinne der dienstorientierten Architektur, da es sich aus fachlicher Sicht um einen identischen Vorgang handelt, der sich lediglich hinsichtlich der Zugriffsberechtigungen unterscheidet. Die Abbildung auf eine einzige Dienstschnittstelle und damit die saubere Trennung zwischen Fachfunktionalität und

Zugriffsberechtigungen ermöglicht unter anderem Erleichterungen hinsichtlich der Wartbarkeit auf Fachseite.

5.1.3 Die Zugriffsberechtigung als Relation

Nach der Feinspezifizierung der Subjekte und der Objekte ist der nächste Schritt bei der Entwicklung eines Zugriffskontroll-Modells, die Zugriffsberechtigung als Relation beider Mengen zu definieren. Wie in Kapitel 2.2.2 eingeführt wurde, werden Zugriffsberechtigungen üblicherweise in zwei Formen gespeichert: entweder in Form von Zugriffskontroll-Listen (engl. *Access Control List*, ACL), die das Objekt in den Vordergrund stellen und definieren, welche Berechtigungen ein Subjekt auf dem Objekt hat. Es können alternativ auch Befähigungens-Listen (engl. *Capability Lists*) eingesetzt werden, die das Subjekt in den Vordergrund rücken und die Rechte eines Subjekts auf die zu Verfügung stehenden Objekte definiert. Es gibt auch Ansätze, die eine unmittelbare Ankopplung der Zugriffsberechtigungen an Subjekte beziehungsweise an Objekte aufheben wollen (vgl. [YT05], Kap. 3.3.2). Zugriffskontroll-Policies sollen demnach unabhängig von konkreten Subjekten und Objekten formuliert werden und sich aus allgemeinen Unternehmensrichtlinien ableiten. Wie bereits in der zugehörigen Bewertung aufgezeigt, liegt das Problem dieses Ansatzes im aufwendigen Policy-*Matching*.

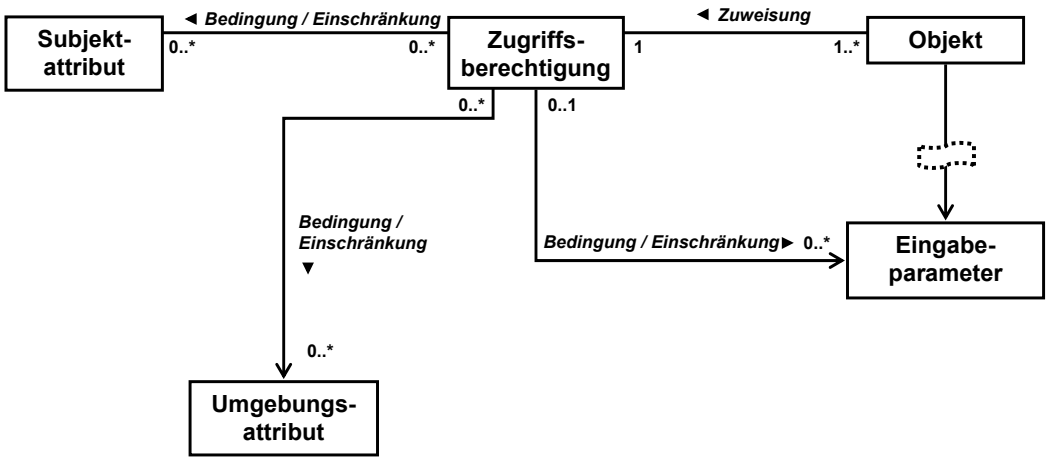


Abbildung 41: Grundlegendes Zugriffskontroll-Modell

Der wichtigste zu berücksichtigende Punkt ist jedoch die Integration in den Software-Entwicklungsprozess, wie in Anforderung A3.2 definiert. Erst dies ermöglicht die frühzeitige Einbeziehung des Fachentwicklers bei der Spezifikati-

on von Zugriffskontroll-Policies (Anforderung A1.2). Daher wird im vorgestellten Zugriffskontroll-Modell das zu schützende Objekt als Ausgangspunkt zur Spezifikation der zugehörigen Zugriffskontroll-Policy genommen. Vom Grundgedanken her wird dazu jedem Objekt eine Zugriffsberechtigung zugewiesen. Dies erleichtert, wie in Kapitel 6 gezeigt wird, die Einbindung von Geschäftsanalysten und Software-Entwicklern, da dadurch ein unmittelbarer Bezug zum Webservice als Fachobjekt gegeben wird. Um die Möglichkeit der Wiederverwendung von Zugriffsberechtigungen anzudeuten, wurde in Abbildung 41 die Assoziation zwischen der Zugriffsberechtigung und dem Objekt mit den Multiplizitäten „1“ beziehungsweise „1..*“ definiert. Dies macht deutlich, dass jedem Objekt genau eine Zugriffsberechtigung zugewiesen ist, es jedoch die Möglichkeit gibt, dass mehrere Objekte dieselbe Zugriffsberechtigung verwenden können. Durch die sich stark vergrößernde Menge an Objekten, also an fachlich motivierten Webservices, spielt dies gerade für den Betrieb einer dienstorientierten Architektur eine nicht zu unterschätzende Rolle. Neben dem Objekt, üblicherweise definiert durch seinen Objekt-Identifikator, muss die Möglichkeit geschaffen werden, dass die Zugriffsberechtigung auch die entsprechenden Eingabeparameter eines Objekts einbeziehen kann. Dies wird durch eine entsprechende Assoziation spezifiziert. Zur Definition von Bedingungen beziehungsweise Einschränkungen, die hinsichtlich des aufrufenden Subjekts zu treffen sind, wird eine Assoziation zwischen den Elementen „Subjektattribut“ und „Zugriffsberechtigung“ gesetzt. Um auch die Möglichkeit zu haben, Umgebungszustände, beispielsweise zeitlicher Art oder die Auslastung betreffend, bei der Zugriffskontroll-Entscheidung zu berücksichtigen, ist eine entsprechende Assoziation vorgesehen.

Abbildung 42 führt die vorangegangenen Teilmodelle zusammen und verfeinert das Element „Zugriffsberechtigung“ durch Auftrennung in die Elemente „Zugriffskontroll-Policy“ und „Zugriffsberechtigung“. Ausgangspunkt ist, dass die Wiederverwendung von Diensten in unterschiedlichen Nutzungskontexten (Geschäftsprozessen) erleichtert werden soll. Dabei besteht eine gewollte Unabhängigkeit der jeweiligen Nutzungskontexte voneinander. Dieser Sachverhalt wird durch das Element „Zugriffskontroll-Policy“ abgebildet. Jedes Objekt ist mit einer Zugriffskontroll-Policy verbunden. Diese Policy an sich ist als Auflistung einzelner Zugriffsberechtigungen zu verstehen, die jeweils die Bedingungen hinsichtlich eines konkreten Nutzungskontexts zusammenfassen. Diese Darstellung bringt unmittelbare Vorteile mit sich: Nutzungskontexte sind unabhängig voneinander darstellbar. Für den Fall, dass Nutzungskontexte für ein konkretes

Objekt hinzugefügt oder entfernt werden müssen, ergeben sich keine Änderungen hinsichtlich der sonstigen Zugriffsberechtigungen. Auch wenn sich innerhalb eines Nutzungskontextes Änderungen hinsichtlich notwendigerweise zu erfüllenden Bedingungen ergeben, wirken sich diese nicht auf die weiteren Nutzungskontexte (Zugriffsberechtigungen) des Objekts aus. Die Zugriffskontroll-Policy stellt also lediglich eine Verzeigerung (engl. *Pointer*) auf die anwendbaren Zugriffsberechtigungen dar. Dies bringt Vorteile auch bei der Verwaltung dieser Informationen mit, wie bei der Einbettung in den Software-Entwicklungsprozess (Kapitel 6) gezeigt wird.

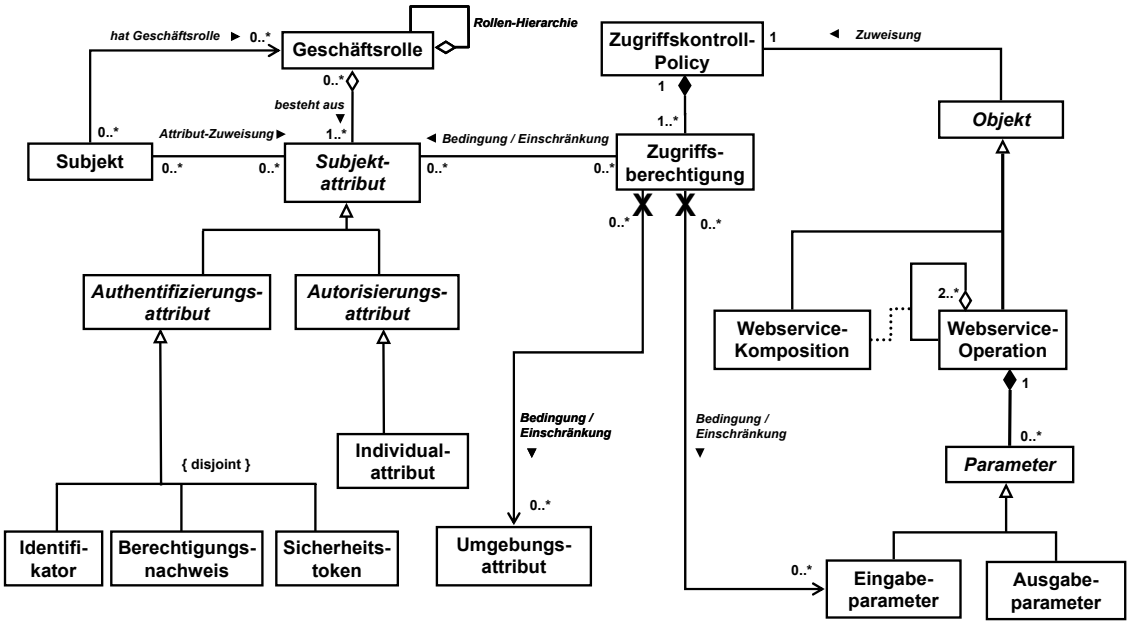


Abbildung 42: Zugriffskontroll-Modell für Webservice-Architekturen

Als Beispiel zur Verdeutlichung kann der in Kapitel 5.1.2 eingeführte Dienst zur Bereitstellung personenbezogener Daten genutzt werden. Er ist realisiert durch eine Webservice-Operation, die zu einer als Eingabeparameter übergebenem Subjekt-Identifikator das entsprechende Resultat übermittelt. Die Webservice-Operation wird als zu schützendes Objekt mit einem eindeutigen Objekt-Identifikator versehen und in der Policy-Datenbank wird die entsprechende Zugriffskontroll-Policy hinterlegt, die eine Auflistung der anwendbaren Zugriffsberechtigungen darstellt. Im konkreten Fall wären dies zwei, entsprechend der beschriebenen Nutzungskontexte. Der erste Nutzungskontext wäre die Selbstauskunft. Hierfür muss der Identifikator des anfragenden Subjekts mit dem zu den angefragten Daten gehörenden Identifikator übereinstimmen, um Zugriff zu erhalten. Für den zweiten, und aus Sicht der Zugriffskontrolle vollständig

unabhängigen Fall, bei dem die Verwaltung mit weitreichenden Rechten ausgestattet ist, genügt die Prüfung eines Subjektattributes, das die organisatorische Zugehörigkeit zur Personalabteilung repräsentiert. Eine Prüfung der Eingabeparameter ist in diesem Falle nicht notwendig, um Zugriff zu erhalten. Dies wird durch eine getrennte Zugriffsberechtigung dargestellt. Beide Zugriffsberechtigungen werden durch die zum Objekt zugehörige Zugriffskontroll-Policy verzeigert.

5.1.4 Berücksichtigung von Dienstkompositionen

In dienstorientierten Architekturen werden Dienstkompositionen explizit spezifiziert. Unter einer Dienstkomposition wird im Kontext Webservice-orientierter Architekturen ein BPEL-Prozess verstanden, der eine Vielzahl verschiedener Webservice-Operationen verschaltet. Ein BPEL-Prozess als Ablaufbeschreibung wird ebenfalls in Form eines Webservices bereitgestellt und kann als ein solcher verwendet werden. BPEL-Prozesse vereinfachen die systemübergreifende Anwendungsintegration, da standardisierte Dienstschnittstellen und Kommunikationsprotokolle zum Einsatz kommen. Da BPEL-Prozesse oft langlaufend sind, können sich Rücksetz-Operationen (engl. *Rollback-Operation*) beliebig aufwendig gestalten. Ziel ist daher, den Inhalt der explizit gemachten Dienstkomposition für eine Zugriffskontroll-Entscheidung zu berücksichtigen. Dienstkompositionen können unter anderem Verzweigungen enthalten, das bedeutet, dass der Kontrollfluss nicht unmittelbar voraussagbar ist. Daher ist zwischen zwei Arten von Dienstaufrufen einer Dienstkomposition zu unterscheiden: solche, die auf jeden Fall durchgeführt werden, und solche, die sich auf einem Verzweigungsast befinden. Das vorgestellte Zugriffskontroll-Modell ermöglicht es, die Policies der auf jeden Fall aufgerufenen Webservice-Operationen der Dienstkomposition zu aggregieren; sie können in einer Vorabprüfung beim Aufruf der Dienstkomposition verifiziert werden. Damit lassen sich Rücksetz-Operationen verhindern. Dabei muss jedem BPEL-Prozess ein initialer Aufruf des Autorisierungsprüfungsdiensts hinzugefügt werden, der genauso generisch ist wie der des *Secure Service Agents*. Auch die Erstellung der Policies für die Dienstkomposition lässt sich auf einfache Art bewerkstelligen: Es findet eine UND-Verknüpfung aller Zugriffskontroll-Policies der Webservices statt, die auf jeden Fall aufgerufen werden müssen.

5.2 Policy-Sprache

Nachdem im vorangegangenen Kapitel die Mengen und Relationen zur Spezifikation von Zugriffskontroll-Policies im Webservice-Kontext eingeführt wurden, wird in diesem Kapitel mit der *Web Services Access Control Markup Language* (WSACML) eine Domänen-spezifische Policy-Sprache entwickelt. Diese erweitert die durch das konzeptionelle Metamodell vorgegebene abstrakte Syntax zu einer konkreten Syntax.

5.2.1 Entwicklung Domänen-spezifischer Sprachen

Die formale Modellierung stellt den Ausgangspunkt der Erstellung von unmittelbar auswertbaren Zugriffskontroll-Policies dar und wird im späteren Verlauf (vgl. Kapitel 6) den Einstieg in die modellgetriebene Entwicklung dieser Policies bereiten. Hierzu werden die relevanten Begriffe aus diesem Kontext mit den in dieser Arbeit verwendeten Artefakten in Beziehung gestellt. Abbildung 43 stellt die Begriffsbildung für die Erstellung Domänen-spezifischer Sprachen dar [SV+05]. Bei der betrachteten Domäne kann es sich um eine technische und/oder fachliche Domäne handeln. In dieser Arbeit handelt es sich um die technische Domäne der Webservice-orientierte Architekturen, für die Zugriffskontroll-Policies spezifiziert werden sollen. Eine fachliche Einschränkung wird bewusst nicht vorgenommen (vgl. Anforderung A1.1).

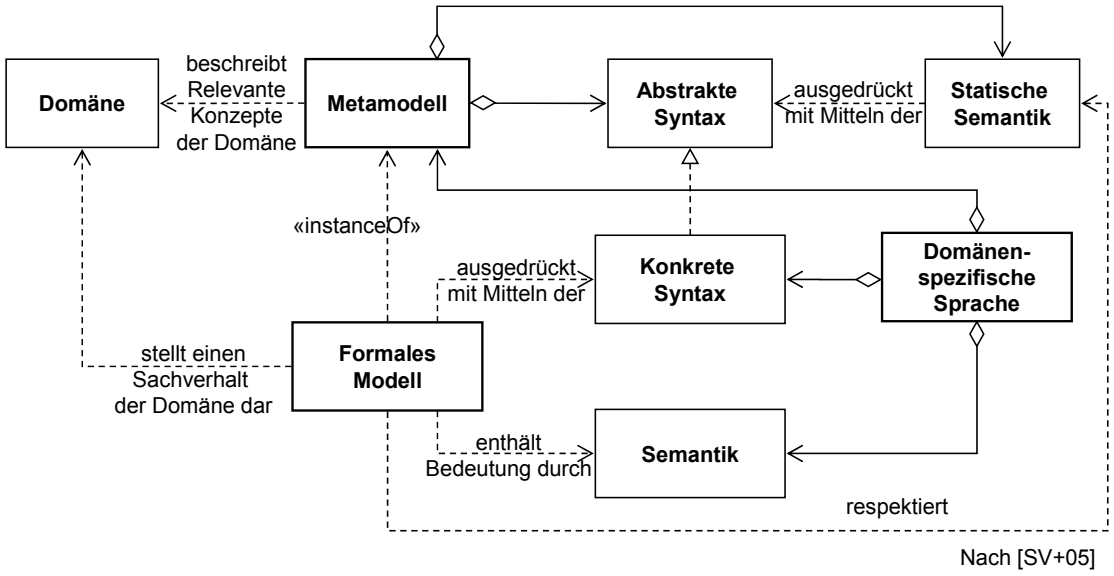


Abbildung 43: Modellierung Domänen-spezifischer Sprachen

Um die relevanten Inhalte der Domäne formal beschreiben zu können, muss man sie in soweit verstehen, dass man sie formalisieren kann. Hierzu wurden in Kapitel 5.1 die Mengen und Relationen erfasst, die zur Durchführung einer Zugriffskontroll-Entscheidung herangezogen werden können. Dies wird mit dem Begriff „abstrakte Syntax“ bezeichnet. Zusätzlich umfasst das Metamodell auch die statische Semantik einer Sprache, die die Bedingungen (engl. *Constraints*) festlegt, die ein wohlgeformtes Modell erfüllen muss. Eine domänenspezifische Sprache (engl. *Domain-Specific Language*, DSL) besteht im Kern aus einem Metamodell mit dazugehöriger abstrakter Syntax und statischer Semantik und definiert darauf eine konkrete Syntax. Mit Hilfe der konkreten Syntax, die sich aus der abstrakten Syntax ableiten lassen können muss, besteht die Möglichkeit, formale Modelle zu erstellen. Ein formales Modell im Kontext dieser Arbeit ist ein Artefakt, das eine Zugriffskontroll-Policy so darstellt, dass sie effektiv weiterverarbeitbar ist. Das bedeutet, dass eine solche Policy unmittelbar auswertbar ist oder im Kontext der modellgetriebenen Entwicklung automatisiert in ein solches Artefakt transformierbar ist.

5.2.2 Die Web Services Access Control Markup Language (WSACML)

Die konkrete Syntax der *Web Services Access Control Markup Language* (WSACML) wurde als XML-Schema entsprechend der XML-Schema-Definition (XSD) definiert [EK+08]. Dies ermöglicht einerseits syntaktische Verifikation der XML-Artefakte gegen die XSD und bildet für den Einsatz im Rahmen der modellgetriebenen Policy-Entwicklung einen idealen Ausgangspunkt, da die Werkzeugunterstützung für XML-basierte Spezifikationen sowie darauf aufbauenden Transformationen ausgereift ist. Zur besseren Darstellung des entwickelten XML-Schemas wurde in Abbildung 44 eine UML-basierte Darstellung der WSACML als Klassendiagramm gewählt. Diese ist unmittelbar mit dem XSD verknüpft: UML-Klassen werden zu XSD *Complex Types*, Aggregationen zeigen Eltern-Kind-Beziehungen und Attribute der UML-Klassen werden zu Attributen in der XSD. Für die Abbildung sind insbesondere die mit `<<root>>` markierten Klassen wichtig; sie erzeugen ein XSD-Wurzelelement. Die Begriffe der WSACML als konkrete Syntax des konzeptionellen Zugriffskontroll-Modells wurden in englischer Sprache fixiert.

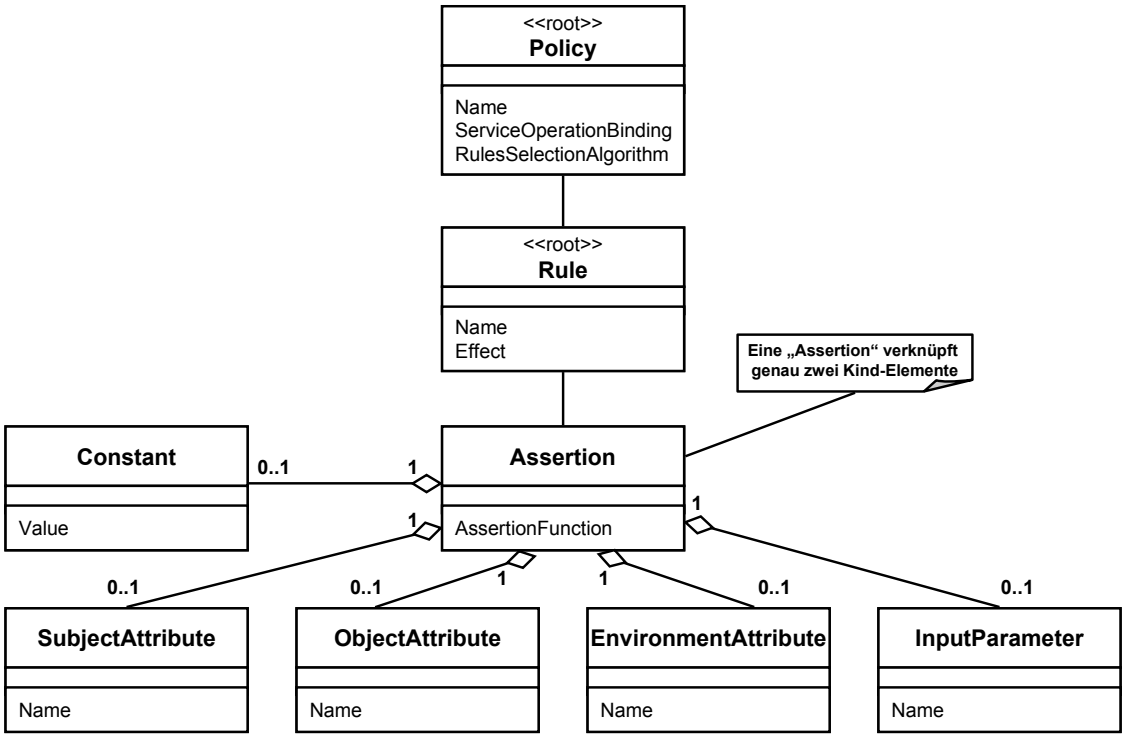


Abbildung 44: Web Services Access Control Markup Language

In Tabelle 5 wird dargestellt, wie die zur Spezifikation einer Zugriffskontroll-Policy notwendigen Elemente (vgl. Abbildung 41), mit den Sprachelementen der WSACML korrelieren.

Element des Zugriffskontroll-Modells	Element der WSACML
Zugriffskontroll-Policy	Policy
Zugriffsberechtigung	Rule / Assertion
Subjektattribut	SubjectAttribute
Objekt (Objekt-Identifikator)	ObjectAttribute
Umgebungsattribut	EnvironmentAttribute
Eingabeparameter	InputParameter
<keine Entsprechung>	Constant

Tabelle 5: Abbildung des Zugriffskontroll-Modells auf WSACML

Die WSACML wurde dazu bewusst in zwei Teile zerlegt: Einerseits gibt es die direkte Verknüpfung der zu schützenden Objekte, genauer gesagt der Webservice-Operationen, mit den Zugriffskontroll-Policies. Dies wird durch das Wurzelelement (engl. *Root*) `Policy` der WSACML abgebildet. Das `Policy`-Element nimmt durch sein Attribut `ServiceOperationBinding` unmittelbar und eineindeutig Bezug zu einer Webservice-Operation. Aus Sicht der `Policy` ist es unerheblich, ob es sich bei der Webservice-Operation um einen Basisdienst oder um eine Dienstkomposition handelt. Dieser Sachverhalt wird im Rahmen der `Policy`-Erstellung berücksichtigt. Das `Policy`-Element selbst ist eine geordnete Auflistung von anwendbaren `Rules` (dt. Regelwerken). Jede `Rule` entspricht dabei einem Nutzungskontext der Webservice-Operation, wobei der Nutzungskontext einer Dienstnutzung in einem konkreten Geschäftsprozess entspricht. Zur Bestimmung einer geeigneten `Rule` wird im `Policy`-Element die Möglichkeit geschaffen, einen Auswahlmechanismus zu definieren. Dies wird durch das Attribut `RuleSelectionAlgorithm` spezifiziert. Häufig eingesetzte Mechanismen sind, den ersten anwendbaren Effekt (engl. *Effect*) zu verwenden (`First-Applicable`), oder dass ein Verbot grundsätzlich stärker bindet als eine Erlaubnis (`Deny-Overrides`). Die `Policy` selbst ist durch ihr Attribut `Name` individuell adressierbar. Sie ist im Grundsatz ein eigenständiges Objekt; es besteht jedoch ein 1:1-Bezug zu den Webservice-Operationen.

Das zweite Wurzelelement der WSACML ist das Element `Rule`. Sie verkörpert die notwendigen Bedingungen, die für die Ausführung der Webservice-Operation in einem gegebenen Nutzungskontext erfüllt sein müssen. Jede `Rule` ist durch ihr Attribut `Name` eindeutig referenzierbar, was für die spätere Einbindung in die `Policies` notwendig ist und zusätzlich die Wiederverwendbarkeit der `Rules` erleichtert. Das Attribut `Effect` der `Rule` definiert, ob ihre Anwendbarkeit zu einer Gestattung (engl. `Permit`) oder zu einem Verbot (engl. `Deny`) führt. Zur Realisierung dienstorientierter Zugriffskontrolle genügen diese beiden Effekte, da die Fachkomponente lediglich daran interessiert ist, ob ein Zugriff durchgeführt werden darf (`Permit`) oder ob er abgewiesen werden soll (`Deny`). Sonderfälle, wie beispielsweise wenn keine `Rule` anwendbar ist, lassen sich auf ein Standardverhalten abbilden, üblicherweise den Effekt `Deny`. Eine WSACML-`Rule` beinhaltet nicht nur die notwendigen Bedingungen, die erfüllt sein müssen, um eine Zugriffskontroll-Entscheidung zu treffen, sondern definiert auch, ob eine Anwendbarkeit der `Rule` zulässig ist oder nicht. Dies wird durch das WSACML-Element `Assertion` gewährleistet. Mathematisch gesehen stellt die

`Assertion` ein Prädikat im Sinne der Prädikatenlogik dar. Sie definiert genau zwei Kindelemente als Leerstellen des Prädikats, verbunden durch eine Funktion der Enumeration `AssertionFunction`. Als Elemente der `Assertion-Function` sind Vergleichsoperatoren wie „ungleich“, „gleich“, „kleiner-gleich“ oder „größer-gleich“ vorgesehen. Durch den Vergleich zweier Werte entsprechend der Vorgabe der `Assertion` ist sichergestellt, dass bei der Auswertung ein boolescher Wert entsteht. Damit eine `Rule` anwendbar ist, müssen alle ihre `Assertions` auf WAHR evaluiert werden. Dies stellt sicher, dass sowohl der Nutzungskontext zutreffend ist (alle `Assertions` sind evaluierbar) als auch dass die notwendigen Bedingungen des Nutzungskontext erfüllt sind (alle `Assertions` evaluieren auf WAHR). Für diesen Fall ist der Nutzungskontext sowohl gegeben als auch positiv anwendbar, die `Rule` erzeugt also den durch sie definierten `Effect`. Im Gegensatz zu anderen Policy-Sprachen gibt die WSACML die Freiheit, beliebige Kindelemente der `Assertion` zu kombinieren. Es sind nicht nur Vergleiche gegen einen konstanten Wert (`Constant`) zulässig, sondern insbesondere auch Vergleiche von variablen Werten. So ist es möglich, Daten aus dem Objektbereich (`ObjectAttributes`) gegen Daten des Subjekts (`SubjectAttributes`) zu vergleichen oder Eingabeparameter (`InputParameters`) gegen Subjektattribute zu prüfen.

Die durch XML-Schema vorgegebene Sprachdefinition von WSACML legt die nutzbaren Attribute als Elemente der `Assertions` nicht fest. Dies gibt auf der einen Seite weitreichende Freiheitsgrade, aber für eine konkrete Einsetzbarkeit muss für die zu schützende Domäne ein Vokabular im Sinne der nutzbaren Attribute für die Bereiche Subjekt, Objekt und Umgebungszustand definiert werden. Den Eingabeparametern kommt eine gesonderte Rolle zu, da diese automatisiert aus den WSDL-basierten Schnittstellenbeschreibungen der Webservices extrahiert werden können. Die Pflege des Vokabulars ist insbesondere für den im nachfolgenden Kapitel beschriebenen Werkzeug-gestützten und modellgetriebenen Entwicklungseinsatz relevant.

In Abbildung 45 wird exemplarisch ein Vokabular in Form eines UML-Klassendiagramms dargestellt. Die möglichen Attribute werden den unterschiedlichen Bereichen zugewiesen und durch einen Zugriffskontroll-Kontext (engl. *Access Context*) miteinander verknüpft. Zur Erstellung des Vokabulars empfiehlt sich ein *Middle-Out*-Vorgehen. Einerseits sollten Informationen aus der Geschäftsprozessmodellierung berücksichtigt werden, andererseits empfiehlt es sich, auch

die Datenquellen der bestehenden Anwendungssysteme zu untersuchen. Das Vokabular kann in zwei Ebenen aufgetrennt werden: einerseits in eine technische Auflistung, die für syntaktische Validierung von WSACML-Policies verwendet werden kann. Anderseits sollte auch eine Ausfertigung in Prosa in Analogie eines Glossars entstehen, die einem Fachentwickler die Erstellung von Zugriffskontroll-Policies ermöglicht.

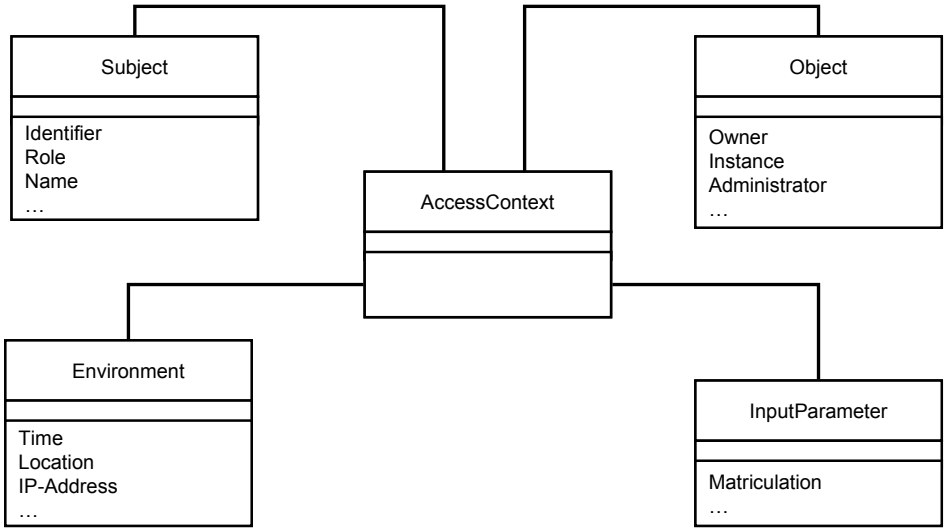


Abbildung 45: Definition des Vokabulars

Ziel der WSACML ist es, Komplexität hinsichtlich der Zugriffskontroll-Policies zu verteilen. Die Sprache an sich ist unabhängig von den später eingesetzten Sicherheitsprodukten, und ermöglicht daher die Abstraktion der Policies auf ein rein fachliches Niveau. Für die in Kapitel 6 vorgestellte Einbindung in den Software-Entwicklungsprozess ist eine solche Abstraktion sehr hilfreich, da sie ermöglicht, bereits frühzeitig die vorliegenden Informationen zur Zugriffskontrolle formal zu spezifizieren. Im Kontext der modellgetriebenen Software-Entwicklung ist sie auf Ebene der plattformunabhängigen Modelle (PIM) einzuordnen.

5.2.3 Beispielhafter Einsatz der WSACML

Zur Verdeutlichung wird das in Kapitel 5.1 eingeführte Beispielsszenario ausgeprägt: Ein Notenauszug wird als Webservice angeboten. Bereits bei der Entwicklung sind zwei Nutzungskontexte des Dienstes gegeben: Einerseits darf jeder Studierende seinen eigenen Notenauszug abrufen, jedoch keine fremden. Ande-

rerseits dürfen Studienberater im Rahmen der Studierendenberatung auf die Notenauszüge aller Studierenden zugreifen.

```
<Policy Name=„createToR_Policy"
   ServiceOperationBinding=„ToRService/createToR“
   RuleSelectionAlgorithm=„first-applicable “>
  <RuleReference>StudentSelfService</RuleReference>
  <RuleReference>StudentConsultation</RuleReference>
</Policy>
```

Abbildung 46: Beispiel einer WSACML-Policy

Der Notenauszugs-Webservice stellt eine Menge an Operationen zur Verfügung, die in unmittelbarem Zusammenhang mit der Erstellung von Notenauszügen stehen. Unter anderem auf die hier relevante Operation, die die eigentliche Erstellung vornimmt. In Abbildung 46 wird die Policy als Artefakt dargestellt. Sie beginnt mit dem XML-*Tag* `Policy`, das zu Referenzierungszwecken einen Namen zugewiesen bekommt. Im einfachsten Fall kann als Name der Policy der Name des Webservices und/oder der Webservice-Operation gewählt werden. In diesem Fall wurde `createToR_Policy` gewählt. Das `Policy`-Element besitzt eine Menge an Attributen, das wichtigste dabei ist die Verknüpfung mit der zu schützenden Webservice-Operation. Es bietet sich an, diese aus Gründen der Eindeutigkeit in der Kombination <Name des Webservice>/<Name der Webservice-Operation> darzustellen. Hier handelt es sich also um die WSACML-`Policy`, die mit der Operation `createToR` des Webservices `ToRService` assoziiert ist. Als Auswahlmechanismus wurde `first-applicable` definiert, also die erste zutreffende `Rule` bestimmt den Effekt der `Policy`. Anschließend werden die anwendbaren Nutzungskontexte durch das Element `RuleReference` spezifiziert. Es empfiehlt sich, auch hier mit sprechenden Namen zu arbeiten, auch wenn dies aus technischer Sicht nicht erforderlich ist. Die zwei vorgenannten Nutzungskontexte werden in den `Rules` `StudentSelfService` für die Selbstbedienungsfunktionalität und `StudentConsultation` für die Studierendenberatung fixiert.

Abbildung 47 zeigt die zwei Nutzungskontexte dargestellt als WSACML-`Rules`. Die erste `Rule` bildet die Selbstbedienungsfunktionalität ab. Sie besteht aus zwei `Assertions`. Zunächst muss ein `SubjectAttribute` mit dem Namen `role` existieren. Dies wird zur Abbildung von Geschäftsrollen auf

Subjekte (im Sinne ihrer digitale Identitäten) durchgeführt. Es muss mit dem konstanten Wert `student` übereinstimmen. Damit wird geprüft, ob es sich um den Zugriff eines Studierenden handelt. Die zweite `Assertion` prüft, ob das `SubjectAttribute` des aufrufenden Subjekts des Ende-zu-Ende-Zugriffs mit dem Namen `identifier` mit dem übergebenen `InputParameter` mit dem Namen `matriculation` übereinstimmt.

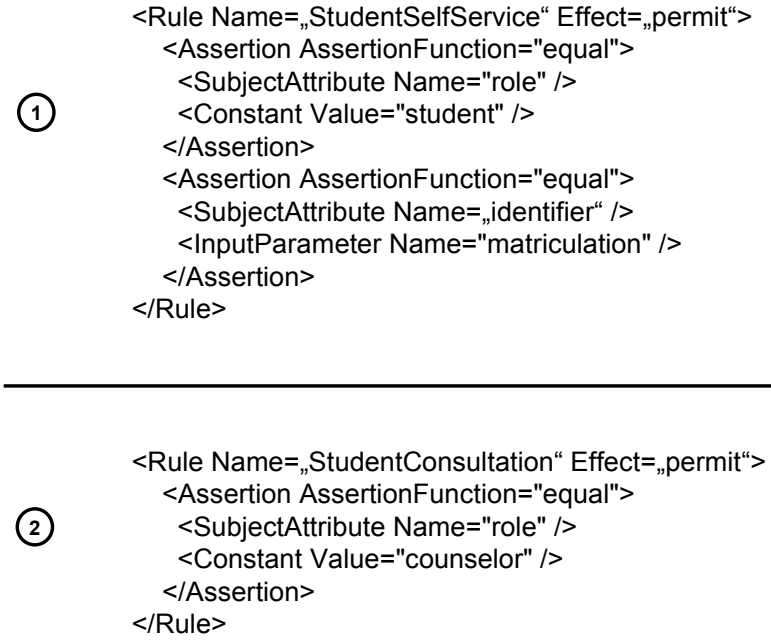


Abbildung 47: Abbildung verschiedener Nutzungskontexte mit WSACML

Auf das Subjekt und die zugehörigen Subjektattribute kann die Zugriffskontroll-Architektur wie in Kapitel 4.1 dargestellt über das beim Dienstaufruf mitgesendete Sicherheitstoken rückschließen. Für den Fall, dass beide `Assertions` auf WAHR evaluieren, greift der definierte `Effect` der Policy, der hier mit `Permit` angegeben ist. Für den zweiten Nutzungskontext, die Studierendenberatung, wird hier auf ein rein rollenbasiertes Kriterium zurückgegriffen. Es wird lediglich das Rollenattribut geprüft. Die Erkennung, ob ein gegebener Nutzungskontext vorliegt, findet wie dargestellt implizit bei der Auswertung der `Assertions` statt. Es besteht die Möglichkeit, ein Standardverhalten zu definieren, wenn keine der angegebenen `Rules` anwendbar sind. Üblicherweise entspricht das einem Verbot, also einem `Effect` mit der Wirkung eines `Deny`.

5.3 Resümee

In diesem Kapitel wurde ein Zugriffskontroll-Modell für Webservice-orientierte Architekturen entwickelt. Dazu wird die klassische Subjekt/Objekt-Relation untersucht und erweitert. Im Bereich der Subjekte gilt es, Geschäftsprozesse und damit verbundene (Geschäfts-)Rollen sowie Attribute der Subjekte in Zusammenhang zu stellen. Hierzu wurden Forschungsergebnisse aus dem Kontext der rollenbasierten Zugriffskontrolle (RBAC) [SC+96] aufgegriffen, die jedoch das Problem der Abbildung von Geschäfts- auf Systemrollen besitzen. Im hier vorgestellten Modell wird das Subjektattribut als das verbindende Element in der Subjekt/Objekt-Relation motiviert. Es ermöglicht einerseits die Nutzung aller zu Verfügung stehender Attribute der digitalen Identität und erlaubt gleichzeitig die unmittelbare Einbeziehung von Geschäftsrollen. Dabei wird eine Geschäftsrolle auf ein oder mehrere Subjektattribute abgebildet. Dies erlaubt einerseits die weitere Nutzung des klassischen RBAC-Modells, gibt andererseits jedoch die Möglichkeit, Ansätze der attributbasierten Zugriffskontrolle (ABAC) zu integrieren. Im Gegensatz zu den Autoren von [YT05], die RBAC und ABAC als gegensätzliche Modelle darstellen, wird hier bewusst eine Kombination aus beidem angestrebt. Dabei wird die „Rolle“ jedoch in den Bereich der Geschäftsprozessmodellierung zurückgedrängt, in dem sie ihren Ursprung hatte: Ein Prozess als Abfolge einzelner Aktivitäten wird durch eine Rolle durchgeführt. Während der automatisierbare Teil des Prozesses auf eine Dienstkomposition abgebildet werden kann, muss sich die Geschäftsrolle in der Zugriffskontroll-Policy wiederfinden. Zur Lösung des n-Quadrat-Problems im Rollenmanagement wird daher die Umsetzung der Geschäftsrolle in Subjektattribute propagiert. Auch im Bereich der Objekte bringt diese Arbeit den Stand der Forschung voran. Während klassische Objekte in Zugriffsmodellen hinsichtlich ihrer angewendeten Operationen (*Read, Write, Update, Execute, ...*) betrachtet wurden, verändert sich beim Übergang in Webservice-orientierte Architekturen die Granularität der zu schützenden Objekte. Daher wurden zunächst mittels eines Dienstmodells die Objekteigenschaften untersucht. Dabei wurde die Einbeziehung der „Aktion“ in klassischen Zugriffskontroll-Modellen gestrichen, da es sich bei Webservices grundsätzlich um „Aufrufe“ (*Execute*) handelt. Die eigentliche Aufrufsemantik spiegelt sich in den Eingabeparametern des Dienstaufrufs wider, die daher in dem hier vorgestellten Zugriffskontroll-Modell explizite Berücksichtigung finden. Das Modell an sich wurde plattformunabhängig hinsichtlich der tatsächlich eingesetzten Sicherheitskomponenten entwickelt und bezieht sich damit rein auf die entwickelte Funktionalität der Facharchitektur. Es ist unmittelbar auf die

Bedürfnisse der Webservice-orientierten Architektur zugeschnitten und erfüllt damit ohne Einschränkungen die Anforderung A3.1.

Da das Zugriffskontroll-Modell nur die Mengen und Relationen darstellt, wird in Kapitel 5.2 mit der *Web Services Access Control Markup Language* (WSACML) eine zugehörige Policy-Sprache vorgestellt. Bei der Umsetzung wurde besonders darauf geachtet, dass eine Einbindung der Sprache und ihrer Artefakte in einen Software-Entwicklungsprozess erleichtert wird. Dies wurde einerseits durch ihre Basierung auf XML geschaffen, was in der Software-Entwicklung gerne eingesetzt wird. Insbesondere lässt sich dies jedoch im eingeführten Konzept der Nutzungskontexte erkennen: Eine Webservice-Operation (Objekt) wird über die zugehörige WSACML-`Policy` mit einer Menge von WSACML-`Rules` verknüpft. Sowohl die Verbindung zur `Policy` als auch zu den einzelnen `Rules` wird bereits während der Software-Entwicklung durchgeführt. Nutzungskontexte (und damit die `Rules`) werden direkt aus Anwendungsfällen abgeleitet und können beliebig ergänzt oder gestrichen werden, ohne dass dies zu Veränderungen der Fachkomponente führt. Dies ermöglicht eine frühzeitige Einbindung in den Software-Entwicklungsprozess (Anforderung A3.2), ohne dass sich für den Fachentwickler eine merkliche Aufwandssteigerung ergibt (Anforderung A1.2).

Nachdem nun die Zugriffskontroll-Architektur und die Policy-Sprache vorgestellt wurden, wird im nächsten Kapitel die Frage behandelt, wie sich eine Einbindung in den Software-Entwicklungsprozess gestalten lässt. Dabei werden Konzepte der modellgetriebenen Architektur (engl. *Model-Driven Architecture*, MDA) aufgegriffen und hinsichtlich des konkreten Umfelds ausgeprägt.

6 Modellgetriebene Entwicklung von Zugriffskontroll-Policies

Neben der Spezifikation einer Zugriffskontroll-Architektur und eines geeigneten Zugriffskontroll-Modells mit zugehöriger Policy-Sprache ist die Einbindung der Wissensträger, insbesondere der Geschäftsanalysten und der Fachentwickler, wichtig (vgl. Anforderungen A1.2 und A3.2). Hierzu bietet sich das Vorgehen der modellgetriebenen Software-Entwicklung an, das auf einer klaren Trennung von Funktionalität und Technik beruht (vgl. Kapitel 2.3.4). Dies erlaubt eine bessere Handhabbarkeit der intrinsischen Komplexität durch Abstraktion. Ziel ist es, durch die Abstraktion eine frühzeitige und formale Einbindung von Zugriffskontroll-Informationen in den Software-Entwicklungsprozess zu erreichen. Dabei stellen unmittelbar auswertbare Zugriffskontroll-Policies im Sinne der Parametrisierung der eingesetzten Zugriffskontroll-Architektur die zu erstellenden Zielartefakte dar.

6.1 Motivation und Ziel

Während man zur Entwicklung von fachfunktionalen Komponenten auf eine Vielzahl bewährter Software-Entwicklungsprozesse zurückgreifen kann [Ba00], wird die IT-Sicherheit, insbesondere auch der Bereich der Zugriffskontrolle nicht ähnlich systematisch behandelt. Anwendungsentwicklung einerseits und die Realisierung von Sicherheitsmechanismen andererseits werden oft getrennt voneinander ausgeführt. Dies führt dazu, dass die Sicherheit in IT-Systemen oftmals nur als nachrangiges Ziel behandelt wird und somit schlecht in das Gesamtsystem integriert wird [Lo03]. In vielen Fällen werden Sicherheitsanforderungen erst kurz vor oder während der Inbetriebnahme eines Software-Systems analysiert und implementiert. Aufgrund von Kosten- und Zeitdruck wird sogar häufig auf wesentliche Sicherheitsaspekte ganz verzichtet, so dass der Betrieb des Software-Systems mit erheblichen Risiken verbunden ist.

Zugriffskontroll-Policies als Parametrisierung einer Zugriffskontroll-Architektur sind die Artefakte, die eine vereinbarungsgemäße Nutzung der Rechnerunterstützung sicherstellen. Während sie derzeit üblicherweise nachgelagert entwickelt werden, wäre eine Einbindung in den Software-Entwicklungsprozess ein großer Fortschritt. Analog der Erfassung der Problemdomäne im Software-

Entwicklungsprozess, die zur Festlegung der Ziele führt (Analyse-Phase), auf der eine Spezifikation der zu erstellen Rechnerunterstützung erstellt (Entwurfs-Phase) und anschließend eine entsprechende Lösung programmiert (Implementierungs-Phase) und ausgeliefert wird (Auslieferungs-Phase), sollen auch die zur Fachfunktionalität gehörenden Sicherheitsanforderungen erfasst und sukzessive verfeinert werden. Ausgangspunkt zur Definition von Zugriffskontroll-Policies stellen auch hier die Geschäftsprozesse dar, die definieren, wer welche Aufgabe in welchem Zusammenhang ausführen darf und soll und welche Dienste dabei in Anspruch genommen werden.

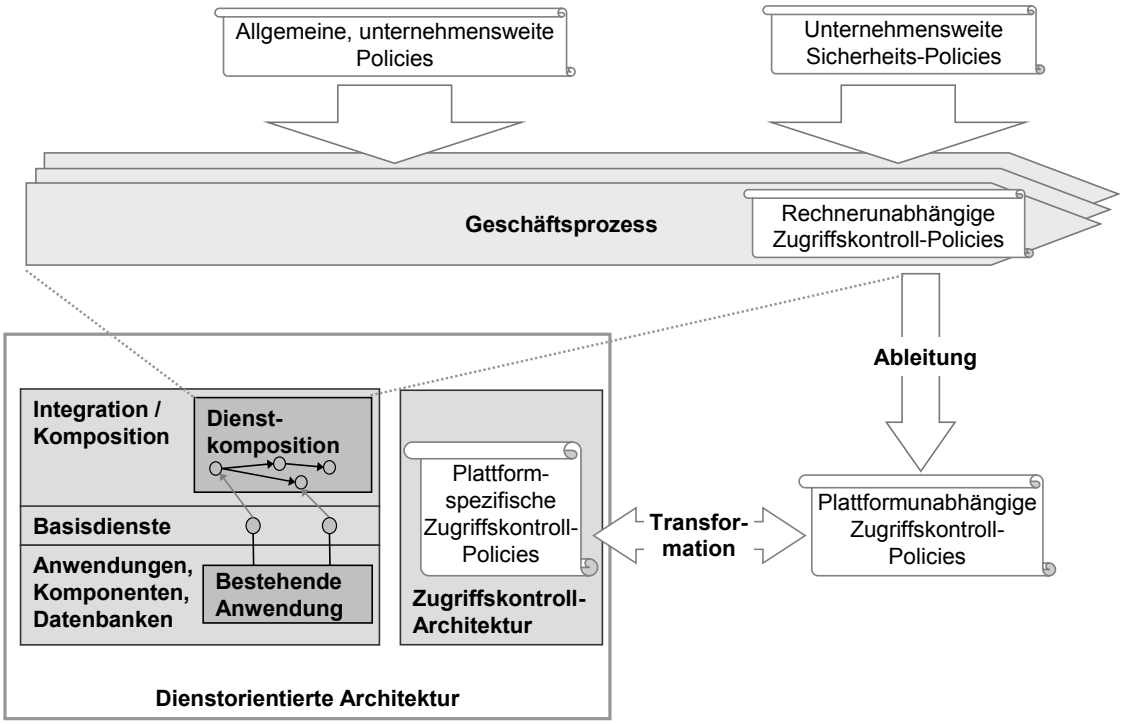


Abbildung 48: Modellgetriebene Entwicklung von Zugriffskontroll-Policies

Im Rahmen eines strukturierten Software-Entwicklungsprozesses sollen daher bereits in der Analyse-Phase parallel zur Erfassung der fachfunktionalen Anforderungen auch die für die Zugriffskontrolle benötigten Informationen erhoben werden. Abbildung 48 stellt diesen Zusammenhang grafisch dar. Im linken Teil wird die dienstorientierte Facharchitektur dargestellt (vgl. Abbildung 1), erweitert um eine Zugriffskontroll-Architektur (vgl. Abbildung 28). Ziel einer dienstorientierten Architektur ist es, Geschäftsprozesse und zugehörige Rechnerunterstützung unmittelbarer aufeinander abzubilden. Hierzu werden die durch Dienste zu unterstützenden Teile der Geschäftsprozesse auf explizit dargestellte Dienstkompositionen abgebildet. Bei der Spezifikation der Geschäftsprozesse müssen

allgemeine, unternehmensweite Policies berücksichtigt werden. Zur Einbeziehung von Sicherheitsanforderungen ist ein analoges Vorgehen durchzuführen. Dazu wird bei der Modellierung von Geschäftsprozessen auch sicherheitsspezifische Policies des Unternehmens auf den jeweiligen Geschäftsprozess ausgeprägt. In Analogie zu den Ansätzen der modellgetriebenen Software-Entwicklung werden rechnerunabhängige Zugriffskontroll-Policies zunächst vollständig systemunabhängig und damit rein aus fachlicher Sicht im Geschäftsprozess spezifiziert [KW+08]. Im Sinne der MDA handelt es sich dabei um ein rechnerunabhängiges Modell (engl. *Computation-Independent Model,* CIM), da erst in einem weiteren Schritt definiert wird, welche Schritte überhaupt auf Rechnerunterstützung abgebildet werden. Eine sich im Zusammenhang mit den rechnerunabhängigen Zugriffskontroll-Policies stellende Herausforderung besteht darin, diese Policies durch weitestgehend modellgetriebene Konkretisierung durchgängig bis auf die Ebenen einer dienstorientierten Architektur umzusetzen. Dazu sollen aus den rechnerunabhängigen Zugriffskontroll-Policies in einem ersten Schritt plattformunabhängige Zugriffskontroll-Policies (engl. *Platform-Independent Model*, PIM) abgeleitet werden, die auf dieser Ebene jedoch bewusst noch keine Spezifika hinsichtlich der eingesetzten Zugriffskontroll-Architektur aufweisen sollen. Dies ermöglicht eine effiziente und flexible Wartbarkeit im Hinblick auf die sich weiterentwickelnden Geschäftsprozesse. Der Schritt von rechnerunabhängigen Modellen (CIM) zu plattformunabhängigen Modellen (PIM) lässt sich heutzutage noch nicht gut automatisieren. Während auf der fachfunktionalen Seite Überlegungen anstehen, welche Dienste wie geschnitten und bereitgestellt werden müssten, um eine optimale Unterstützung zu erzielen, ergeben sich analog für den Bereich der Zugriffskontrolle darüber hinausgehende Fragestellungen. Auch wenn im Übergang von CIM auf PIM menschliches Handeln erforderlich ist, ist dieser Schritt jedoch nicht zu übergehen: Im CIM muss sich letztendlich die Begründung für eine konkret ausgestaltete Zugriffskontroll-Policy auf Ebene des plattformspezifischen Codes (engl. *Platform-Specific Code*, PSC) finden lassen.

Im Kern dieses Kapitels steht der gut automatisierbare Übergang zwischen plattformunabhängigen (PIM) und plattformspezifischen (PSM) Modellen sowie dem damit unmittelbar verknüpften Code (PSC). Abbildung 49 visualisiert diesen Zusammenhang grafisch: Zielartefakt sind die plattformspezifischen Zugriffskontroll-Policies respektive ihre Repräsentation als PSC. Sie bilden die Zugriffsberechtigungen aus fachlicher Sicht ab, sind jedoch unmittelbar abhängig zur eingesetzten Zugriffskontroll-Architektur.

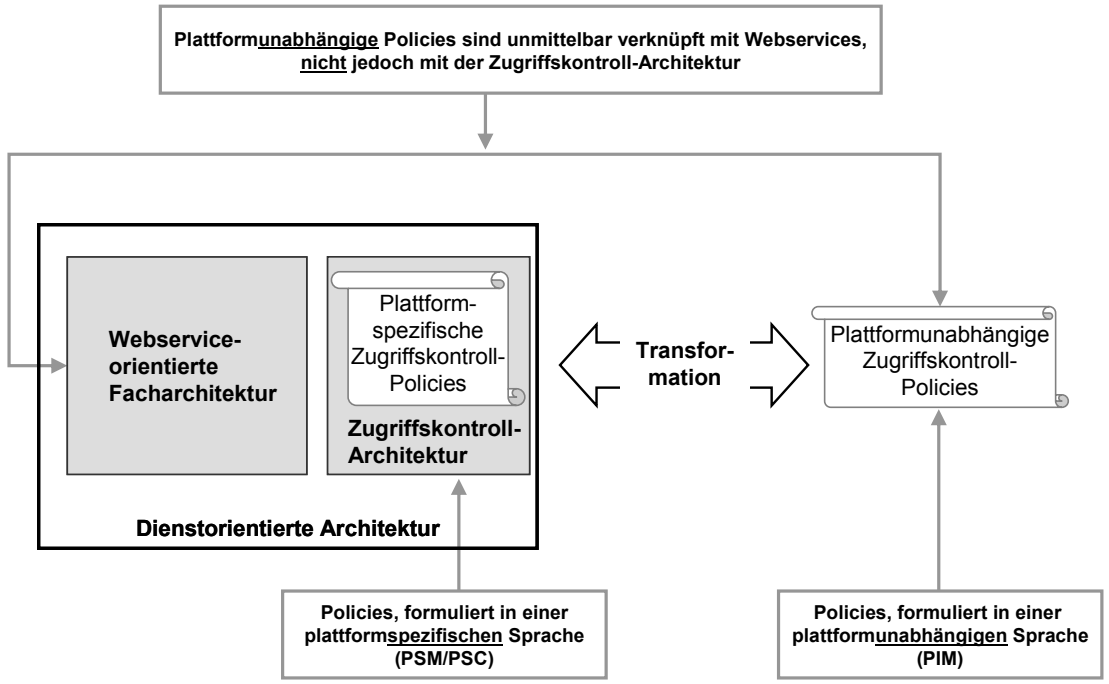
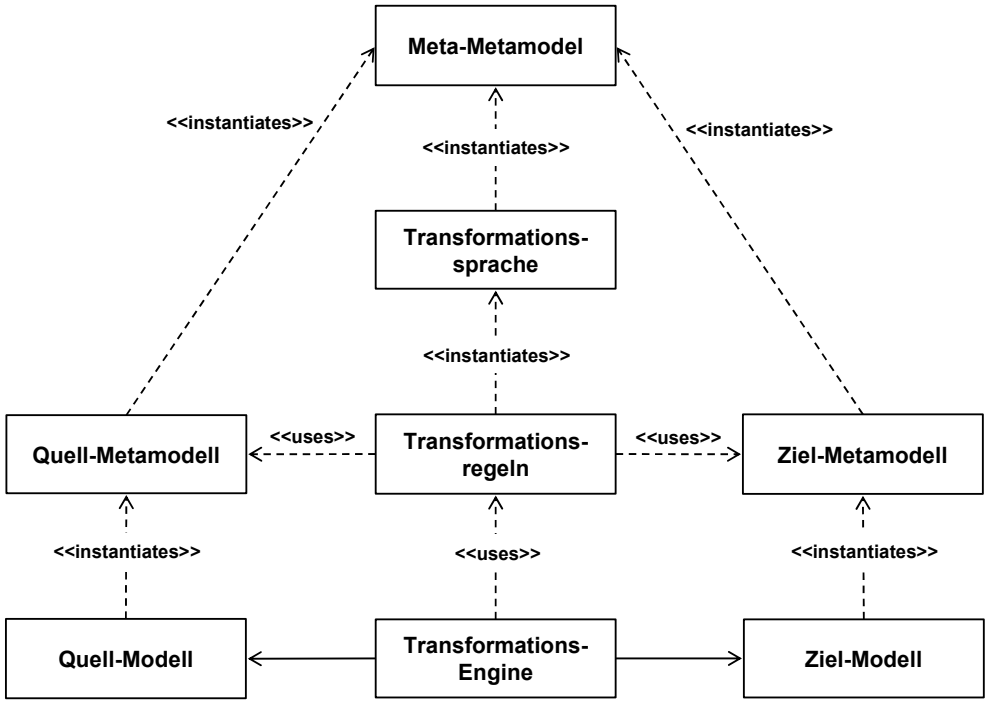


Abbildung 49: Abstraktion auf ein plattformunabhängiges Modell

Da ihr Informationsmodell neben den aus fachlicher Sicht notwendigen Daten auch eine große Zahl produktspezifischer Daten enthält, stellt das plattformunabhängige Modell eine Abstraktion auf die rein fachlichen Zugriffskontroll-Informationen dar. Ziel ist es, neben der Abstraktion eine automatisierte Transformation von PIM zu PSM/PSC zu ermöglichen. Hierzu werden die Konzepte der modellgetriebenen Architektur auf diese Problemstellung ausgeprägt.

Metamodellierung stellt eine wesentliche Grundlage für die modellgetriebene Entwicklung dar [OMG-MDA-Guide]. Abbildung 50 greift dazu die in Abbildung 13 eingeführten Modellebenen der MDA auf und baut darauf ein Transformationskonzept auf. [Ko06] ordnet in dem Schema eine Transformations-*Engine* ein, welche die Transformation der Modelle (M1-Ebene) durchführt und dessen Transformationsregeln die Quell- und Ziel-Metamodellen zur Definition nutzen (M2-Ebene). Die Transformationsregeln wiederum unterliegen einer Transformations-Sprache, welche selbst dem Meta-Metamodell untergliedert ist. Kernaussage ist, dass ein Quell-Metamodell und ein Ziel-Metamodell spezifiziert werden muss. Beide liegen im Sinne der Metamodellierungshierarchie auf der M2-Ebene (vgl. Kapitel 2.3.3).



Nach [Ko06]

Abbildung 50: Transformationskonzept der modellgetriebenen Architektur

Transformationsregeln spezifizieren, wie Elemente der Quellsprache auf Elemente der Zielsprache abgebildet werden. Diese Spezifikation findet auf Ebene der Sprachspezifikation und damit auf M2-Ebene statt. Die für diese Arbeit relevanten Ausgangs-Artefakte sind das Quell-Metamodell, das die Sprachspezifikation der plattformunabhängigen Zugriffskontroll-Policies darstellt. Dies ermöglicht die Validierung von Zugriffskontroll-Policies, die auf Modellebene dem Quell-Modell entsprechen, gegen die Sprachspezifikation (Quell-Metamodell). Das Quell-Metamodell wurde mit der Entwicklung der *Web Services Access Control Markup Language* (WSACML) in Kapitel 5 spezifiziert. Das Ziel-Metamodell steht in unmittelbarem Zusammenhang zur konkret eingesetzten Zugriffskontroll-Architektur. Dabei kann es sich beispielsweise um ein kommerzielles Sicherheitsprodukt (*Commercial Of The Shelf*, COTS) handeln, wie die in Kapitel 3.2.3 vorgestellten Produkte CA eTrust SiteMinder oder den IBM Tivoli Access Manager.

6.2 Festlegung eines konkreten Sicherheitsprodukts

Um die Umsetzung des im vorangegangenen Kapitel vorgestellten Vorgehens darzustellen, wird nun eine konkrete Zugriffskontroll-Architektur als Ziel-

Plattform festgelegt. Exemplarisch und ohne Beschränkung der Allgemeingültigkeit wird dazu CA eTrust SiteMinder ausgewählt, das derzeit im großindustriellen Kontext neben der IBM Tivoli-Produktreihe führend im Einsatz ist.

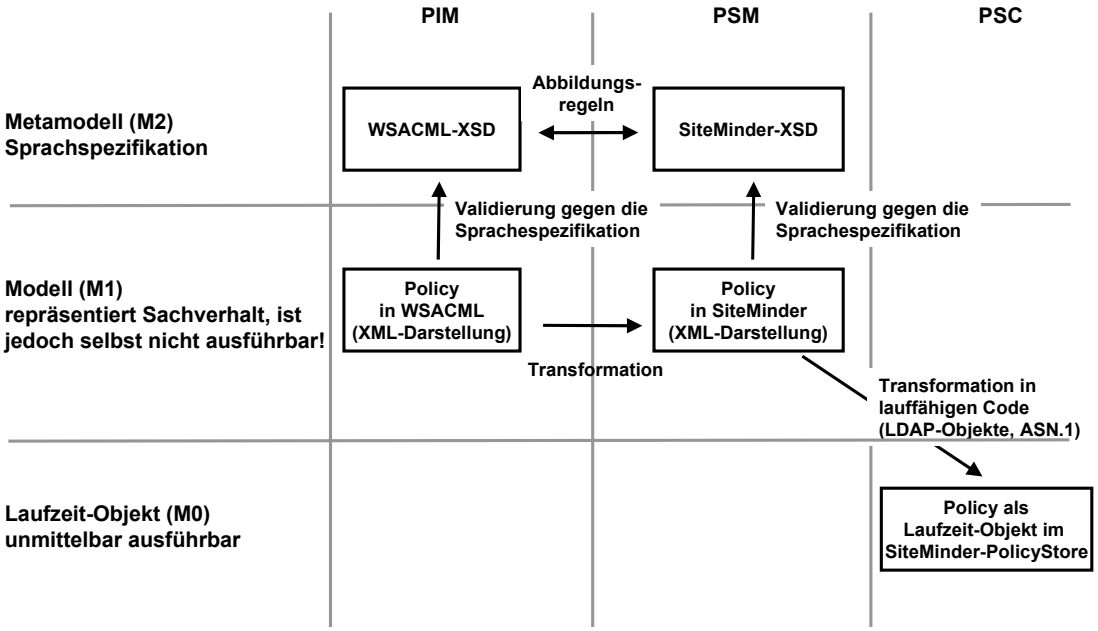


Abbildung 51: Modellgetriebene Policy-Entwicklung

Abbildung 51 stellt eine Konkretisierung der in Abbildung 50 vorgestellten Artefakte dar. Horizontal sind die Schichten der Metamodellierungshierarchie aufgetragen. Auf PIM-Ebene wird die bereits vorgestellte WSACML genutzt. Dabei wird die Sprachspezifikation durch das XML-Schema (XSD) durchgeführt. Modelle der M1-Ebene stellen Zugriffskontroll-Policies dar, die jedoch noch nicht hinsichtlich einer konkreten Zugriffskontroll-Architektur ausgeprägt sind. Ihre syntaktische Korrektheit kann gegen die WSACML-XSD validiert werden. Zum Übergang in die PSM-Ebene muss eine konkrete Zugriffskontroll-Architektur festgelegt werden; hier wird exemplarisch CA eTrust SiteMinder verwendet. Dabei wird das Vorgehen in den Vordergrund gestellt, das auf andere Plattformen übertragbar ist. Zunächst muss ein plattformspezisches Policy-Metamodell erstellt werden. Üblicherweise liefern dies die Hersteller nicht mit dem Produkt aus, es ist implizit im Produkt „verwoben“. Administratoren sollen über produktspezifische Benutzerschnittstellen die entsprechenden Zugriffskontroll-Policies in das System eingeben. Das Zugriffskontroll-Modell und die interne Repräsentation der zugehörigen Policy-Sprache werden als Teil des geistigen Eigentums (engl. *Intellectual Property*, IP) betrachtet und nur sehr abstrakt offengelegt, um den Mitbewerbern die Nachentwicklung zu erschweren.

Daher muss zunächst das Metamodell der Zielplattform herausgearbeitet werden. Konkrete Policies können dann gegen das Metamodell validiert werden. Die PSM-Policies auf M1-Ebene können dann in eine die Repräsentation der Laufzeitumgebung überführt werden. Bei SiteMinder werden die Policies in einem LDAP-basierten Verzeichnis abgelegt. Dies wird im Übergang von PSM auf PSC durchgeführt, der auch von der Modell-Ebene (M1) auf die Ebene der Laufzeitobjekte (M0) wechselt. Die Transformation PSM zu PSC stellt lediglich eine syntaktische Transformation in eine andere Repräsentationsform dar. Das zugrundeliegende Modell muss hierbei nicht angepasst werden. Jedoch sind erst die Policies auf PSC-Ebene (M0) unmittelbar durch die Zugriffskontroll-Architektur auswertbar.

6.3 Entwicklung des PSM-Metamodells

Für den üblichen Fall, dass das Zugriffskontroll-Modell auf PSM-Ebene nicht explizit offengelegt ist, gibt es verschiedene Herangehensweisen, um sich die Syntax und Semantik herzuleiten. Dazu sind Produkthandbücher, die durch das Produkt bereitgestellte GUI, eventuell bereitgestellte Programmierschnittstellen (engl. *Application Programming Interface*, API) und die Datenhaltung des Produkts zu untersuchen.

Ein guter Ausgangspunkt zur Erstellung des PSM-Modells für SiteMinder stellen der „SiteMinder Concepts Guide“ [CA-SM-CG] und die „SiteMinder Policy API“ [CA-SM-PA] dar. Sie schaffen ein Grundverständnis über die Policy-Spezifikation mit SiteMinder. Unter anderem werden die zentralen Elemente `Policy`, `Rule`, `Realm`, `User Directory` und `Action` eingeführt. Dabei besteht folgender Zusammenhang: Eine `Policy` schützt (Web-)-Ressourcen, indem explizit der Zugriff eines Nutzers des `User Directories` über `Rules` erlaubt oder verboten (`Action`) wird. Die Ressourcen werden dabei zu so genannten `Realms` zusammengefasst.

Über mehrere Stufen lässt sich daraus der Kern des SiteMinder-Policy-Metamodells (PSM) ableiten, der in Abbildung 52 in Form eines UML-Klassendiagramms vorgestellt wird [EK+08]. Ausgangspunkt stellt das Element `SmRealm` dar, das über das Attribut `Name` identifiziert wird. Das Attribut `ResourceFilter` beschreibt einen URI-basierten Filter, der die Menge der geschützten Objekte beschreibt.

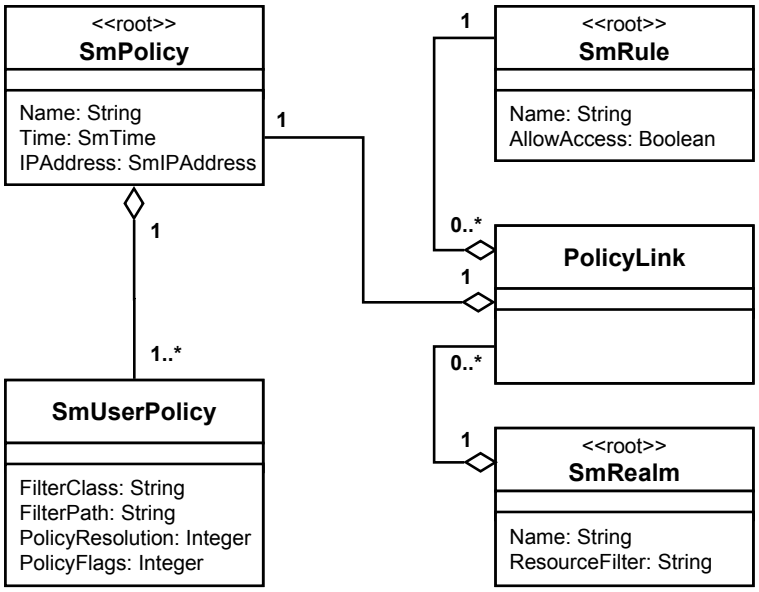


Abbildung 52: Plattformspezifisches Policy-Modell (SiteMinder)

Über das Element `PolicyLink` findet die Verknüpfung mit den Bedingungen statt, die mit den geschützten Objekten verbunden sind und bei einer Zugriffskontroll-Entscheidung berücksichtigt werden müssen. Aussagen, ob ein Zugriff erlaubt oder verboten werden soll, werden durch das Element `SmRule` festgelegt, konkret durch sein Attribut `AllowAccess`. Der `PolicyLink` stellt eine weitere Verknüpfung bereit, die zu Einschränkungen zeitlicher Art und hinsichtlich genutzter IP-Adressen führen kann. Dies wird durch das Element `SmPolicy` abgebildet. Die Betrachtung der Nutzer findet über das verbundene Element `SmUserPolicy` statt. Das Element `FilterPath` ermöglicht die Spezifikation eines LDAP-Suchausdrucks, während das Element `Filter-Class` eine Beschränkung auf LDAP-Objekttypen ermöglicht (vgl. Kapitel 2.2.4). Die Attribute `PolicyResolution` und `PolicyFlags` sind technischer Natur und beschreiben, wie eine Gruppenbeziehung im LDAP-Verzeichnis abgebildet wird. Die vorgenannten Elemente müssen in einer validen SiteMinder-Policy enthalten sein. Aus diesen Informationen lässt sich ein XML-Schema (XSD) erstellen, wie es in [EK+08] und [Kr07a] vorgestellt wird. Das XSD stellt das PSM-Metamodell (M2-Ebene in Abbildung 51) dar.

Exemplarisch stellt Abbildung 53 die in Abbildung 47 unten vorgestellte, plattformunabhängige WSACML-Policy im SiteMinder-Modell dar. Da SiteMinder für den Schutz von Web-Ressourcen im Sinne klassischer Intranet-Anwendungen entwickelt wurde, muss eine Abbildung von Webservice-Operationen auf ein URI-Schema stattfinden. Hierbei ist ein pragmatischer

Ansatz möglich: Jede Webservice-Operation stellt einen eigenen `Realm` mit zugehörigem `ResourceFilter` dar.

```
<SmRealm Name='createToR_Policy'
   ResourceFilter='ToRService/createToR'>
  <PolicyLink>
    <SmRule Name='StudentConsultation'
     AllowAccess='true' />
    <SmPolicy Name='StudentConsultation'>
      <SmUserPolicy
       FilterPath='cn=counselor, ou=roles, dc=com'
       FilterClass='organizationalUnit'
       PolicyResoultion='5'
       PolicyFlags='0' />
    </SmPolicy>
  </PolicyLink>
</SmRealm>
```

Abbildung 53: Exemplarische SiteMinder-Policy in XML-Notation

Dieses Vorgehen skaliert deshalb, da die Menge an Webservice-Operationen im Gegensatz zur Menge an Dateien in einem Webverzeichnis deutlich begrenzter ist. Das in Abbildung 53 durch den `SmRealm` geschützte Objekt ist der in Kapitel 5 eingeführte Notenauszugs-Webservice mit dem Namen `ToRService`, genauer gesagt, seine Operation `createToR`. Die durch den `PolicyLink` definierte `SmRule` trägt den Namen `StudentConsultation` und gibt im Erfolgsfall den Zugriff auf die Ressource frei. Die `SmUserPolicy`, die die zulässigen Subjekte beschreibt, beschreibt die Position in der Verzeichnis-Hierarchie, die auf die Rolle des Studierendenberaters zurückschließen lässt.

6.4 Entwicklung der Transformationsregeln

Um eine Abbildung von WSACML-Policies in SiteMinder-Policies durchführen zu können, müssen zunächst konzeptionelle Abbildungsregeln getroffen werden, die anschließend in einer konkrete Transformationssprache festgeschrieben werden. Dies ermöglicht dann die automatisierte Transformation von Modellen des plattformunabhängigen Modells in plattformspezifischen Code.

Zunächst müssen die grundlegenden Zusammenhänge des plattformunabhängigen Modells (vgl. Abbildung 41) umgesetzt werden:

- Die Verbindung zwischen einem zu schützenden Objekt und seiner Zugriffskontroll-Policy wird in WSACML durch das `serviceOperationBindung` definiert. Diesen Zusammenhang gibt es in SiteMinder nicht, hier wird mit dem *Realms*-Konzept gearbeitet, das eine Menge an Objekten gruppiert. Die Möglichkeit der direkten Zuordnung von Webservice-Operationen zu einer Policy lässt sich dadurch lösen, dass eine Pseudo-URI gebildet wird, die eindeutig eine entsprechende Webservice-Operation identifiziert. Hierzu bietet sich ein Namensschema an, das im einfachsten Fall den Namen des Webservices und der zu schützenden Webservice-Operation enthält. Dies lässt sich dann auf das Attribut `ResourceFilter` des SiteMinder-Elements `smRealm` abbilden.

- Während sich bei WSACML Zugriffsberechtigungen aus Attributen und Attributvergleichen zusammensetzen, bildet SiteMinder diese über LDAP-Objekte und entsprechende Suchausdrücke ab. Identitätsbasierte Abfragen lassen sich über das Attribut `FilterPath` definieren, während Rollenbeziehungen über eine Kombination von `FilterClass` und `FilterPath`-Einträgen erkannt werden können. Dies setzt jedoch Kenntnis über das konkret genutzte Vokabular voraus, das dann in die Transformationsregeln einfließen muss.

Nachdem das Kernmodell der WSACML konzeptionell in das SiteMinder-Metamodell überführt wurde, müssen nun die Elemente und ihre Attribute abgebildet werden. Dies wird ebenfalls zunächst konzeptionell definiert und stellt die Vorstufe für eine Transformationsspezifikation in einer konkreten Transformationssprache wie XSLT dar. Hierzu ist es zu empfehlen, einen Experten des eingesetzten Sicherheitsprodukts, also der Zugriffskontroll-Architektur, zu Rate zu ziehen, da die Semantik der Attribute durch fehlende Offenlegung der Spezifikation durch den Hersteller oft nur aus „sprechenden“ Variablennamen abgeleitet werden kann.

Die einfachsten Transformationen stellen rein namensbasierte Abbildungen dar, die insbesondere die Werte der Attribute nicht verändern. Hierbei werden Elemente und Attribute des Quell-Modells ohne Werteänderung in das Zielmodell umgesetzt. Die wesentlichen Transformationsregeln zwischen den Metamodellen von WSACML und SiteMinder sind in Tabelle 6 dargestellt. Auch die Werteumbenennung für den `Effect` der WSACML-`Rule` fällt hierunter, da es

sich um dieselbe abstrakte Syntax und damit lediglich um eine andere Repräsentation (konkrete Syntax) handelt.

Quell-Metamodell: WSACML	Transformation	Ziel-Metamodell: SiteMinder
Policy		smRealm
Policy.name		smRealm.cn
Policy.serviceOperationBinding		smRealm.smResourceFilter
Rule		smPolicy smRule
Rule.name		smPolicy.cn smRule.cn
Rule.effect	Permit → True Deny → False	smRule.smAllowAccess
SubjectAttribute.identifier		uid
SubjectAttribute.name		cn

Tabelle 6: Einfache Transformationsregeln

Während die Umsetzung der Subjektattributs `Identifikator` und dem Namen des Subjekts auf die LDAP-Objektattribute `uid` und `cn` vergleichsweise einfach gelingt, sind weitere Attribute deutlich aufwendiger umzusetzen. Diese müssen auf die durch die Plattform vorgegebenen Mechanismen, insbesondere die Attribute `FilterPath` und `FilterClass` des Elements `SmUser-Policy`, abgebildet werden. Dies erfordert Entwicklungsaufwand hinsichtlich der Transformationsregeln, der jedoch bei Konstanz des plattformunabhängigen und plattformspezifischen Modells zu vernachlässigen ist, da die Transformation nicht nur einmalig ausgeführt wird, sondern wiederholt durchgeführt werden kann, wie in Kapitel 6.5 gezeigt wird.

Tabelle 7 zeigt exemplarisch komplexe Transformationsregeln auf konzeptioneller Ebene. Der erste Eintrag stellt die Abbildung einer WSACML-`Assertion` dar, die ein Subjektattribut mit einer Konstanten vergleicht. Der zweite Eintrag stellt die Abbildung des Rollen-Attributs eines Subjekts dar, das jedoch abhängig davon ist, wie die Rollenzuweisungen in SiteMinder modelliert wurden. Denkbar ist die Spezifikation als spezieller Teilbaum, über ein gesondertes Attribut oder als ausgezeichnete Objektklasse.

Quell-Metamodell: WSACML	Transformation	Ziel-Metamodell: SiteMinder
Assertion(SubjectAttribute, AssertionFunction ‘equal’, Constant)	smFilterPath = (attribute = constant value), smFilterClass = userattribute, smPolicyResolution = 3	smUserPolicy (smFilterPath)
Assertion(SubjectAttribute ‘role’, AssertionFunction ‘equal’, Constant)	smFilterPath = <DN der Rolle> z.B. ‘CN=counselor, OU=roles, DC=com’ smFilterClass = group smPolicyResolution = 2	smFilterPath

Tabelle 7: Komplexe Transformationsregeln

Ausgehend von den konzeptionellen Transformationsregeln ist der nächste Schritt, diese in einer ausführbaren Transformationssprache zu spezifizieren. Hierzu bietet sich die XSLT an. Abbildung 54 zeigt die Kurzfassung einer Transformationsspezifikation von WSACML zu SiteMinder. Die Langfassung der Transformation findet sich in [Kr07a], weitergehende Beschreibungen zur Erweiterung wurden in [EK+08] gegeben.

```
<xsl:template match=„Rule">
  <PolicyLink>
    <SmRule>
     <xsl:attribute name=„Name">
       <xsl:value-of select='@Name'/>
     </xsl:attribute>
     <xsl:attribute name=„AllowAccess">
       <xsl:value-of select="@Effect = 'permit'"/>
     </xsl:attribute>
    </SmRule>
    <SmPolicy>
     <xsl:attribute name=„Name">
       <xsl:value-of select='@Name'/>
     </xsl:attribute>
     <xsl:apply-templates/>
    </SmPolicy>
  </PolicyLink>
</xsl:template>
```

Abbildung 54: XSLT-basierte Transformation (Kurzfassung)

6.5 Einbettung in den Software-Entwicklungsprozess

Wie bereits in Kapitel 6.1 dargestellt, soll die modellgetriebene Entwicklung der Zugriffskontroll-Policies analog des fachlichen Software-Entwicklungsprozesses

stattfinden. Dazu sollen soweit wie möglich die fachlichen Modelle genutzt und entsprechend erweitert werden. Dies soll zu einer Konsistenz zwischen Fachfunktionalität und zugehöriger Sicherheitsfunktionalität wie Zugriffskontrolle führen.

Die modellgetriebene Architektur (engl. *Model-Driven Architecture*, MDA) der *Object Management Group* (OMG) legt sich nicht auf einen konkreten Software-Entwicklungsprozess fest, sondern stellt in den Mittelpunkt, dass es sich bei den zwischen den Phasen des Entwicklungsprozesses ausgetauschten Artefakte um Modelle handelt, die sukzessive verfeinert werden, bis hin zu ausführbarem Code. Auch diese Arbeit legt sich nicht auf einen konkreten Software-Entwicklungsprozess fest. Für den weiteren Verlauf wird wieder das bereits in Kapitel 5 eingeführte Beispiel der Erstellung eines Notenauszugs (engl. *Transcript of Records*, ToR) genutzt.

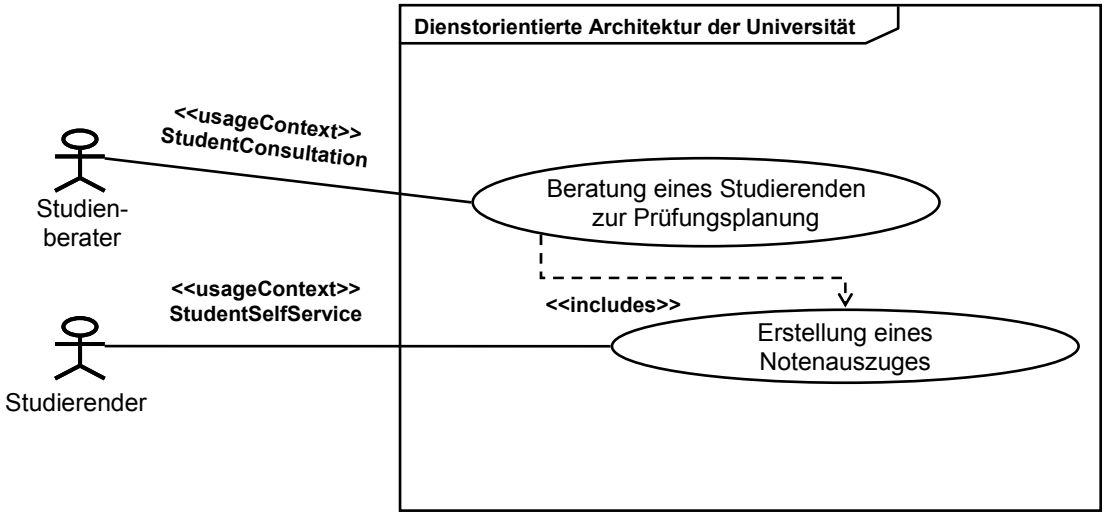


Abbildung 55: Erweitertes Anwendungsfalldiagramm

In der Analyse-Phase arbeiten Software-Entwickler eng mit den Geschäftsanalysten der zu unterstützenden Fachdomäne zusammen. Hierbei werden neben der Geschäftsprozessmodellierung aus fachlicher Sicht auch Geschäftsanwendungsfälle definiert, die in mehreren Geschäftsprozessen auftreten können. Zur formalen Notation bietet es sich an, UML-Anwendungsfalldiagramme hierfür einzusetzen. Ein Anwendungsfall (engl. *Use Case*) definiert eine Interaktion zwischen Akteuren und dem betrachteten System. Das System wird dabei durch eine „harte“ Systemgrenze charakterisiert. Die Interaktionen werden durchgeführt, um ein bestimmtes fachliches Ziel zu erreichen. UML-Anwendungsfalldiagramme wurden vor der formalen Modellierung von dienstorientierten Architekturen

entwickelt. Nichtsdestotrotz lässt sich ihre Semantik sehr gut auch auf diese Architekturen abbilden; dabei kommt der Systemgrenze eine besondere Rolle zu. Dienstorientierte Architekturen möchten bestehende Systemgrenzen überwinden und Funktionalität an Dienstschnittstellen bereitstellen, ohne dass ein technischer Systembezug in den Vordergrund gestellt wird. Daher sollte die Systemgrenze des Anwendungsfalls dem der gesamten, dienstorientierten Architektur entsprechen. Alle Dienste, und damit auch alle Systeme, die Dienst-implementierende Komponenten zur Verfügung stellen, befinden sich innerhalb der Systemgrenze des UML-Anwendungsfalldiagramms. Ein Anwendungsfall stellt dann eine durch die dienstorientierte Architektur bereitgestellte Dienstfunktionalität dar. Anwendungsfalldiagramme eignen sich besser als Ausgangspunkt für den modellgetriebenen Software-Entwicklungsprozess als Geschäftsprozess-Diagramme. Der Bezug zur Systemtheorie zeigt, dass Anwendungsfälle und Geschäftsprozesse jeweils eine andere Sicht auf das zu modellierende System beschreiben: Anwendungsfälle stellen die Frage in den Vordergrund, was die Umwelt vom System erwartet. Das entspricht der *Black-Box*-Sicht des Dienstes. Geschäftsprozesse zeigen, wie das System intern arbeitet, um die Anforderungen der Umwelt zu erfüllen. Während in klassischen Anwendungsfalldiagrammen Rollen mit Anwendungsfällen in Beziehung gestellt werden, wird durch die in dieser Arbeit propagierten erweiterten Anwendungsfalldiagramme ein Nutzungskontext (engl. *Usage Context*) bereits auf Ebene der Anwendungsfall spezifiziert. Ein Nutzungskontext kapselt auf abstrakte Weise den Kontext, in dem die Rolle den Anwendungsfall einsetzt. Die Erweiterung wird durch ein UML-Profil durchgeführt, wie der Autor in [EK+08] gezeigt hat. Es wird dazu das Stereotyp `<<usageContext>>` definiert, das über einen eindeutigen Namen identifiziert wird. Dieses Element bildet das Verbindungselement zwischen Fachfunktionalität und zugehöriger Zugriffskontrolle.

Zur weiteren Präzision wird eine schematisierte Anwendungsfall-Beschreibung eingesetzt, die sich an [OW+03] orientiert (vgl. Abbildung 56). Diese enthält neben dem Namen des Anwendungsfalls und seiner Kurzbeschreibung auch die auslösenden Ereignisse und den Ablauf. Die Eingabeparameter (engl. *Input*), Ergebnisse (engl. *Output*), Vorbedingungen (engl. *Precondition*) und Nachbedingungen (engl. *Effect*) werden ebenfalls erfasst. Sie werden oft auch als englischsprachiges Akronym „IOPE“ dargestellt.

<table>
<tr><td>Name des Anwendungsfalls</td><td colspan="3">Erstellung eines Notenauszuges</td></tr>
<tr><td>Kurzbeschreibung</td><td colspan="3">Ein Notenauszug wird für eine übergebene Matrikelnummer erzeugt</td></tr>
<tr><td rowspan="4">Rollen, Nutzungskontexte und Zugriffsberechtigungen</td><td colspan="3">Standard-Berechtigung: Zugriff verboten</td></tr>
<tr><td>Rolle</td><td>Nutzungskontext</td><td>Zugriffsberechtigung</td></tr>
<tr><td>Studierender</td><td>StudentSelfService</td><td>Zugriff nur auf eigenen Notenauszug erlaubt</td></tr>
<tr><td>Studienberater</td><td>StudentConsultation</td><td>Darf alle Notenauszüge abrufen</td></tr>
<tr><td>Auslösende Ereignisse</td><td colspan="3">Studierender fordert im Studierendenportal einen Notenauszug an
-oder-
Studienberater benötigt einen Notenauszug für Beratungsgespräch</td></tr>
<tr><td>Eingabeparameter</td><td colspan="3">Matrikelnummer</td></tr>
<tr><td>Ergebnisse</td><td colspan="3">Notenauszug (XML)</td></tr>
<tr><td>Vorbedingungen</td><td colspan="3">Studierender muss immatrikuliert sein</td></tr>
<tr><td>Nachbedingungen</td><td colspan="3">-</td></tr>
<tr><td>Ablauf</td><td colspan="3">(1) Prüfen, ob der Studierende immatrikuliert ist.
(2) Daten zu Studierendem, Prüfungsordnung, Modulkatalog, und Prüfungsergebnisse sammeln.
(3) Notenauszug erstellen und zurückliefern.</td></tr>
</table>

Abbildung 56: Der Anwendungsfall “Erstellung eines Notenauszuges”

Eine wesentliche Ergänzung der Anwendungsfall-Beschreibung wird durch die neu hinzugefügte Zeile „Rollen, Nutzungskontexte und Zugriffsberechtigungen“ durchgeführt. Dies greift den durch das Anwendungsfalldiagramm spezifizierten Nutzungskontext, repräsentiert durch das Stereotyp `<<usageContext>>` auf und ergänzt weitere Details. Zu beachten ist, dass sich sowohl das Anwendungsfalldiagramm als auch die jeweiligen Detaillierungen der Analyse-Phase eines Software-Entwicklungsprozesses zuordnen lassen. Das bedeutet, dass in Zusammenarbeit mit Geschäftsanalysten eine Anforderungserhebung durchgeführt wird. Daher muss eine „Gratwanderung“ zwischen der Präzision der Modelle und nicht hinreichender Kompetenz der Beitragenden durchgeführt werden. Mit der detaillierten Anwendungsfall-Beschreibung wird ein Kompromiss zur Verfügung gestellt: Einerseits wird ein wohldefiniertes Schema zur Beschreibung eines Anwendungsfalls genutzt, andererseits können die Einträge „in Prosa“ durchgeführt werden. Dabei wird die Rolle aus dem Anwendungsfalldiagramm übernommen, ebenso entspricht der Nutzungskontext dem Namen des jeweiligen Stereotyps `<<usageContext>>`. Im Feld „Zugriffsberechtigung“ werden dann durch den Geschäftsanalysten weiterführende Angaben getätigt. Trotzdem sich diese Eintragungen nicht unmittelbar für automatisierte Transformationen eignen, bieten sie doch den Ausgangspunkt zur Spezifikation von auswertbaren

Zugriffskontroll-Policies. Die Erstellung eines Glossars zur Schaffung eines gemeinsamen Verständnisses über die zentralen, verwendeten Begriffe kann auch für den Bereich der Zugriffskontrolle hilfreich sein [Ba00].

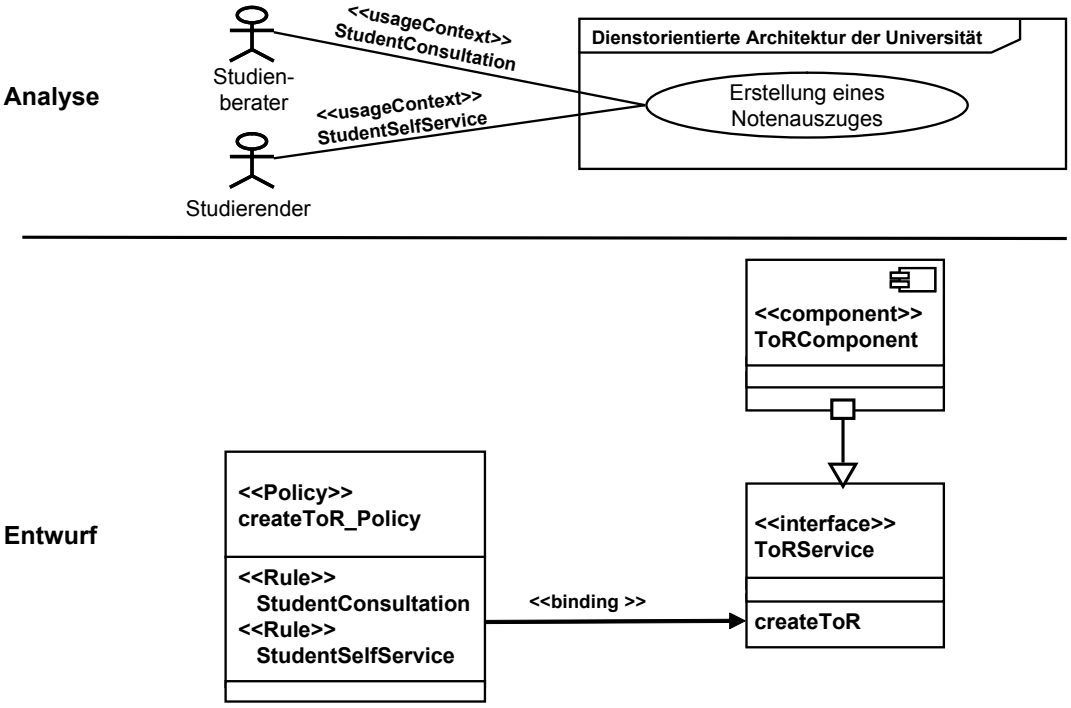


Abbildung 57: Übergang von der Analyse in den Entwurf

Im Übergang von der Anforderungserhebung der Analyse-Phase in den Entwurf ändern sich die im Software-Entwicklungsprozess beteiligten Personen. Der Abnehmer (Kunde) ist nicht mehr beteiligt, stattdessen konzentrieren sich die Architekten und Software-Entwickler darauf, das in der Analyse-Phase erstellte Material (Modelle) weiter zu verfeinern. Dazu wird in Abbildung 57 im oberen Bereich ein verkürztes Anwendungsfalldiagramm dargestellt. Die zwei Rollen „Studienberater“ und „Studierender“ sind sichtbar, genauso wie der Anwendungsfall „Erstellung eines Notenauszuges“, der von der dienstorientierten Architektur bereitgestellt wird. Die Rollen sind mit dem Anwendungsfall über definierte Nutzungskontexte verbunden. Im Übergang in den Entwurf findet oft auch ein sprachlicher Übergang ins Englische statt, falls die Modelle der Analyse-Phase noch zum besseren Verständnis in deutscher Sprache formuliert waren. Für die Entwurfs-Phase bietet sich die Darstellung mittels eines Komponenten-Diagramms an. Der Anwendungsfall wird zu einem Dienst (Webservice) umgesetzt wird, erkennbar durch das Stereotyp `<<interface>>` mit der Operation `createToR` und der Dienst-implementierenden Komponente `ToRCompo-`

nent. Der Nutzungskontext wird ebenfalls übernommen und über ein <<binding>> unmittelbar mit der Dienstoperation verknüpft. Während der Name der Policy sich aus einer Kombination des Names des Webservice und der entsprechenden Operation zusammensetzen lässt, werden die Nutzungskontexte in Attribute der <<Policy>> des Typs <<Rule>> umgesetzt.

Die unmittelbare Verknüpfung zwischen Zugriffskontroll-Policy und Webservice-Operation ermöglicht nun die Auftrennung in der Entwicklung. Während Fachentwickler nun Aktivitäts- und Sequenzdiagramme in ablauffähigen Programmcode überführen, können Sicherheitsingenieure die Zugriffskontroll-Policies spezifizieren. Ausgangspunkt bilden dabei die <<Policy>> mit den entsprechenden <<Rules>>. Sie entsprechen unmittelbar der Syntax und der Semantik der WSACML, wie in Kapitel 5.2 eingeführt.

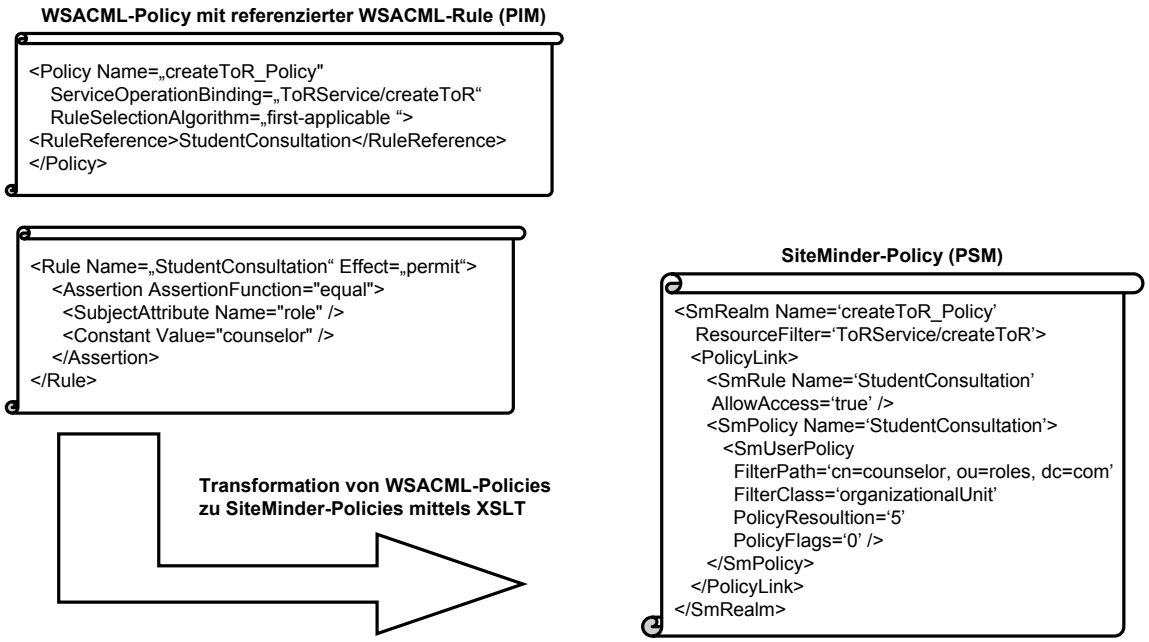


Abbildung 58: Transformation zu plattformspezifischen Policies

Der erste Schritt im Übergang von dem in Abbildung 57 unten links dargestellten Rumpf einer Zugriffskontroll-Policy findet mit der Erstellung einer WSACML-konformen Policy statt. Hierzu ist neben der Sprachspezifikation selbst auch das Vokabular der Domäne zu berücksichtigen, das zur Erstellung der die Nutzungskontexte widerspiegelnden WSACML-Rules erforderlich ist (vgl. Abbildung 45). Das Vokabular schlägt sich auch in der (einmaligen) Erstellung der XSLT-basierten Transformation nieder.

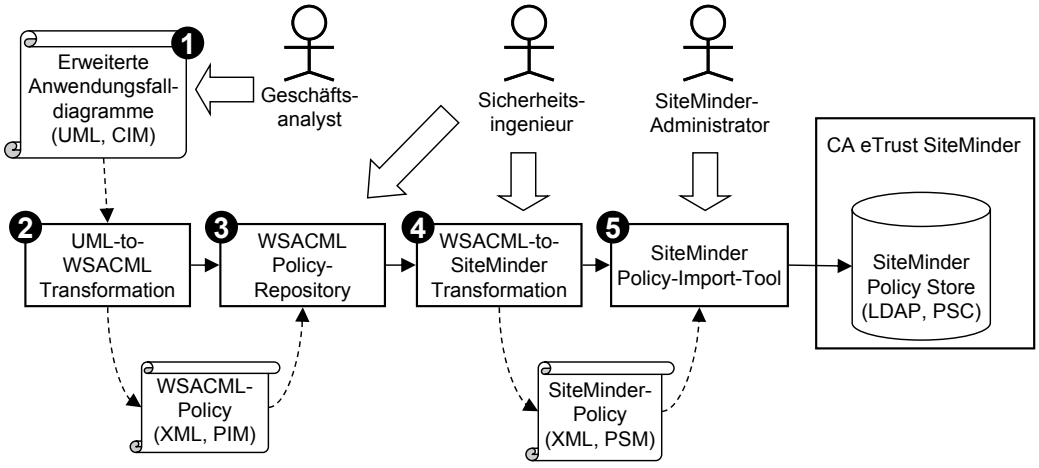


Abbildung 59: Modellgetriebene Policy-Entwicklung – Ablauf und Rollen

Das Vorgehen wird im Ganzen in Abbildung 59 vorgestellt. Geschäftsanalysten wirken bei der Erstellung von um Zugriffskontroll-Informationen erweiterten UML-Anwendungsfalldiagrammen mit (Schritt 1). Da in dieser Phase auch Prozessschritte erfasst werden, die später nicht auf Rechnerunterstützung abgebildet werden, lässt sich dies der CIM-Ebene der MDA zuordnen. Aus diesen Anwendungsfalldiagrammen lassen sich Rümpfe von WSACML-`Policies` erzeugen (Schritt 2). Insbesondere die Policy selbst mit ihrer Verzeigerung zu den jeweiligen WSACML-`Rules` lässt sich fast vollständig automatisiert generieren. Die einzelnen WSACML-`Rules` werden dann feinspezifiziert. Die `Rules` werden in einer zentralen Datenablage gespeichert, was eine Wiederverwendung der `Rules` selbst in anderen `Policies` vereinfacht (Schritt 3). Diese `Policies` können dann automatisiert in das Zielmodell transformiert werden (Schritt 4). Die finale Auslieferung (engl. *Deployment*) wird dann durch den SiteMinder-Administrator durchgeführt, der die XML-basierten `Policies` in den SiteMinder Policy Store einlädt (Schritt 5).

6.6 Resümee

Zugriffskontroll-Policies werden idealerweise genauso strukturiert entwickelt wie die Fachkomponenten, die sie vor unautorisierten Zugriffen schützen sollen. Daher sollten analog die Geschäftsprozesse den Ausgangspunkt der Erstellung bilden. Geschäftsprozesse zeigen in rechnerunabhängiger Darstellung den definierten Ablauf und die beteiligten Rollen auf. Bereits auf dieser Ebene lassen sich (zumindest informell) Informationen zusammenstellen, welches Subjekt unter welchen Randbedingungen gewisse Handlungen durchführen darf. In dieser

Arbeit werden UML-Anwendungsfalldiagramme erweitert, um als Ausgangspunkt der formalen Darstellung der Verbindung zwischen Fachfunktionalität und Zugriffskontrolle zu dienen. Die in diesem Diagrammtyp zentrale Systemgrenze stellt die Grenze der Webservice-orientierten Architektur dar, während die Anwendungsfälle mit den durch die WSOA bereitzustellenden Diensten korrelieren. Während sich die aus dem Geschäftsprozess ableitenden Rollen traditionell am linken Rand darstellen lassen, wird die Beziehung zwischen Rolle (Subjekt) und Anwendungsfall (Dienst, Objekt) um den sogenannten Nutzungskontext erweitert. Ein Subjekt nutzt ein Objekt im Rahmen eines Geschäftsprozesses. Aus Sicht der Zugriffskontrolle stellt dies den Nutzungskontext dar. Bereits in der verfeinerten Darstellung des Anwendungsfalls, repräsentiert durch eine strukturierte Darstellung, die unter anderem Vor- und Nachbedingungen sowie nähere Beschreibungen zum Anwendungsfall selbst gibt, werden Rollen und Zugriffsberechtigungen zu den Nutzungskontexten hinzugefügt. Dies führt dazu, dass bereits in der Analyse-Phase des Software-Entwicklungsprozesses, in die üblicherweise auch der Kunde involviert ist, die für Zugriffskontrolle relevanten Daten explizit erfasst werden. Dies bildet dann den Ausgangspunkt, um mit Methoden der modellgetriebenen Architektur effektiv auswertbare Zugriffskontroll-Policies zu erstellen. Dazu wurde die im vorangegangenen Kapitel vorgestellte, plattformunabhängige Policy-Sprache WSACML genutzt. Sie stellt eine Abstraktion einer Zugriffskontroll-Policy dar, die über explizit definierte Transformationen vollständig automatisiert in das Zielformat des eingesetzten Sicherheitsprodukts umgeformt werden kann (vgl. Anforderung A4.1). Diese Abstraktion erleichtert die Einbeziehung des Fachentwicklers und schafft Unabhängigkeit hinsichtlich des eingesetzten Sicherheitsprodukts.

Die Ergebnisse dieser Arbeit wurden im Rahmen eines Projekts bei einem großen deutschen Automobilhersteller entwickelt und dort zum Einsatz gebracht. Hierbei zeigte sich, dass gerade die Abstraktion der Komplexität der Sicherheitsprodukte einen immensen Mehrwert für alle an der Software-Entwicklung beteiligten Rollen mit sich bringt: Die Anforderungen der Fachabteilungen werden auf formalisiert aufgenommen und die Sicherheitsingenieure können unmittelbar auf diesen Artefakte aufsetzen.

7 Semantische Integration von Benutzerdaten

Grundlage für die Einführung einer dienstorientierten Architektur ist ein strukturiertes und effizientes Identitätsmanagement. Die Verwaltung und Bereitstellung der digitalen Identitäten stellen dabei eine zentrale Voraussetzung zur Durchführung von Zugriffskontrolle dar. Benutzerdaten liegen üblicherweise verteilt in den entsprechenden Datenquellen der verschiedenen Anwendungssysteme vor. Analog der Verschattung der Komplexität der Anwendungssysteme hinter Dienstschnittstellen soll auch die Verteilung der Benutzerdaten über verschiedene Systeme hinweg für den Dienstnehmer transparent sein. Aus Sicht der in Kapitel 4 eingeführten Zugriffskontroll-Architektur ist es notwendig, eine einheitliche Sicht auf die Benutzerdaten zu haben. Das Benutzerverzeichnis muss dazu einen Dienstzugangspunkt auf die entsprechenden Daten bereitstellen. Da bei der Einführung dienstorientierter Architekturen die Altsysteme nicht abgelöst, sondern lediglich standardisierte und geschäftsrelevante Schnittstellen angebracht werden, müssen die in den Altsystemen integrierten Benutzerverzeichnisse mit dem der dienstorientierten Architektur synchron gehalten werden. Dies resultiert nicht nur in einem initialen Aufwand eines einmaligen Datentransfers. Bei Neueinträgen, Änderungen und Löschungen müssen entsprechende Benachrichtigungen erstellt und an alle relevanten Datenquellen verteilt werden. Für diese Informationsverteilung hat sich bisher kein Standard durchgesetzt; hinzukommt, dass jedes System seine eigene Repräsentation zur Datenspeicherung nutzt. Zur Verbesserung werden in diesem Kapitel zwei Beiträge geliefert: Einerseits wird eine erweiterbare Personen-Ontologie entwickelt, die eine semantische Beschreibung der Attribute einer digitalen Identität ermöglicht und damit die Grundlage für eine semantische Integration der syntaktisch heterogenen Benutzerdaten bildet (vgl. Kapitel 7.1). Dies ist durchführbar, da im Bereich der Benutzerdaten die Heterogenität auf syntaktischer beziehungsweise struktureller Ebene liegt, während die semantische Ebene weitestgehend homogen ist. Diese erweiterbare Personen-Ontologie bildet den Grundstein für den zweiten Beitrag: eine Synchronisations-Architektur, die auf der Basis der Ontologie Benutzerverzeichnisse synchronisiert (vgl. Kapitel 7.2). Die nachfolgenden Inhalte wurden im Rahmen einer Projektarbeit entwickelt und sind ausführlich in [EL+07] sowie [La06] beschrieben.

7.1 Die Personen-Ontologie

7.1.1 Hintergrund

Der Wunsch der Unternehmen, ihre Geschäftsprozesse zu automatisieren und bestehende Anwendungssysteme besser zu integrieren, hat zur Migration in dienstorientierte Architekturen geführt. Identitätsmanagement stellt dazu eine wesentliche Voraussetzung dar [Ku05]. Wesentliche Kernprozesse des Identitätsmanagements stellen die Erstellung, die Pflege und die Nutzung von digitalen Identitäten dar [Bl05]. Zur Bereitstellung dieser Daten gehört insbesondere auch die Synchronisation von verteilten Datenquellen [Wi05]. Die Daten werden bei der Durchführung von Zugriffskontroll-Entscheidungen durch die dienstorientierte Architektur benötigt (vgl. Abbildung 28). Die notwendigen Prozesse des Identitätsmanagements sollten jedoch vor der Zugriffskontroll-Architektur verschattet werden. Dies wird unter dem Schlagwort *Identity as a Service* zusammengefasst: Es soll eine einheitliche Sicht auf die digitalen Identitäten geben (vgl. Abbildung 14). Eine digitale Identität ist dabei eine Menge von Attributen (Schlüssel/Wert-Paaren, vgl. Abbildung 7). Die Integration von Daten kann auf verschiedenen Ebenen erfolgen. Üblicherweise erfolgt die Integration auf einer syntaktischen Ebene. Dabei werden individuelle Konnektoren entwickelt und Abbildungsregeln zwischen einem Quell- und Zielsystem fest in das Synchronisationsprodukt fixiert (vgl. Kapitel 3.5). Dies führt zu quadratischem Aufwand bei der Einbindung einer zusätzlichen Datenquelle in die Synchronisation. Daher liegt in diesem Abschnitt das Hauptaugenmerk auf der Explizitmachung der Semantik der Daten. Zur Darstellung von Semantik bietet sich die Nutzung von Ontologien an [CJ+02]. Ontologien helfen dabei, die Bedeutung von Daten (Semantik) von ihrer Repräsentation zu trennen. Die Semantik wird aus den bestehenden Datenquellen extrahiert und kann unabhängig von der Speicherform geändert werden. Dies bringt mehrere Vorteile mit sich: Zunächst stellt eine Ontologie eine einheitliche Nomenklatur für die Domäne zur Verfügung. Dabei wird implizit eine semantische Eindeutigkeit erreicht, da alle Elemente (engl. *Entity*) einer Ontologie einen eindeutigen Namen sowie eine semantisch wohldefinierte Bedeutung besitzen. Dies ermöglicht es, dass syntaktisch identische Elemente (beispielsweise Elemente gleichen Namens), die semantisch jedoch ungleich sind, fälschlicherweise für bedeutungsgleich erachtet werden. Des Weiteren ermöglicht der Einsatz von Ontologien eine flexiblere Integration, da die feste Verdrahtung im Sinne der Abbildungsregeln zwischen zwei Synchronisations-Endpunkten zugunsten einer externalisierten, semanti-

schen Abbildung aufgegeben wird. Ontologien ermöglichen auch die Durchführung von Konsistenzprüfungen. Zur Implementierung von Ontologien bietet sich die Nutzung von OWL an (vgl. Kapitel 2.4).

7.1.2 Herausforderungen

Die erste Hürde bei der Integration von Benutzerverzeichnissen stellt die Heterogenität der Datenquellen dar. Dieses Problem wurde bereits im Kontext verteilter Datenbanken diskutiert. [KS96] unterscheidet Heterogenität wie folgt: einerseits in strukturelle und syntaktische Heterogenität, die als schematische Heterogenität bezeichnet wird, andererseits in semantische Heterogenität, die als Daten-Heterogenität bezeichnet wird. Strukturelle Heterogenität impliziert, dass unterschiedliche Schemata (Strukturen) zur Datenspeicherung genutzt werden, wohingegen sich bei semantischer Heterogenität der Informationsgehalt und damit die Bedeutung eines Datums unterscheidet. Um eine semantische Integration von Daten jeglicher Art zu ermöglichen, muss die Bedeutung (Semantik) der ausgetauschten Daten systemübergreifend festgeschrieben werden. Die Umsetzung solcher semantischer Interoperabilität wird vereinfacht, wenn die semantischen Domäne die gleiche ist [CJ+02] [Pa02]. Obwohl gerade in dienstorientierten Architekturen eine Vielzahl verschiedener Anwendungssysteme zum Einsatz kommen, die hinsichtlich ihrer Fachdomäne sehr heterogen sein können, so ist jedoch der durch diese Arbeit betrachtete Bereich ihrer Benutzerdaten semantisch sehr homogen. Die Semantik lässt sich daher gut durch Ontologien abbilden.

Für den Einsatz von Ontologien gibt es nach [WV+01] unterschiedliche Herangehensweisen: die Einführung einer gemeinsamen Ontologie (engl. *Single Ontology Approach*), die Verwendung mehrerer unabhängiger Ontologien (engl. *Multiple Ontology Approach*) sowie hybride Ansätze. Der Einsatz einer gemeinsamen Ontologie stellt ein gemeinsames Vokabular zur Verfügung, das zur Beschreibung verwendet wird. Alle Datenquellen werden zu dieser gemeinsamen Ontologie in Beziehung gesetzt. Dieses Vorgehen bietet sich an, wenn sich die Datenquellen mit derselben Domäne befassen und eine ähnliche Sichtweise auf diese Domäne haben. Bei der Verwendung mehrerer Ontologien wird jede Datenquelle individuell und damit unabhängig voneinander beschrieben. Dies verschiebt jedoch die Komplexität zu einer „Verbindungs-Ontologie“, die die jeweiligen Ontologien in Überdeckung bringt. Hybride Ansätze arbeiten mit lokalen Ontologien, versuchen jedoch, eine globale Ontologie im Sinne eines gemeinsamen Vokabulars oder Glossars zu nutzen. Diese globale Ontologie

bildet dann für den (menschlichen) Entwickler bei der Erstellung der lokalen Ontologien eine Leitlinie. Für den Einsatz im Kontext dieser Arbeit bietet sich die Verwendung einer gemeinsamen Ontologie (engl. *Single Ontology Approach*) an.

Es gibt ein grundsätzliches, gemeinsames Verständnis über digitale Identitäten [Wi05], jedoch wird die Repräsentation und Speicherung in den verschiedenen Anwendungssystemen unterschiedlich gehandhabt. Die Probleme, die dabei zu lösen sind, sind vielfältiger Natur, unter anderem auch Namenskonflikte der Attribute [Go97]. Der einfachste Fall eines Namenskonflikts stellt die Verwendung von unterschiedlichen Attributnamen in unterschiedlichen Systemen für dasselbe Attribut dar, beispielsweise `givenName='Hans'` und `firstName='Hans'`. Dieser Zusammenhang kann durch die explizite Definition von Synonym-Beziehungen in der Ontologie zum Ausdruck gebracht werden. Eine weitere Herausforderung stellt sich durch unterschiedliche Datengranularität. Ein Beispiel hierfür sind die (Einzel-)Attribute `firstName` und `lastName` in einer Datenquelle, während in einer anderen Datenquelle lediglich der `commonName` als Kombination aus beiden Attributen gespeichert ist. Auch dieser Sachverhalt lässt sich mittels Ontologien darstellen.

7.1.3 Entwicklung der Ontologie

Zur Entwicklung einer Ontologie gibt es mehrere Möglichkeiten [Pa02]. Das für diese Arbeit sinnvollste Vorgehen stellt ein *Bottom-Up*-Ansatz dar, bei dem die Schemata der unterschiedlichen Datenquellen aus Ausgangspunkt genommen werden. Daraus lässt sich ein Grundgerüst einer Ontologie erstellen, das zunächst keine Spezifika eines konkreten Anwendungssystems beinhaltet. Hierzu bieten sich die LDAP-Objektschemata an [RFC-2251], auch wenn sie ihre Semantik lediglich in „sprechenden“ Attributnamen mit zugehöriger Prosa-Beschreibung spezifizieren. Aus praktischer Sicht macht es jedoch keinen Sinn, Aufwand in die Erstellung eines globalen LDAP-Objektschemas für digitale Identitäten zu investieren. Dies müsste in jeder beteiligten Datenquelle eingepflegt werden und würde bei Schema-Änderungen immensen Aufwand erzeugen [Zö05]. Auch Ansätze wie [DC+02], [UMBC-PO] und [SU06] stellen einen Ausgangspunkt dar. Die HR-XML-Initiative [HR-XML], die aus dem Bereich des Personalwesens entstanden ist, gibt eine Übersicht über eine Vielzahl an möglichen Attributen sowie eine textuelle Beschreibung. Es findet zwar keine formale Spezifikati-

on statt, nichtsdestotrotz lassen sich die angebotenen Daten als Grundlage für den Aufbau einer Personen-Ontologie nutzen.

Die Entwicklung der Personen-Ontologie lehnt sich an das allgemeine Vorgehen zur Ontologie-Erstellung aus [NM01] an. Dazu werden zunächst die wichtigsten Attribute von Personen erfasst. Als Heuristik für die Wichtigkeit wird dabei das Auftreten in unterschiedlichen Referenzen (LDAP, HR-XML, s.o.) verwendet. Wichtige Elemente, die dabei ins Auge stechen, sind `commonName`, `address`, `credentials`, `title`, `email` und `telephone`. Diese Liste ist nicht abschließend, sondern stellt lediglich einen Ausgangspunkt zur Ontologie-Erstellung dar. Im nächsten Schritt werden die Elemente in (einfache) Attribute (engl. *Datatype Property*) und komplexe Elemente aufgetrennt. Die komplexen Elemente werden als Klassen definiert und mittels OWL [W3C-OWL] in eine Hierarchie einsortiert. Beispiele für komplexe Elemente sind `commonName`, `address` und `credentials`. Zusätzlich werden die Beziehungen zwischen diesen Klassen über Objekteigenschaften (engl. *Object Properties*) formalisiert. Dabei werden Klassen wir `commonName` und `cn` als semantisch äquivalent charakterisiert (`owl:equivalentClass`). Auch Klassen mit Divergenzen wie `userPassword` aus dem LDAP-Objektschema und `password` aus HR-XML werden entsprechend spezifiziert (`owl:disjointWith`). Abschließend werden die Attribute jeder Klasse mittels des OWL-Konstrukts `owl:DatatypeProperty` definiert. Dabei lag der Fokus auf der Definition der semantisch gleichen Attribute, da diese Information für den späteren Synchronisationsprozess von entscheidender Bedeutung ist.

Abbildung 60 zeigt einen Ausschnitt der Personen-Ontologie, grafisch modelliert mittels DLG² (*Directed Labeled Graph Two*) [WA05]. DLG² ist eine Notation, die RDF-Dateien (*Resource Description Framework*) und damit OWL-Modelle grafisch darstellen kann. In der Abbildung werden die relevanten Konstrukte in einer gemeinsamen Übersicht dargestellt: Zentrales Element ist die Person mit einer Menge von Attributen (`owl:DatatypeProperty`). Diese werden in optischer Analogie zu UML-Klassendiagrammen unmittelbar der Klasse `Person` zugeordnet.

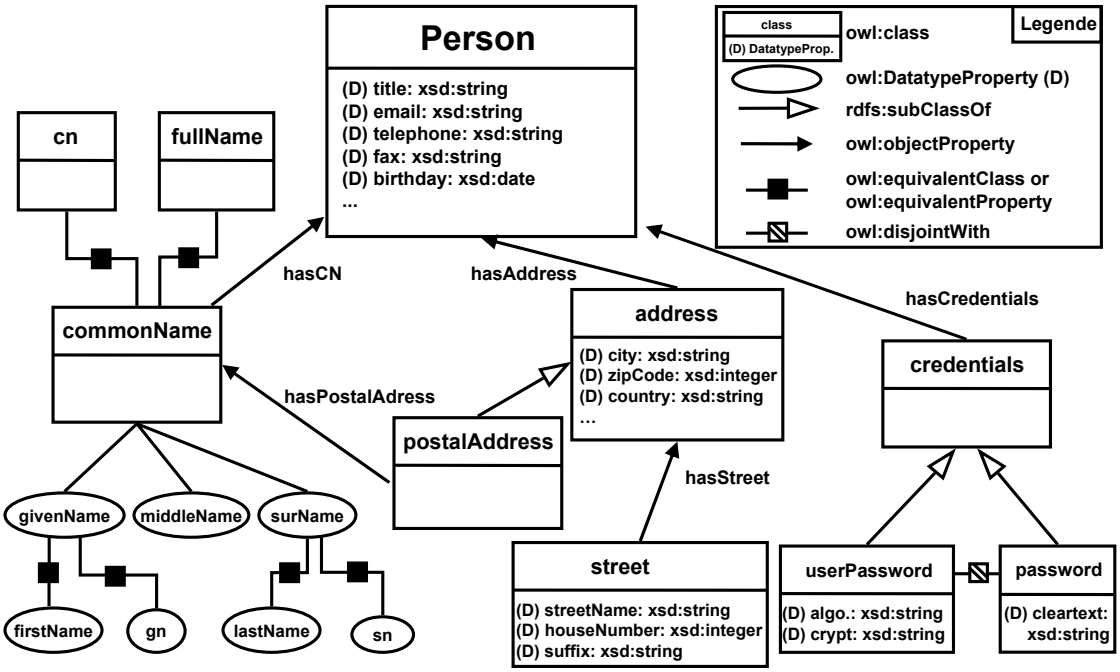


Abbildung 60: Die Personen-Ontologie (Auszug in DLG²)

Die Klasse `Person` wird über `hasAddress` mit der Klasse `address` verbunden (`owl:objectProperty`). Die Klasse `address` ist wiederum eine Super-Klasse für weitere Klassen wie `postalAddress` oder `street`. Die Klasse `postalAddress` ist mit dem `commonName` durch die Objekteigenschaft `hasCN` verbunden. Verbindungslinien mit ausgefülltem Quadrat repräsentieren die Beziehung `owl:equivalentClass` oder `owl:equivalentProperty` und stellen damit eine Gleichheit dar. Auch Ungleichheit lässt sich explizit formulieren. Als Beispiel hierfür dienen die Klassen `userPassword` und `password`, die explizit als ungleich definiert wurden (`owl:disjointWith`). Wegen der umfangreichen Darstellung in XML wird auf den vollständigen Abdruck an dieser verzichtet. Sie besteht aus neun Klassen sowie 71 Datentype- und Objekt-Eigenschaften (engl. *Datatype and Object Properties*). Die OWL-Datei besteht aus circa 800 Zeilen und hat eine Größe von 40 Kilobytes. Die Personen-Ontologie wurde mit dem Werkzeug „Protégé“ [SMI05] entwickelt. Zur Verifikation der Konformität zu OWL-DL (vgl. Kapitel 2.4) wurde der OWL-Validator „WonderWeb“ [BV03] genutzt. Die Konsistenz der Ontologie wurde durch den *Reasoner* „Racer DIG“ (*Description Logic Implementation Group*) [RACER] geprüft. Eine detaillierte Darstellung der Personen-Ontologie wurde durch den Autor in [EL+06] gegeben.

7.1.4 Flexible Erweiterbarkeit

Wie eingangs erwähnt, ist die vorgestellte Personen-Ontologie ein Grundgerüst, das flexibel erweiterbar ist. Daher müssen sich neu an der Synchronisation teilnehmende Datenquellen leicht integrieren lassen. Dazu muss die gemeinsam genutzte Personen-Ontologie erweitert werden. Wenn die Semantik der hinzuzufügenden Datenquelle bereits durch eine Ontologie beschrieben wurde, kann diese als Ausgangspunkt zur Erweiterung genutzt werden. Falls nicht, muss dies nachgeholt werden. Darauf aufbauend kann die semantische Verknüpfung mit der Personen-Ontologie durchgeführt werden. Wenn die Menge an hinzuzufügenden Attributen gering ist, kann dies manuell geschehen. Es gibt auch Möglichkeiten, die Zusammenführung automatisiert oder halb-automatisch durchzuführen [HG00]. Für den Fall, dass zwei semantisch unterschiedliche Attribute aus unterschiedlichen Datenquellen den gleichen Namen tragen, kommt das Konzept der Namensräume (engl. *Namespaces*) zu tragen. Dabei kann als Namensraum der Name der jeweiligen Datenquelle dienen.

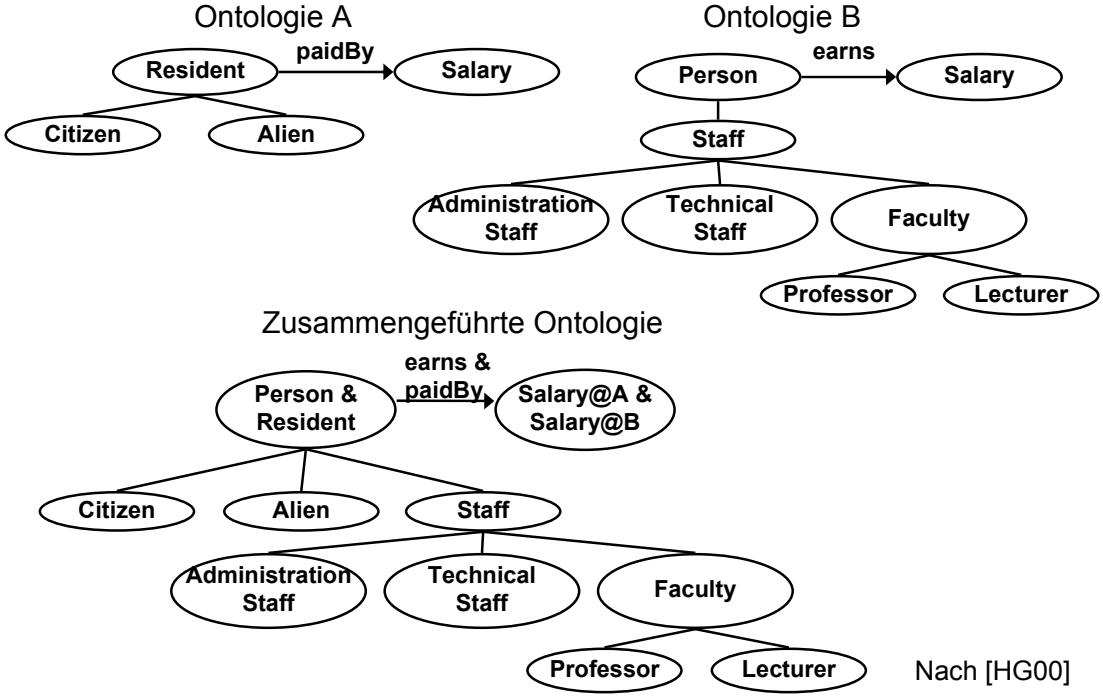


Abbildung 61: Zusammenführung von Ontologien (Beispiel)

Abbildung 61 zeigt beispielhaft, wie Ontologien zusammengeführt werden können [HG00]. Ausgangspunkt ist die gemeinsame Klasse `Salary` (dt. Gehalt). Von einer vollautomatischen Zusammenführung ist zwar abzuraten, trotz

allem vereinfacht die daraus resultierende Werkzeugunterstützung selbst die halbautomatische Zusammenführung erheblich.

7.2 Der DataRepositoryContainer (DRC)

Neben der Personen-Ontologie, die die Semantik der Benutzerdaten abbildet, werden Komponenten benötigt, die diese Semantik explizit nutzen, um damit verteilte Benutzerdaten zu integrieren. Diese Komponenten lassen sich unter dem Begriff „Synchronisations-Architektur“ zusammenfassen (vgl. Kapitel 3.5).

7.2.1 Hintergrund

Verzeichnisse sind passive Datenhaltungskomponenten, die sich üblicherweise nicht selbst um Informationsverteilung im Falle von Datenänderungen kümmern. Synchronisation von Benutzerdaten über verschiedene Verzeichnisse hinweg setzt voraus, dass Änderungen in Datenquellen effizient erkannt und verteilt werden können [BW+02]. Es gibt verschiedene Ansätze zur Lösung, unter anderem das *Retro Change Log* (RCL) des Herstellers SUN [SUN-RCL], das einen zusätzlichen Unterbaum im LDAP-Verzeichnis einrichtet. Jede Änderung führt zu einer Spiegelung der Änderung im RCL-Unterbaum; die geänderten Objekte werden dorthin dupliziert. Dies führt dazu, dass zur Erkennung einer Änderung nicht die gesamte LDAP-Hierarchie durchsucht werden muss. Eine Synchronisationssoftware prüft diesen Unterbaum, liest die geänderten Objekte aus und löscht diese dann an der ursprünglichen Stelle. Dieser Ansatz ist zwar durch die direkte Abbildung auf den LDAP-Standard sehr interoperabel einsetzbar, er hat jedoch auch mehrere Schwachpunkte. Unter anderem muss der RCL-Unterbaum des LDAP-Verzeichnisses regelmäßig abgefragt werden (engl. *Polling*), um über die Änderungen informiert zu werden. Sinnvoller wäre eine Nachrichtenverteilung durch das Verzeichnis selbst im Falle einer Änderung (engl. *Push*). Damit würde sich das Verzeichnis aus seiner passiven Rolle in eine aktive Rolle bewegen. Probleme entstehen auch, wenn eine Synchronisation mit mehreren anderen Verzeichnissen durchgeführt werden soll. Hierbei ist unklar, wer zu welchem Zeitpunkt über die Änderungen informiert ist und wann das Änderungsobjekt final gelöscht werden kann. Eine andere Herangehensweise wird in [AF+99] vorgestellt. Hier wird das Konzept der *Trigger* aus dem Bereich der relationalen Datenbanken auf ein Verzeichnis ausgeprägt. Eine zusätzliche Komponente wird der LDAP-Dienstschnittstelle vorgeschaltet, die alle einge-

henden Anfragen überwacht. Dabei können Anfragen Aktionen auslösen. Während dies zwar ein Ansatz ist, um aus einem passiven Verzeichnis ein aktives zu machen, bleibt die explizite Darstellung der Semantik der ausgetauschten Daten außen vor. Die Abbildungsregeln und Interaktionsbeziehungen sind weiterhin „hart“ in der Erweiterungskomponente codiert.

7.2.2 Funktionale Anforderungen und Umsetzungskonzept

Anforderung A5.1 beschreibt die Notwendigkeit der expliziten Definition der Semantik für die Benutzerdaten-Synchronisation. Um dieses Konzept mit einer Synchronisationskomponente zu verbinden, sollte das (passive) Benutzerverzeichnis um zusätzliche Komponenten erweitert werden, die sich um die Abbildung der Semantik sowie die Ereignisverteilung (engl. *Event Distribution*) im Sinne der Benutzerdatenänderungen kümmert. Hierzu wird in dieser Arbeit mit dem `DataRepositoryContainer` (DRC) ein Konzept eingeführt, das das Basiselement darstellt, um alle Arten von Benutzerdatenquellen semantisch zu integrieren. Grundsätzlich ist ein DRC jedoch als Erweiterung zu sehen, die das bisherige Verhalten eines Anwendungssystems nicht beeinträchtigt. Zusätzlich wird noch eine Synchronisationskomponente benötigt, die ebenfalls wie das DRC die Personen-Ontologie nutzt und die verschiedenen DRCs miteinander verbindet.

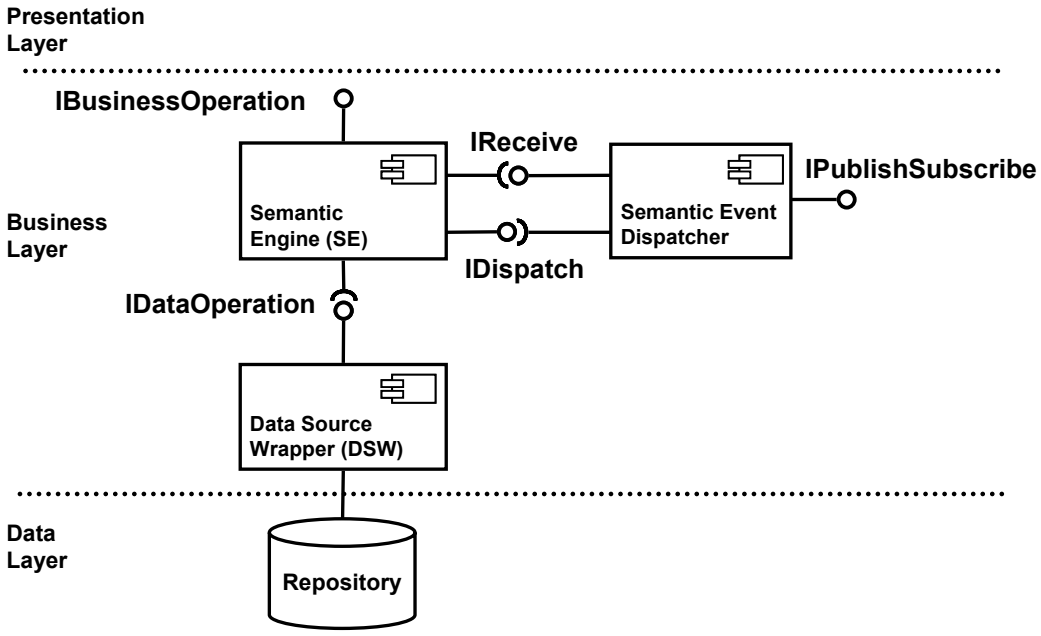


Abbildung 62: Die Architektur des DataRepositoryContainer

Der interne Aufbau eines DRC wird in Abbildung 62 dargestellt. Hier wird eine Drei-Schichten-Architektur gezeigt, wobei die Benutzerdatenquelle auf der Ebene der Datenhaltung (engl. *Data Layer*) einsortiert ist. Es findet bewusst keine Einschränkung auf Verzeichnisse (engl. *Directories*) statt, sondern das Ziel ist, das Konzept des DRC auf alle Arten von Benutzerdatenquellen (engl. *Repositories*) ausprägen zu können. Dies soll den Einsatz in den verschiedensten Anwendungssystemen ermöglichen. Im Sinne einer Trennung der Zuständigkeiten (engl. *Separation of Concerns*) werden die drei angesprochenen Aufgaben auf getrennte Komponenten aufgeteilt, die entsprechend des Grundgedankens der dienstorientierten Architektur ihre Funktionalität gegenseitig an wohldefinierten Dienstschnittstellen zur Verfügung stellen.

Der `DataSourceWrapper` (DSW) dient zur Abstraktion der unterschiedlichen Datenhaltungstypen wie Verzeichnissen, relationalen Datenbanken, XML-Dateien oder Sonstigem. Ziel ist es, einen einheitlichen Dienstzugangspunkt auf eine Benutzerdatenquelle anzubieten. Dies wird durch die Schnittstelle `IDataOperation` des `DataSourceWrapper` erreicht. Die Einordnung der Semantik der Benutzerdaten der lokalen Benutzerdatenhaltung in die gemeinsam genutzte Personen-Ontologie wird durch die `SemanticEngine` (SE) durchgeführt. Sie ist mit mehreren Schnittstellen ausgestattet. `IBusinessOperation` stellt die mit Semantik unterstützte Schnittstelle des DRC dar. Neben der Verwendung eines einheitlichen Protokolls stehen hier die Begrifflichkeiten der Personen-Ontologie im Vordergrund. Des Weiteren verfügt die SE über zwei Synchronisations-Schnittstellen. `IReceive` empfängt eingehende *Events*, während `IDispatch` dafür geeignet ist, selbst *Events* auszulösen. Der `SemanticEventDispatcher` (SED) stellt dabei die Komponente dar, die sich um die Verbindung mit der extern liegenden Synchronisationsplattform kümmert mit der sie über die Schnittstelle `IPublishSubscribe` verbunden ist. Dabei empfiehlt sich ein nachrichtenbasierter Informationsaustausch auf Basis des *Publish/Subscribe*-Musters [GH+95]. Dabei bietet es sich an, eine Synchronisations-Architektur einzusetzen, die als Art *Middleware* die einzelnen `DataRepositoryContainer` verbindet. Nachfolgend werden die einzelnen Komponenten und ihr Zusammenspiel vorgestellt.

7.2.3 Der DataSourceWrapper (DSW)

Der `DataSourceWrapper` (DSW) stellt einen Konverter zwischen dem Protokoll und der Technologie der eingesetzten Benutzerdatenquelle (beispielsweise LDAP) und dem DRC dar. Der DSW implementiert das Entwurfsmuster „Adapter“ (engl. *Adapter*, *Wrapper*) aus [GH+95]. Der Grundgedanke eines Adapters ist es, eine proprietäre Schnittstelle auf eine gemeinsame Schnittstelle „anzuheben“, was das Interoperabilitätsproblem zwischen ansonsten inkompatiblen Schnittstellen löst. Der Einsatz des DSW hat das Ziel, eine größtmögliche Wiederverwendung der beiden weiteren Komponenten SE und SED zu erreichen. Der DSW stellt den Zugang auf die eigentliche Datenhaltungsquelle der SE zur Verfügung. Dies bildet den Ausgangspunkt für eine syntaktische Integration.

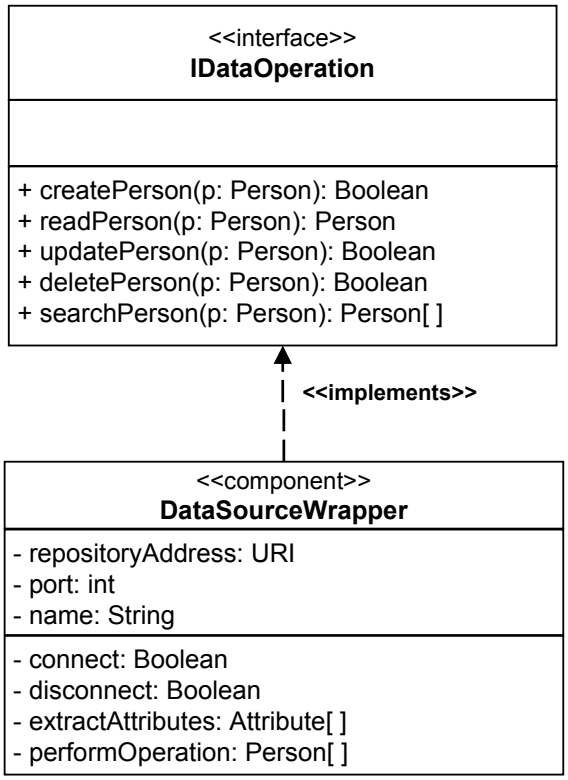


Abbildung 63: Klassendiagramm des DataSourceWrapper

Die Schnittstelle `IDataOperation` nutzt das CRUDS-Muster (*Create, Read, Update, Delete, Search*) als Metaoperationen [Fe04]. Zusätzlich bietet sie die Möglichkeit, gezielt einzelne Attribute eines Benutzers abzufragen. Wie in Abbildung 63 dargestellt, implementiert die `DataSourceWrapper`-Komponente die Schnittstelle und fügt noch weitere private Attribute und Operationen hinzu, die die Anbindung an die individuelle Benutzerdatenquelle

vornehmen. Dabei wird durch die Attribute `repositoryAddress`, `port` und `name` die Quelle eindeutig referenziert, während die Operationen `connect` und `disconnect` die Verbindung herstellen beziehungsweise schließen. Die Operation `extractAttributes` soll einen Zugang zum Datenschema der Quelle ermöglichen, während `performOperation` als Platzhalter für eine proprietäre Operation der Datenquelle steht.

7.2.4 Die SemanticEngine (SE)

Die `SemanticEngine` (SE) führt die semantiktreue Abbildung der lokalen Attribute auf die Personen-Ontologie durch. Dies gilt sowohl für Zugriffe über die Synchronisations-Architektur über die Komponente `SemanticEvent-Dispatcher`, als auch beim direkten Zugriff über die `IBusinessOperation`-Schnittstelle. Die SE setzt damit das Entwurfsmuster „Stellvertreter“ (engl. *Proxy*) um [GH+95]. Obwohl die Datenquelle zu Zwecken der Kompatibilität wie bisher auch direkt genutzt werden kann, ist das Ziel des DRC, jegliche Kommunikation mit der Datenquelle über sich zu lenken. Das erlaubt der SE, bei Änderungsoperationen ein Event über die Schnittelle `IDispatch` an den SED auszulösen.

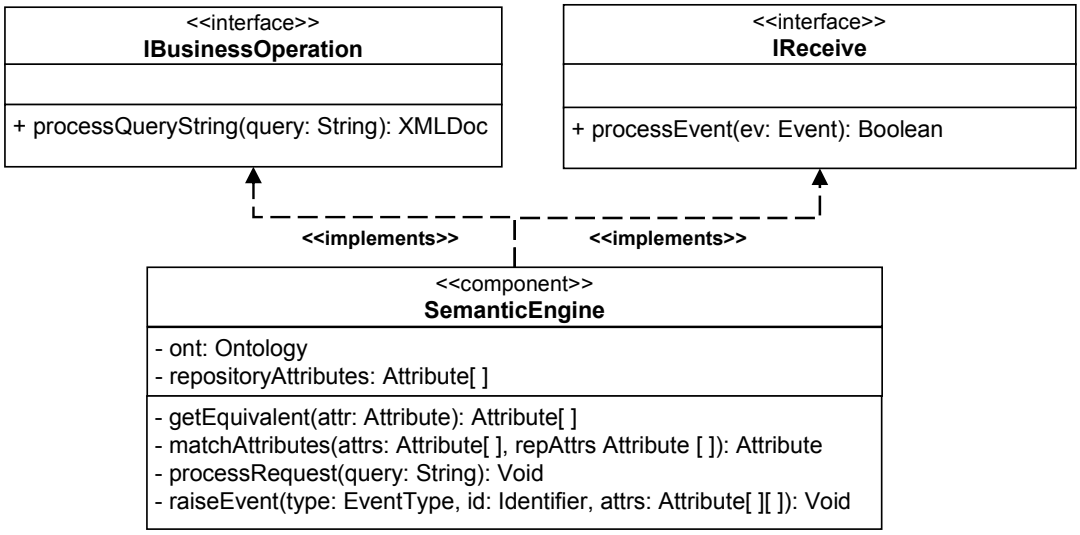


Abbildung 64: Klassendiagramm der SemanticEngine

Die SE implementiert zwei Schnittstellen: einerseits die `IBusinessOperation`, die als Außenschnittstelle des DRC dient, andererseits die Schnittstelle `IReceive`, die vom SED über Änderungen informiert wird. Die SE-

Komponente nutzt noch eine Menge an privaten Attributen und Operationen. Einerseits nutzt sie die durch das Attribut `ont` definierte Personen-Ontologie. In dem Attribut `repositoryAttributes` werden die Attribute der lokalen Datenquelle abgelegt. Die Operation `getEquivalent` gibt semantisch äquivalente Attributsnamen zurück, die dann durch die Operation `matchAttributes` mit Hilfe der Ontologie auf die semantisch äquivalenten Attribute der lokalen Datenquelle abgebildet werden. Dies erlaubt es, eingehende Anfrage auf eine lokal gültige Attributmenge abzubilden, welche dann von der internen Operation `processRequest` an den DSW weitergegeben wird. Im Fall von Änderungen wird zusätzlich noch durch die Operation `raiseEvent` ein Ereignis zur Übergabe an den SED ausgelöst.

7.2.5 Der SemanticEventDispatcher (SED)

Aufgabe des `SemanticEventDispatchers` (SED) ist es, aus der passiven Datenquelle eine aktive Komponente zu machen. Sie teilt ihrer Umwelt mit, wenn Änderungen bei ihr auftreten. Um ein *Separation of Concerns* auch hier umzusetzen, wird diese Außenkommunikation nicht durch die `SemanticEngine` selbst durchgeführt. Sie nimmt *Events* durch die Synchronisationsinfrastruktur entgegen und sendet aktiv auch solche dort hin. Die SED stellt implementiert damit das Entwurfsmuster „Fassade“ [GH+95].

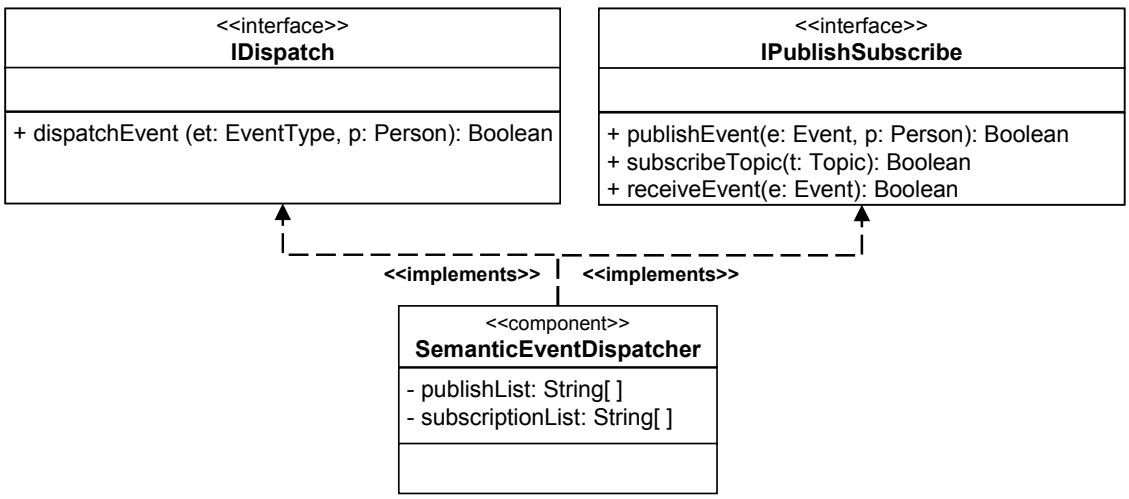


Abbildung 65: Klassendiagramm des SemanticEventDispatcher

Zur besseren Unterscheidung werden *Events* sogenannten *Topics* (dt. Oberbegriff, Thema) zugeordnet. Ein Muster für das Namensschema dieser Topics ist

<hostname>.<eventtype>, also beispielsweise coredirectory.create. Dies erlaubt es einem DRC, sich nur für bestimmte *Events* zu registrieren. Eine Registrierung ohne Filter wird durch *.* abgebildet. DRCs, die sich nur für *Update-Events* interessieren, können sich bei der Synchronisations-Architektur für *.update registrieren. Der SED implementiert zwei Schnittstellen. IDispatch ist die Schnittstelle zur SemanticEngine und nimmt von dort *Events* zur Verteilung entgegen. Ein Event besteht dabei aus einem EventType, der der CRUDS-Operation entspricht, sowie dem entsprechenden Personen-Objekt. Die Operation publishEvent der Schnittstelle IPublishSubscribe gibt das von der SE empfangene Event an die Synchronisations-Architektur weiter. Die Operation subscribeTopic dient zur Anmeldung an der Synchronisations-Architektur und die Operation receiveEvent dient zum Empfang asynchron eintreffender *Events* von dort. Von den zwei privaten Attributen trägt publishList die Information, welche Eventtypen nach außen gegeben werden sollen, während subscriptionList die registrierten Topics speichert.

7.2.6 Zusammenspiel der Komponenten

Neben den Komponenten selbst ist ihr Zusammenspiel von Bedeutung. Abbildung 66 zeigt die Zusammenarbeit in Form eines Kollaborationsdiagramms am Beispiel eines empfangenen Löschauftrags. Wichtig ist, dass auch die erfolgreiche Löschung wiederum ein Event auslösen kann (dargestellt in den Schritten 6 und 7). Für eine ausführlichere Darstellung sowie weitere Interaktionsbeschreibungen sei auf [EL+07] sowie [La06] verwiesen.

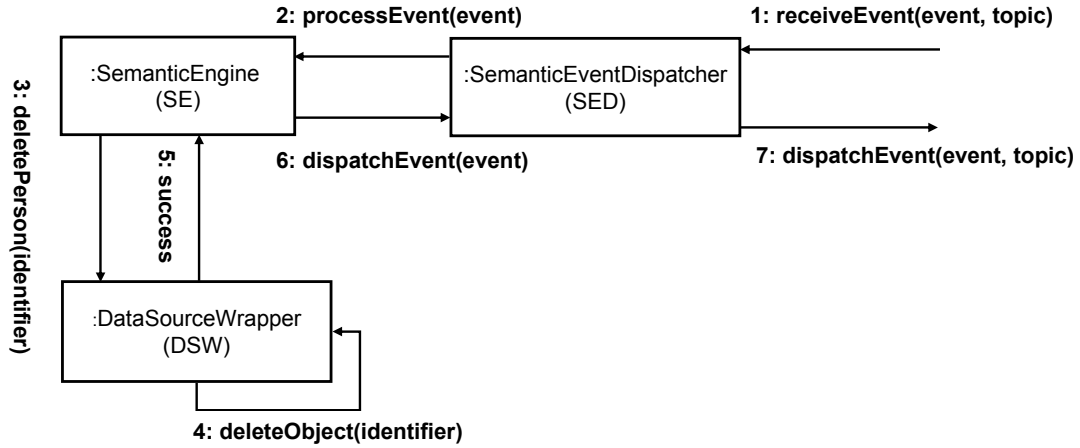


Abbildung 66: Durchführung der Event-Bearbeitung

7.3 Die Synchronisations-Architektur

Damit die verschiedenen DRCs zusammenarbeiten können, bedarf es einer Synchronisations-Architektur. Diese soll als Gegenstelle der `IPublish-Subscribe`-Schnittstellen der einzelnen DRCs fungieren.

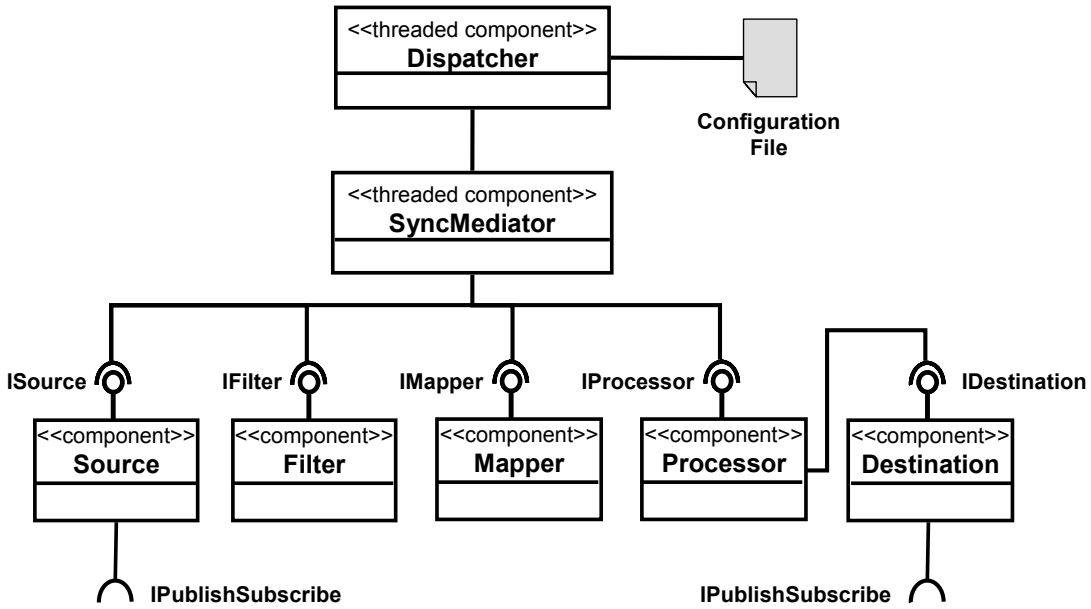


Abbildung 67: Der interne Aufbau des Synchronisations-Architektur

Abbildung 67 zeigt auf hoher Abstraktionsebene den internen Aufbau der Synchronisations-Architektur. Im Kern setzt die Komponente `SyncMediator` das Entwurfsmuster „Mediator“ [GH+95] um. Er wird parametrisiert durch die `Dispatcher`-Komponente, die unter anderem auch die Registrierungen der DRCs verwaltet. Der `SyncMediator` steuert eine Prozesskette. Dabei wird als logisches Eingangselement die `Source`-Komponente genutzt. Sie empfängt *Events* von angeschlossenen DRCs und gibt sie in Form eines serialisierten Transportobjekts an den `SyncMediator` weiter. In den verschiedenen Schritten können durch entsprechende Komponenten Filterungen und Werte-Abbildungen vorgenommen werden, bis im abschließenden Schritt durch den `Processor` eine Verteilung auf die Zielsysteme durchgeführt wird. Die eigentliche Versendung wird durch die `Destination`-Komponente durchgeführt, die sich um die Verteilung an die registrierten DRCs über deren Schnittstelle `IPublishSubscribe` kümmert.

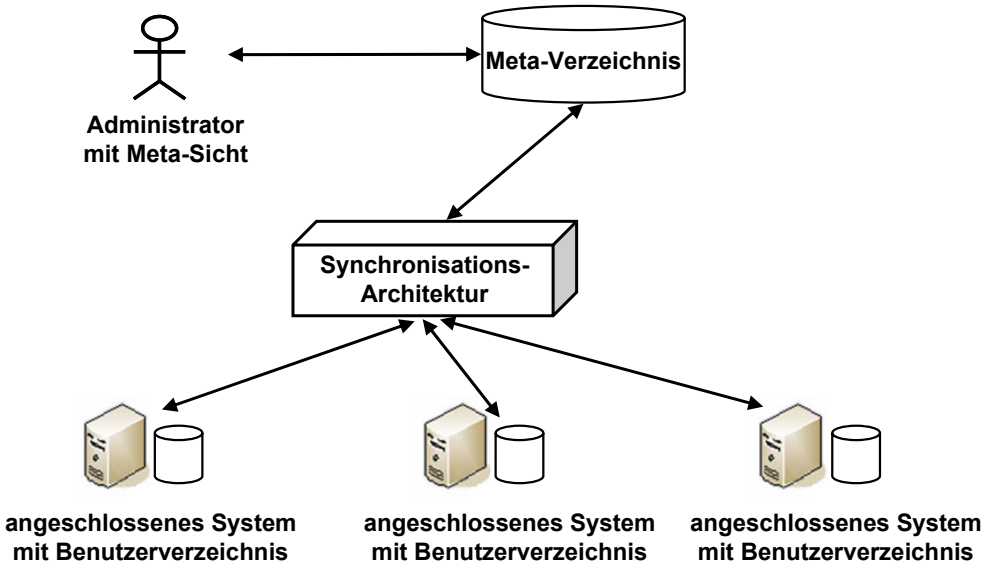


Abbildung 68: Meta-Sicht und Meta-Verzeichnis

Die vorgestellte Architektur ermöglicht einerseits eine direkte Synchronisation gleichberechtigter Benutzerdatenquellen. Durch den Einsatz eines Meta-Verzeichnisses, das an die Synchronisation angekoppelt wird, besteht die Möglichkeit, eine unmittelbare Meta-Sicht auf die gesamten Benutzerdaten zu erzielen. Dadurch können verteilt vorliegende Daten einer digitalen Identität zusammengeführt werden, was zur Administrations und insbesondere Auditierung zweckdienlich ist.

7.4 Prototypische Implementierung

Die vorgestellte Lösung wurde für einen großen Automobilhersteller entwickelt und prototypisch implementiert. Der Software-Entwurf wurde dabei weitestgehend technologieunabhängig durchgeführt, zur Implementierung wurde JEE-Technologie genutzt. Um eine bestmögliche Interoperabilität mit der aufkommenden Webservice-Architektur zu gewährleisten, wurden die Außenschnittstellen der neu geschaffenen *Container* als Webservices bereitgestellt. Dies betrifft insbesondere die Schnittstelle `IBusinessOperation`. Die Kommunikation zwischen den DRCs und der Synchronisations-Architektur findet über SOAP statt. Es fand bewusst keine Festlegung auf einen konkreten *Application Server* statt, auch wenn überwiegend IBM WebSphere Application Server eingesetzt wurden. Um eine Portabilität der einzelnen *Container*-Elemente zu gewährleisten, wurden sie als *Enterprise Java Beans* (EJB) implementiert. Dabei wurde die `SemanticEngine` (SE) als *Stateless Session Bean* umgesetzt, da sie auch

Dienst-implementierende Komponente für die Außenschnittstelle ist. Der `SemanticEventDispatcher` (SED) wurde als *Message-Driven Bean* umgesetzt, die sich ideal mit den Konzepten der Java Message Service-API (JMS) integriert. Der `DataSourceWrapper` (DSW) wurde als *Entity Bean* realisiert. Zur Einbindung der Personen-Ontologie wurde die JENA-API [HP-JENA] eingesetzt. Sämtliche Attribut-Abbildungen werden dynamisch anhand der Ontologie durchgeführt. Daher müssen beim Hinzufügen von neuen Benutzerdatenquellen keine Anpassungen an den bestehenden EJBs vorgenommen werden, die semantischen Überdeckungen werden lediglich in der Ontologie eingetragen. Die Personen-Ontologie besteht aus neun Klassen, 71 Datentype- und Objekt-Eigenschaften (engl. *Datatype* and *Object Properties*). Die OWL-Datei besteht aus circa 800 Zeilen und hat eine Größe von 40 Kilobytes. Die Personen-Ontologie wurde mit dem Werkzeug „Protégé“ [SMI05] entwickelt. Zur Verifikation der Konformität zu OWL-DL (vgl. Kapitel 2.4) wurde der OWL-Validator „WonderWeb“ [BV03] genutzt. Die Konsistenz der Ontologie wurde durch den *Reasoner* „Racer DIG“ (Description Logic Implementation Group) [RACER].

Im Projektkontext wurde zwischen *Online*- und *Offline*-Synchronisation, auch *Batch-Mode* genannt, unterschieden. Die vorgestellte Lösung spielt ihre Stärken in der *Online*-Synchronisation aus, bei der Änderungen unmittelbar propagiert werden. Nichtsdestotrotz lässt sich über die Synchronisation-Komponente eine Sammlung der Auslieferungs-*Events* für ein konkretes Ziel konfigurieren. Diese können dann gesammelt ausgeliefert werden. Neben den durch die verschiedenen Anwendungssysteme ins Spiel kommenden Benutzerdatenquellen wurde ein zentrales Meta-Verzeichnis (vgl. Kapitel 3.5.1.1) eingesetzt. Auch dieses Verzeichnis wurde als DRC bereitgestellt und in die Synchronisation eingebunden. Dabei stellt das Meta-Verzeichnis über seine Außenschnittstelle ebenso wie alle anderen DRCs einen Zugangspunkt dar.

7.5 Resümee

In diesem Kapitel wurde eine Lösung zur semantischen Integration von verteilt vorliegenden Benutzerdaten vorgestellt. Aus Sicht der Zugriffskontroll-Architektur wird lediglich davon ausgegangen, dass auf die digitalen Identitäten über eine definierte Dienstschnittstelle zugegriffen werden kann. Ob sich dahinterliegend ein Meta-Verzeichnis, ein virtuelles Verzeichnis oder eine sonstige Lösung „versteckt“, ist aus Sicht der Zugriffskontroll-Architektur unerheblich.

Der vorliegende Ansatz der semantischen Benutzerdatenintegration stellt einen Synchronisationsmechanismus vor, der in Kombination mit einem Meta-Verzeichnis genau dies liefert: Es wird eine einheitliche Sicht auf die vorliegenden Benutzerdaten erzielt. Dazu wurde zunächst eine erweiterbare Personen-Ontologie erstellt, die in ihrem Kern die Abbildungsregeln der einzelnen Datenquellen auf ein gemeinsames Modell darstellt (vgl. Anforderung A5.1). Zusätzlich wurde mit dem `DataRepositoryContainer` eine Erweiterung von passiven Benutzerdatenquellen vorgenommen, das einerseits die gemeinsam genutzte Personen-Ontologie nutzt und andererseits proaktiv seine Umwelt über Datenänderungen informiert und seinerseits Änderungsereignisse entgegennimmt und verarbeitet. Die verschiedenen DRCs wurden mittels einer Synchronisations-Architektur miteinander verbunden, die nach dem *Publish/Subscribe*-Prinzip arbeitet. Dadurch wird eine Punkt-zu-Punkt-Integration vermieden und stattdessen eine Art Bussystem zur Nachrichtenverteilung eingesetzt.

Die prototypische Implementierung dieser Lösung wurde im Rahmen eines Projekts bei einem großen Automobilhersteller durchgeführt. Dabei zeigte sich, dass durch die Auflösung der Punkt-zu-Punkt-Synchronisationen und durch die Abbildung der einzelnen Datenschemata in eine gemeinsame Ontologie Vorteile entstehen. Einerseits können viele der nur nachts durchgeführten Synchronisationen durch die gemeinsame Synchronisations-Architektur nun in Echtzeit durchgeführt werden, was durch das *Publish/Subscribe*-Prinzip begünstigt wird, andererseits wird das Ankoppeln neuer Datenquellen durch die gemeinsam genutzte Ontologie deutlich vereinfacht. Die dadurch entstehenden Einbußen hinsichtlich der Performanz wurden dadurch aufgewogen.

8 Demonstration der Tragfähigkeit

Um die Tragfähigkeit der in dieser Arbeit entwickelten Konzepte zur Zugriffskontrolle in dienstorientierten Architekturen zu demonstrieren, wird im Folgenden die Umsetzung exemplarisch aufgezeigt. Dazu wird eine Referenzarchitektur für Zugriffskontrolle implementiert, die die Spezifikation in Kapitel 4 umsetzt und unmittelbar auf WSACML-Policies arbeitet. Da im Kontext dieser Arbeit formale Modellierung und modellgetriebene Entwicklung relevante Größen darstellen, wurden zur Unterstützung im Software-Entwicklungsprozess Werkzeuge eingesetzt. Hierzu wurde das Produktportfolie von IBM gewählt, da neben ausgereiften Entwicklungsumgebungen auch passende Laufzeitumgebungen bereitgestellt werden. Zur Modellierung und Implementierung der Webservices (Basisdienste) wird der IBM Rational Software Architect (RSA) [IBM-RSA] eingesetzt, der mit der Laufzeitumgebung IBM WebSphere Application Server (WAS) [IBM-WAS] korreliert. Im Bereich der BPEL-basierten Dienstkompositionen wird der IBM WebSphere Integration Developer (WID) [IBM-WID] genutzt, der als Laufzeitumgebung den IBM WebSphere Process Server (WPS) [IBM-WPS] verwendet. Um den Gedanken der dienstorientierten Architektur umzusetzen, wird zunächst ein Prozessausschnitt modelliert und in die dienstorientierte Facharchitektur umgesetzt. Dabei wird nur die Fachfunktionalität abgebildet; Querschnittsfunktionen wie Zugriffskontrolle bleiben zunächst außen vor. Anschließend wird die Zugriffskontroll-Architektur umgesetzt und durch den Einsatz des *Secure Service Agent* mit der Facharchitektur verbunden. Als zu unterstützende Geschäftsdomäne wird die dienstorientierte Rechnerunterstützung im Bereich der Aus- und Weiterbildung an der Universität Karlsruhe herangezogen. Dies wird unter anderem im Projekt „Karlsruher Integriertes Informations-Management“ (KIM) [UKA-KIM] vorangetrieben.

8.1 Beschreibung des Szenarios

8.1.1 Prüfungsmanagement an einer Hochschule

Das Prüfungsmanagement stellt eine Kernaufgabe einer Hochschule dar. Ziel dieses Bereichs ist die dienstorientierte Unterstützung der Durchführung von Prüfungen und der Verwaltung von Prüfungsergebnissen. Im vorgestellten Beispiel wird ein kleiner Ausschnitt dieses Szenarios detaillierter betrachtet, der

sich mit der Erzeugung eines Notenauszugs (engl. *Transcript of Records*, ToR) beschäftigt, der durch den Bologna-Prozess europaweit vereinheitlich werden soll. Zur Erstellung des Notenauszugs muss zunächst ein Ablaufdiagramm erstellt werden, das das Vorgehen verdeutlicht. Dabei wird sichtbar, dass die notwendigen Informationen zur Erstellung in verschiedenen Organisationsbereichen vorgehalten werden, die wiederum unterschiedliche Anwendungssysteme für ihre Arbeitsprozesse nutzen. Der Erstellvorgang stellt daher ein ideales Anschauungsobjekt dar, um zu zeigen, wie technische Integration durch dienstorientierte Architekturen vereinfacht werden kann. Während die Modellierung des Ablaufes in UML-Aktivitätsdiagrammen oder BPMN-Prozessmodellen den Ausgangspunkt darstellt, findet in weiteren Schritten die Umsetzung der rechnerunterstützten Teilschritte in ausführbare BPEL-Artefakte statt. Dies hat der Autor in mehreren Publikationen bereits gezeigt [EM+05, EW+06]. Eine notwendige Voraussetzung ist es dabei, dass die bestehenden Anwendungssysteme mit entsprechenden Dienstschnittstellen versehen wurden, was einen notwendigen, jedoch nicht zu unterschätzenden Initialaufwand darstellt. Da es sich bei den meisten im Bereich des Prüfungsmanagement angebotenen Diensten um personenbezogene Auskunftsdienste handelt, genügt die rein fachliche Umsetzung nicht. Es ist sicherzustellen, dass nur berechtigte Zugriffe auf die Informationen zugelassen werden, was durch die in dieser Arbeit entwickelte Zugriffskontroll-Architektur realisiert wird.

8.1.2 Das Projekt „Karlsruher Integriertes Informations-Management“

Das Projekt Karlsruher Integriertes InformationsManagement (KIM) [UKA-KIM], das sich um die im Szenario aufgegriffene Aufgabe der dienstorientierten Rechnerunterstützung in Forschung und Lehre an der Universität Karlsruhe (TH) kümmert, wird im Folgenden kurz vorgestellt. Der Autor hat dieses Projekt seit seinem Start im Jahre 2005 kontinuierlich begleitet, insbesondere bei der Initiierung des Schwesterprojekts KIM-IDM, das sich um das notwendige Identitätsmanagement als Grundlage für die Einführung von dezentralisiert durchführbaren Zugriffskontroll-Entscheidungen für IT-Dienste. Das KIM-Projekt wurde aufgrund der Bedeutung organisationsübergreifender Geschäftsprozesse der Universität im Bereich der Forschung und Lehre und der zur Unterstützung dieser Geschäftsprozesse aufgrund der Unabhängigkeit und Heterogenität nur bedingt geeigneten Rechnerunterstützung ins Leben gerufen. Die von verschiedenen Organisationseinheiten angebotene Funktionalität ist zwar in den meisten

Fällen ausreichend, aber die Geschäftsprozess-begleitende Integration dieser Funktionalitäten ist aufgrund von technischen Inkompatibilitäten und Medienbrüchen nicht optimal möglich. Diese Probleme erschweren die organisationsübergreifende Zusammenarbeit und können zu Konsistenzproblemen in den verschiedenen beteiligten Systemen und den unterstützten Geschäftsprozessen führen. Im KIM-Projekt sollen diese Probleme durch die Schaffung einer organisatorischen und technischen Infrastruktur zur Bereitstellung einer integrierten dienstorientierten Architektur angegangen werden. Diese Architektur soll die Integration und das Angebot von Diensten verschiedener Systeme und Organisationseinheiten in einem gesamtuniversitären Kontext ermöglichen bzw. vereinfachen. Das Ziel der Erhöhung der Exzellenz in der Lehre soll durch eine bessere Rechnerunterstützung und eine Vereinheitlichung und Vereinfachung der Geschäftsprozesse erreicht werden. Damit soll eine Optimierung der Zusammenarbeit verschiedener Organisationseinheiten (zentrale Verwaltung, Fakultäten, Institute, Lehrstühle…) ermöglicht werden, die zu einer erhöhten Konsistenz der Geschäftsprozesse führen soll.

8.2 Die Facharchitektur als Ausgangspunkt

Notenauszüge spielen eine wichtige Rolle im Prüfungsmanagement an einer Hochschule: Studierende benötigen Notenauszüge für Bewerbungszwecke oder um sich einen Überblick über ihren Studienfortschritt zu verschaffen. Studienberater nutzen Notenauszüge, um im persönlichen Gespräch Hinweise zur Planung des weiteren Studienverlaufes geben zu können. Durch die fortschreitende Umsetzung des Bologna-Prozesses bekommt auch der Notenauszug einen definierten Stellenwert; er soll europaweit vereinheitlicht werden. Hierzu ist Vorbedingung, sich zunächst über die zur Erstellung notwendigen Daten einen Überblick zu verschaffen. Auf dem Notenauszug finden sich Daten zur Hochschule selbst, Stammdaten des Studierenden, Informationen zu den Studienmodulen und zugehörige Prüfungsergebnisse des Studierenden. Bei einer ersten Analyse des Geschäftsbereichs zeigt sich, dass die Daten zur Hochschule durch eine Stabsstelle des Rektorats definiert werden, die Stammdaten des Studierenden durch das Studiensekretariat gepflegt werden, die Prüfungsergebnisse vom Prüfungsamt verwaltet werden und die Informationen zu den Studienmodulen durch das dezentralisierte Lehrveranstaltungsmanagement der Fakultäten aktuell gehalten werden. Alle diese Organisationseinheiten arbeiten weitestgehend autark und mit ihren eigenen Anwendungssystemen. Bevor über die Bereitstellung von Diensten durch diese Systeme nachgedacht werden sollte, sind zunächst

die Abläufe zur Erstellung als zu erreichender Sollzustand festzuhalten. Die im Folgenden verwendeten Modelle nutzen dazu grundsätzlich die englische Sprache. Zur Abstraktion des konkreten Sachverhaltes werden die Oberbegriffe verwendet. So wird aus der Hochschule ein Schulungsunternehmen (engl. *Training Company*), aus dem Studierenden ein Kunde (engl. *Customer*). Der Notenauszug wird auf einen Trainingsergebnis (engl. *Training Status*) abgebildet.

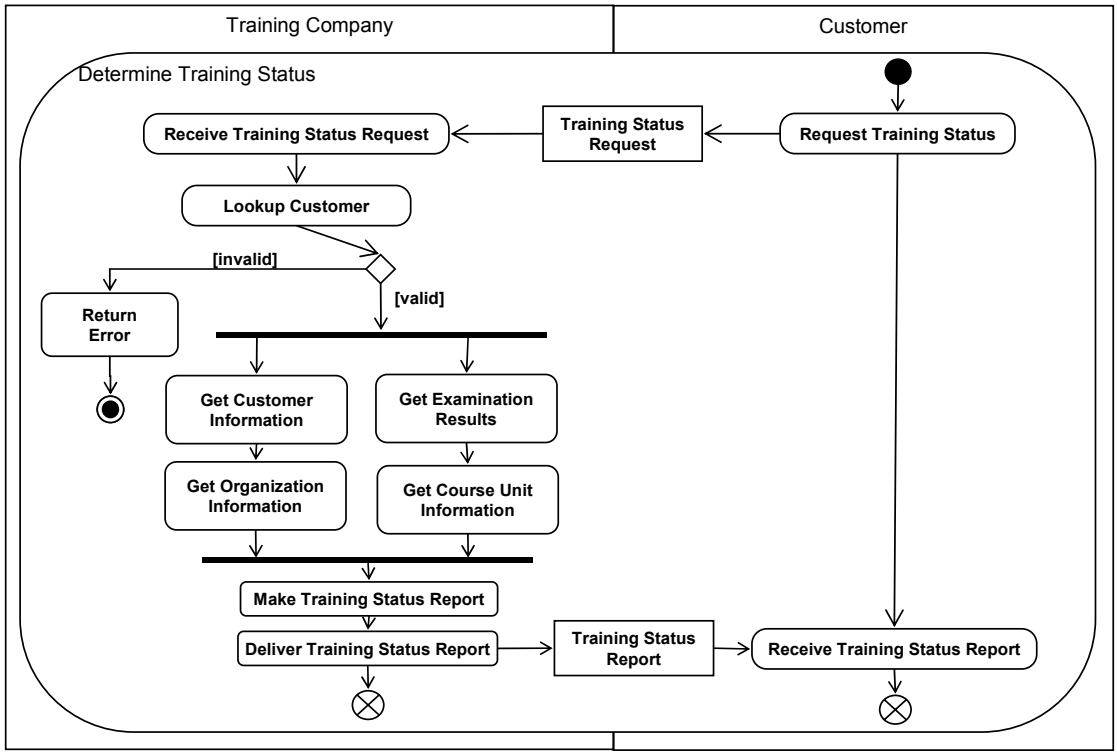


Abbildung 69: Aktivitätsdiagramm "Determine Training Status"

Abbildung 69 zeigt mittels eines UML-Aktivitätsdiagramms den Ablauf der Erstellung eines Trainingsergebnisses. Es finden sich die zwei beteiligten Prozesspartner wieder, der Kunde und das Schulungsunternehmen. Der Kunde stößt durch eine Anfrage die Erstellung an. Nach einer Validierung der Anfrage werden aus vier Bereichen Daten angefordert und diese zu einem Trainingsergebnis zusammengesetzt. Dieses Ergebnis wird dann an den anfragenden Kunden zurückgegeben. Zur Rechnerunterstützung ist dieser Sachverhalt in die Schichten der dienstorientierten Architektur einzuordnen, wie sie in Abbildung 1 vorgestellt wurden. Die nachfolgend vorgestellte Implementierung wurde maßgeblich durch [DM08] realisiert.

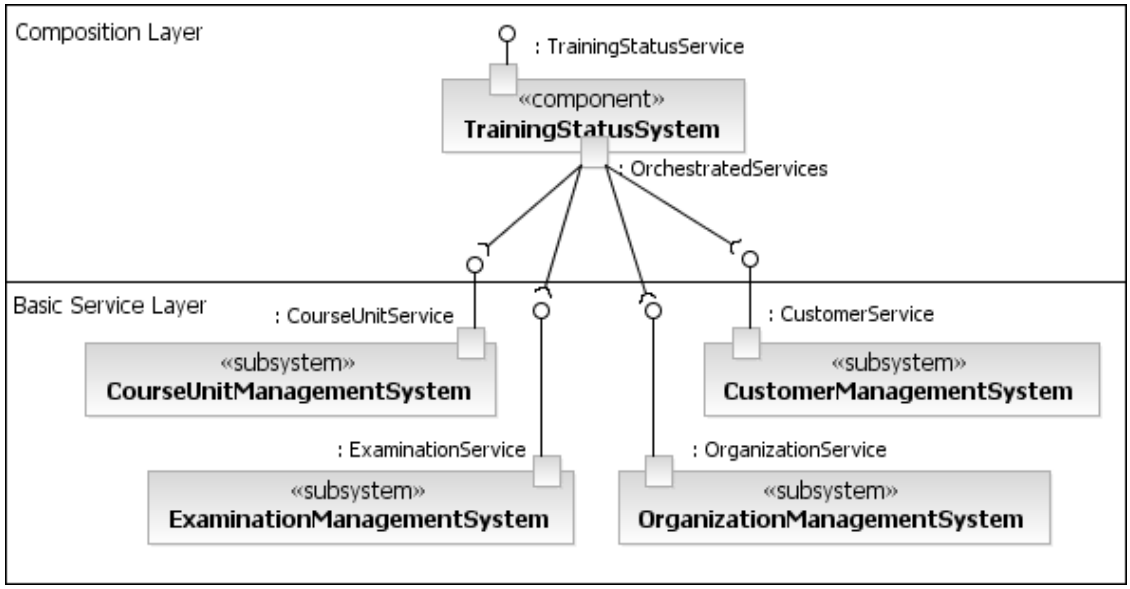


Abbildung 70: Integration bestehender Anwendungen

Da es sich um einen vollständig automatisierbaren Prozess handelt, lässt sich die Trainingsergebnis-Erstellung auf eine Komposition von Basisdiensten abbilden. Dabei werden vier Basisdienste zur Bereitstellung der notwendigen Daten aus den verschiedenen Altsystemen eingesetzt. Trotz der Überschaubarkeit des Prozessablaufes ist es vernünftig, ihn als eigenständigen Prozess zu implementieren, da er in anderen Prozessen wiederverwendet werden kann. Ein Überblick über die beteiligten Altsysteme und ihre Einbettung in die dienstorientierte Architektur wird in Abbildung 70 gegeben.

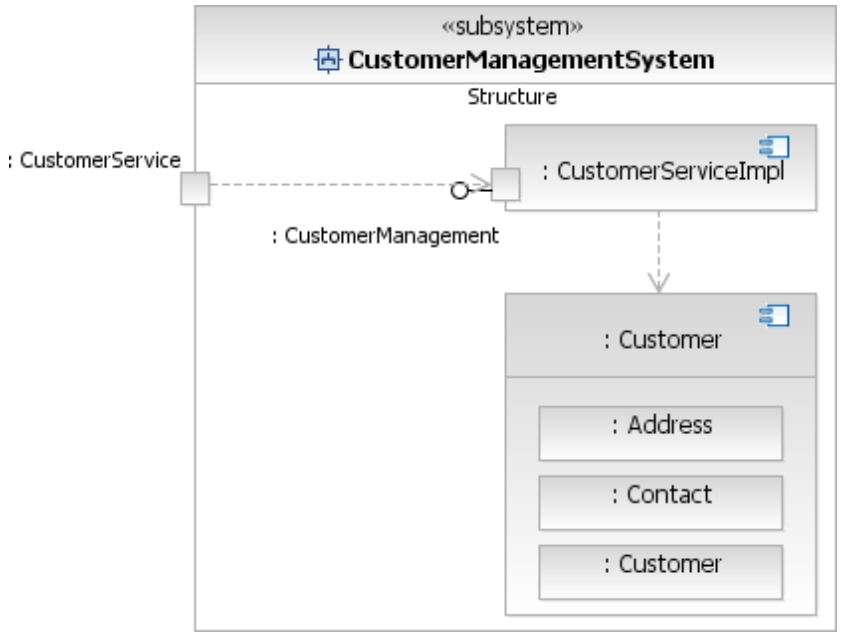


Abbildung 71: Bereitstellung von Basisdiensten

Hauptziel bei der Erstellung der Basisdienste ist ein hohes Maß an Portabilität, da sie über einen längeren Zeitraum verwendet werden können als Dienstkompositionen, die sich entsprechend der Anforderungen der Geschäftsprozesse verändern müssen [KB+05]. Die Dienst-implementierenden Komponenten sollten daher nicht von einer speziellen Laufzeitumgebung abhängig sein. Vor allem im Falle von JEE *Entity Beans* bedeutet dies, dass der Datenzugriff unabhängig vom *Application Server* durchgeführt werden sollte. Abgebildet auf JEE-Technologie bedeutet das den Einsatz sogenannter *Bean-Managed Persistence* (BMP) im Gegensatz zu *Container-Managed Persistence* (CMP). Am Beispiel der Stammdatenverwaltung wird in Abbildung 71 vorgestellt, wie die Bereitstellung von Basisdiensten durch bestehende Anwendungssysteme (Altsysteme) durchgeführt wird. Die Funktionalität wird durch das bestehende `CustomerManagement-System` bereitgestellt, das nach außen die Webservice-Schnittstelle `CustomerService` anbietet. Diese wird durch eine JEE *Stateless Session Bean* implementiert (`CustomerServiceImpl`), die auf dem `Customer`-Objekt operiert.

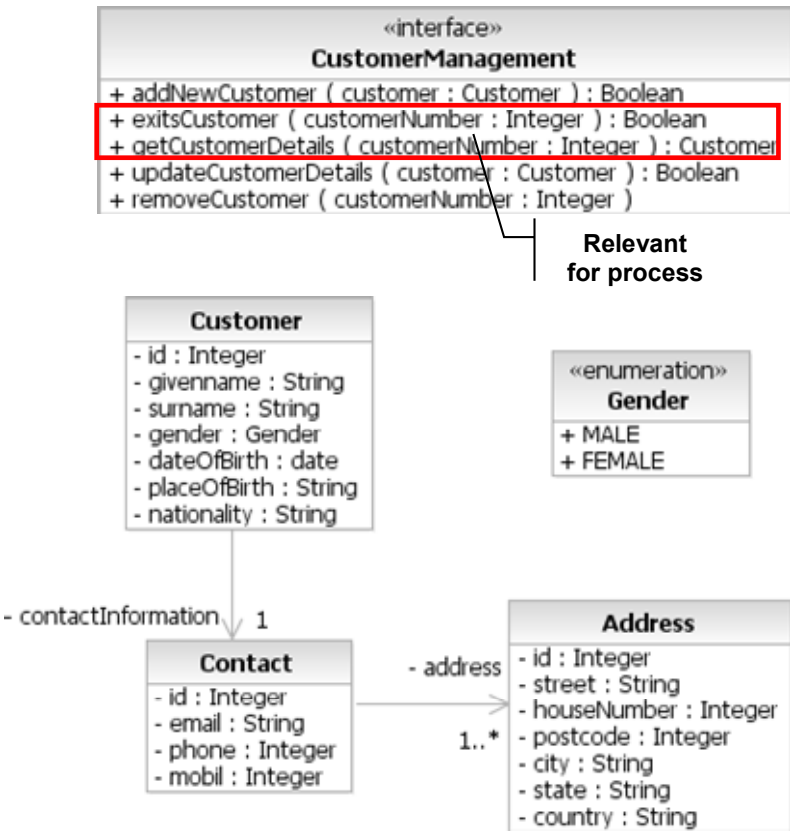


Abbildung 72: Die Schnittstelle des Basisdienstes “CustomerManagement”

Bei der Bereitstellung als Dienst kommt der Schnittstellen-Spezifikation eine wichtige Bedeutung zu. Hier werden die durch den Webservice angebotenen Operationen anhand der ausgetauschten Geschäftsobjekte (Variablen) spezifiziert. Exemplarisch wird in Abbildung 72 die Schnittstelle des Kundenverwaltungs-Diensts (`CustomerManagement`) betrachtetet. Der Webservice stellt fünf Operationen bereit, wobei zwei zur Umsetzung des in Abbildung 69 vorgestellten Prozesses benötigt werden. Diese wurden zur besseren Sichtbarkeit umrandet. Alle angebotenen Operationen beschäftigen sich mit der Verwaltung von Kunden und Kundendaten. Von der Art ihrer Ausprägung handelt es sich um sogenannte CRUDS-Operationen (*Create*, *Read*, *Update*, *Delete, Search*). Da es sich jedoch bei den ausgetauschten Variablen um grobgranulare Geschäftsobjekte handelt, sollte ein Datenmodell erstellt werden, das über alle Dienste hinweg ein einheitliches Verständnis der ausgetauschten Daten ermöglicht. Der Teil, der für den Webservice `CustomerManagement` erforderlich ist, wird im unteren Teil von Abbildung 73 dargestellt.

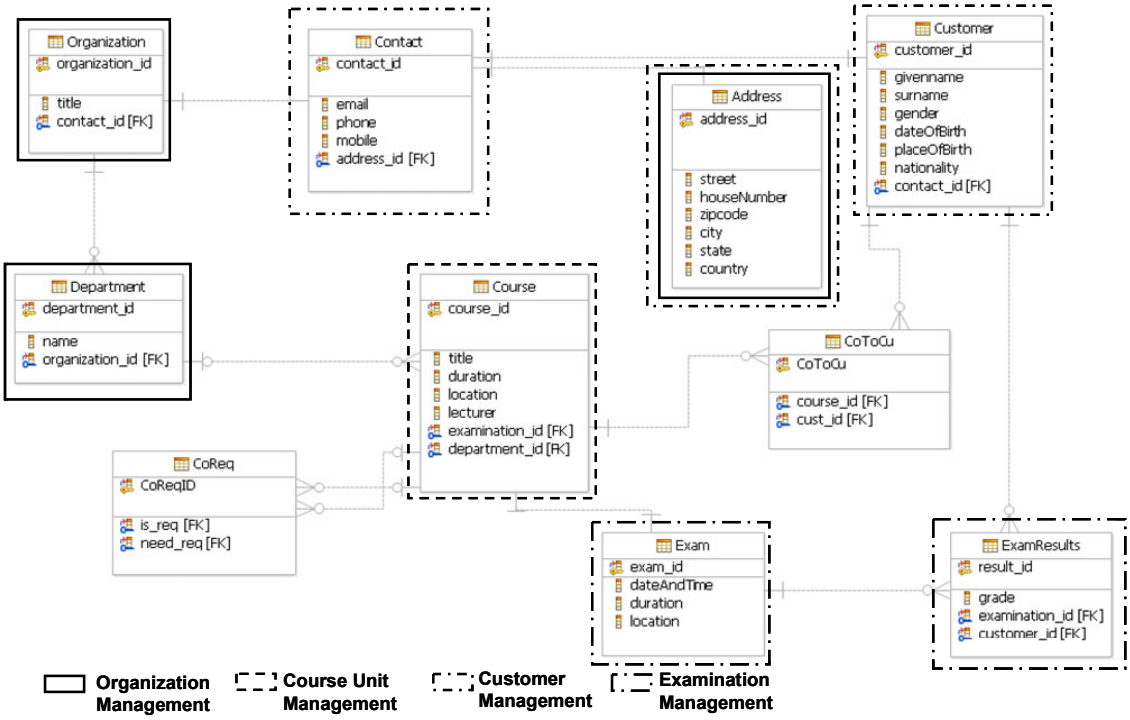


Abbildung 73: Das zugrundeliegende Datenmodell

Schon in dem abgegrenzten Prozess zur Erstellung der Trainingsergebnisse müssen Daten aus vier verschiedenen Altsystemen zusammengeführt werden. Hierbei ergeben sich unmittelbare Abhängigkeiten, die in einem Datenmodell festgehalten werden sollten. Dies ist in Abbildung 73 dargestellt. Neben den

Objekten selbst sind auch die Beziehungen untereinander sowie die Zugehörigkeit zu den einzelnen Basisdiensten dargestellt. Dieses relationale Datenmodell wurde mittels der Datenmodellierungs-Tools des IBM Rational Software Architect (RSA) entwickelt. Die Relationen entsprechen weitestgehend den Geschäftsobjekten und die Attribute der Relationen entsprechen den Daten der Geschäftsobjekte. Hinzukommen lediglich für relationale Datenbanken nötige Artefakte wie beispielsweise Primärschlüssel und Fremdschlüsselbeziehungen. Für n:m-Beziehungen wurden zusätzliche Assoziationstabellen eingefügt, um eine Normalform des Schemas zu gewährleisten.

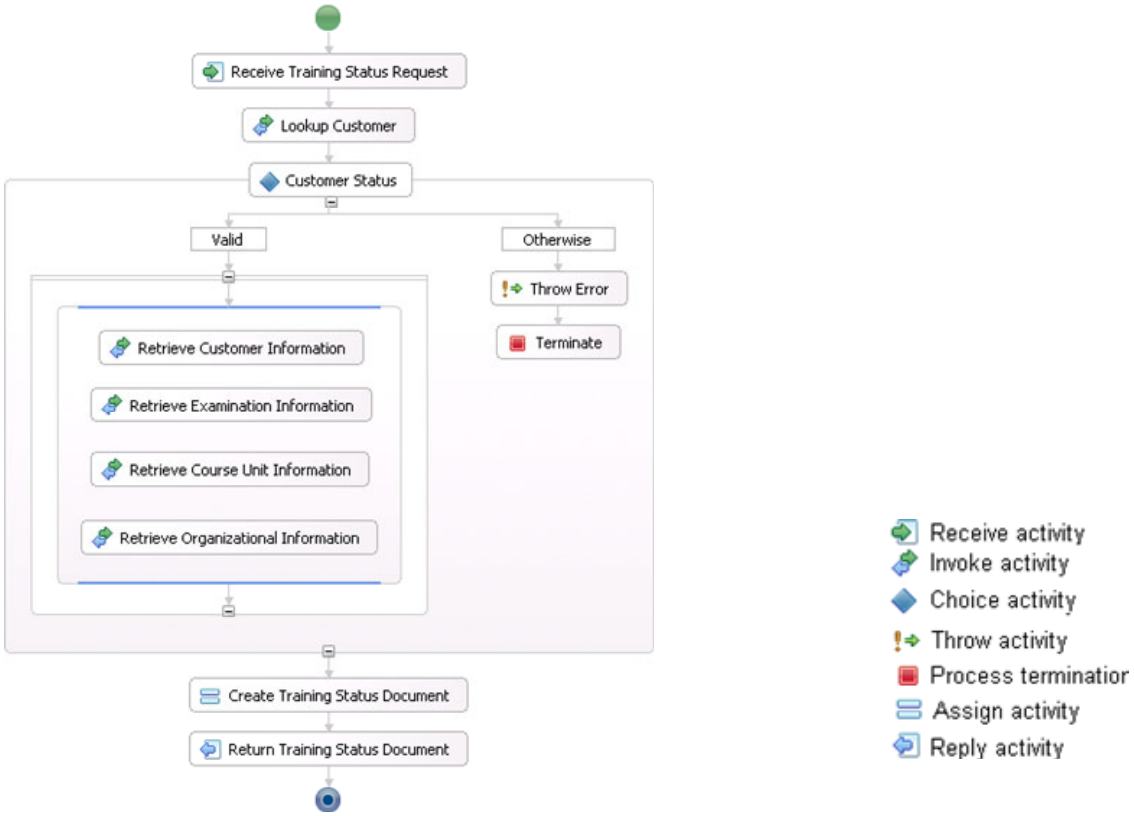


Abbildung 74: Umsetzung als BPEL-Prozess

Während der Prozess zunächst weitestgehend technologieneutral in Form eines UML-Aktivitätsdiagramms dargestellt wurde (vgl. Abbildung 69), gilt es nun, dieses Modell auf eine unmittelbare Prozessausführungssprache abzubilden. Hierbei bietet sich die herstellerunabhängige *Business Process Execution Language* (BPEL) an, die derzeit das Mittel der Wahl darstellt, um eine Abbildung von Geschäftsprozessen auf Webservice-Kompositionen durchzuführen. Der Prozess aus dem zuvor gezeigten UML-Aktivitätsdiagramm wird dazu mit dem IBM WebSphere Integration Developer als BPEL-Prozess umgesetzt. Dazu wird der im WID eingebaute BPEL-Prozesseditor verwendet. Nach dem Eingang der

Anfrage wird durch einen Dienstaufruf ermittelt, ob der anfragenden Kunde existiert. Die dazu nötige Information ist die eindeutige Kundennummer des Kunden. Bei einer erfolgreichen Abfrage werden analog zum Aktivitätsdiagramm die benötigten Informationen aus den verschiedenen Datenquellen zusammengetragen. Auch hier sollte die Kundennummer (Matrikelnummer) ausreichen, um die entsprechenden Informationen aus den Datenquellen zu extrahieren. Danach werden die gesammelten Informationen in einem Geschäftsobjekt zusammengetragen und dem Benutzer bereitgestellt. Falls der Aufruf mit einer ungültigen Kundennummer durchgeführt wird, wird ein Fehler generiert und der Prozess wird beendet.

Mit der Bereitstellung der Basisdienste und der Dienstkomposition sind nun die fachlichen Aspekte umgesetzt. Dies entspricht der Darstellung in Abbildung 1. Neben der Abbildung der Fachdienste ist jedoch auch insbesondere die Zugriffskontrolle hinsichtlich der Dienste sicherzustellen. Dies wird durch die Facharchitektur nicht durchgeführt und in den nachfolgenden Kapiteln dargestellt. Dazu wird in Kapitel 8.3 die Zugriffskontroll-Architektur als Referenzarchitektur implementiert und in Kapitel 8.3.5 die Verknüpfung mit der Facharchitektur durchgeführt.

8.3 Die Zugriffskontroll-Architektur

Abbildung 75 zeigt die um einige Datenzugriffskomponenten angereicherte Zugriffkontroll-Architektur für dienstorientierte Architekturen in einem UML-Komponentendiagramm. Die Darstellung basiert auf der in Abbildung 28 dargestellten konzeptionellen Architektur. Eine Erweiterung der konzeptionellen Architektur wurde mit der `Data Access`-Schicht geschaffen. Sie abstrahiert den Zugriff auf die eigentlichen Datenquellen. Deshalb erfolgt der Zugriff auf benötigte Datenquellen über dedizierte Dienstschnittstellen, die die einzelnen Datenquellen kapseln. Durch diesen zentralen Zugriffspunkt lassen sich nicht nur Zugriffsproblemen vermeiden, auch die Austauschbarkeit der Datenquellen ist damit gewährleistet. Der `PolicyDecisionPoint` bietet Dienste zur Zugriffskontrolle auf Ressourcen an und nimmt somit in der Gesamtarchitektur eine zentrale Rolle an. Er entscheidet, ob ein Dienstnutzer autorisiert ist, eine Ressource zu nutzen. Die Datenquellen, die der PDP zur Entscheidungsfindung benötigt, wurden im Entwurf spezifiziert. Das `TokenRepository` wird verwendet, um Sicherheitstoken auf reale Subjekte abzubilden. Des Weiteren

benötigt er für den aufgerufenen Dienst entsprechende Policies, die in einem `PolicyStore` abgelegt sind.

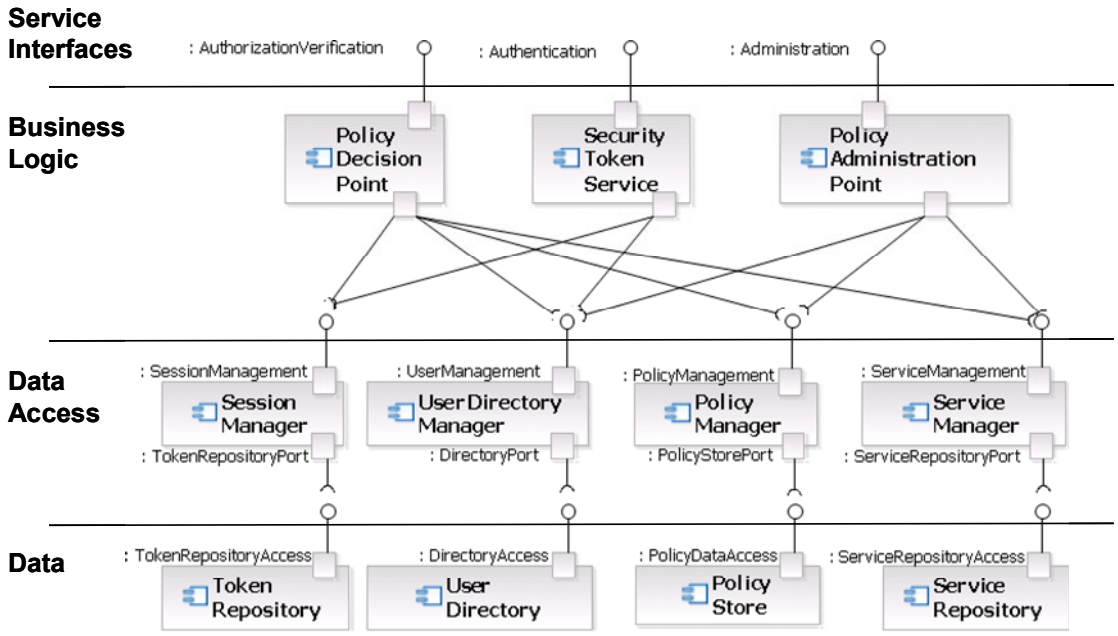


Abbildung 75: Die Zugriffskontroll-Architektur im Modell

Zur Auswertung dieser Policies ist es meist nötig, weitere Eigenschaften und Attribute des Benutzers bzw. des aufgerufenen Dienstes mit zu betrachten. Diese können jeweils in Benutzer- (`UserDirectory`) beziehungsweise Dienstverzeichnissen (`ServiceRepository`) abgerufen werden. Die verschiedenen Daten werden vom PDP über die angebotenen Dienstschnittstellen extrahiert. In den folgenden Unterkapiteln steht der Implementierung des PDP im Mittelpunkt. Zunächst wird die Struktur des PDP weiter verfeinert. Danach werden die beteiligten Datenquellen und die von ihnen angebotenen Dienstschnittstellen betrachtet. Für die eigentliche Autorisierungsprüfung weniger relevanten Komponenten wie etwa der `SecurityTokenService` und `PolicyAdministrationPoint` sowie die `ServiceRegistry` werden im Folgenden nicht weiter beachtet.

8.3.1 Der PolicyDecisionPoint (PDP)

Zur Darstellung des PDP wird zunächst sein interner Aufbau vorgestellt. Daran schließen sich die dynamischen Aspekte sowie seine Schnittstelle an.

8.3.1.1 Der interne Aufbau

In Abbildung 76 wird der interne Aufbau der PDP-Komponente aufgezeigt. Zur Auswertung einer Policy müssen zunächst verschiedene Informationen aus den unterschiedlichen Datenquellen geladen werden. Aus der Anfragenachricht an den PDP lassen sich die Eingabeparameter der aufgerufenen Webservice-Operation herausfiltern. Zudem lassen sich eventuelle Attribute der Laufzeitumgebung extrahieren, welche durch einen *Policy Enforcement Point* (PEP) erstellt wurden. Außerdem wird das Sicherheitstoken übermittelt, anhand dessen die Daten zum aktuellen Sitzungskontext des aufrufenden Benutzers ermittelt werden können. Bei einer gültigen Sitzung werden die für die Webservice-Operation verantwortliche Policy, die Attribute des Benutzers und des Dienstes aus den jeweiligen Datenquellen geladen. Anschließend nutzt die interne *Policy Evaluator*-Komponente die gesammelten Informationen, um die Policy auszuwerten und die Autorisierungsentscheidung zu treffen.

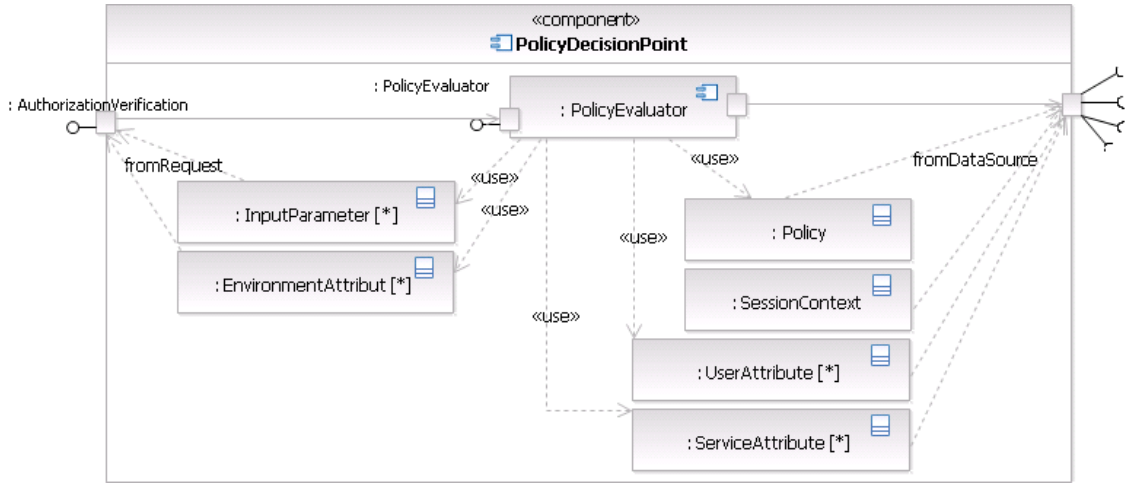


Abbildung 76: Interner Aufbau des PolicyDecisionPoint

Abbildung 77 zeigt den Aufbau der `PolicyEvaluator`-Komponente. Die Hauptaufgabe des `PolicyEvaluator` ist die Auswertung der für eine abgesicherte Ressource zutreffenden Policy. Die Daten, die der Komponente dabei übergeben werden, spiegeln den zum Zeitpunkt des Zugriffs zutreffenden Kontext wider. Die zur Auswertung notwendigen Daten wurden bereits vom PEP bzw. PDP gesammelt und der `PolicyEvaluator`-Komponente zur Verfügung gestellt. Dadurch lassen sich weitere Abhängigkeiten zwischen der Komponenten und der Umwelt vermeiden.

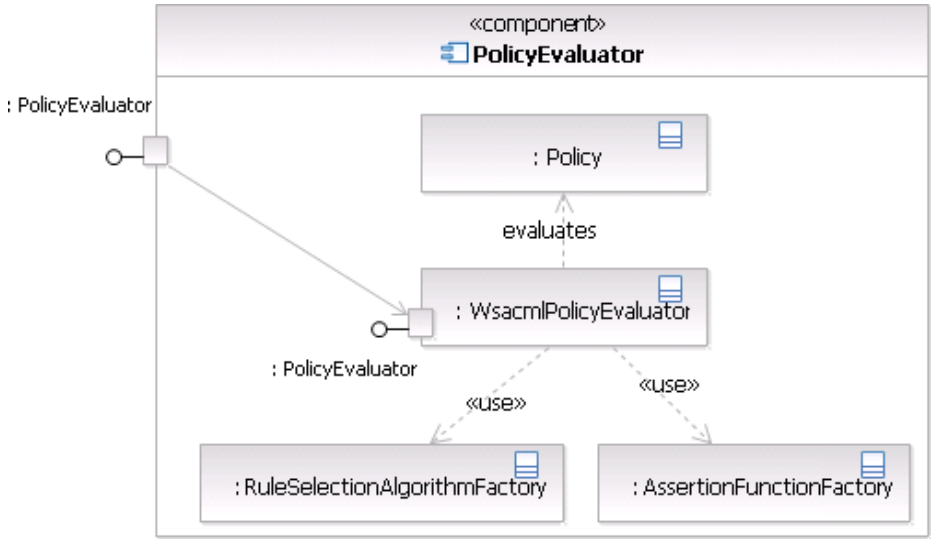


Abbildung 77: Die PolicyEvaluator-Komponente

Abbildung 78 zeigt einen Ausschnitt aus dem Domänenmodell des `PolicyEvaluator` mittels eines UML-Klassendiagramms. Aus der Policy-Spezifikation wird zunächst ein geeigneter Auswahlalgorithmus ausgewählt. Hierzu wird das *Abstract Factory*-Muster [GH+95] angewandt, um den passenden Algorithmus zu erzeugen.

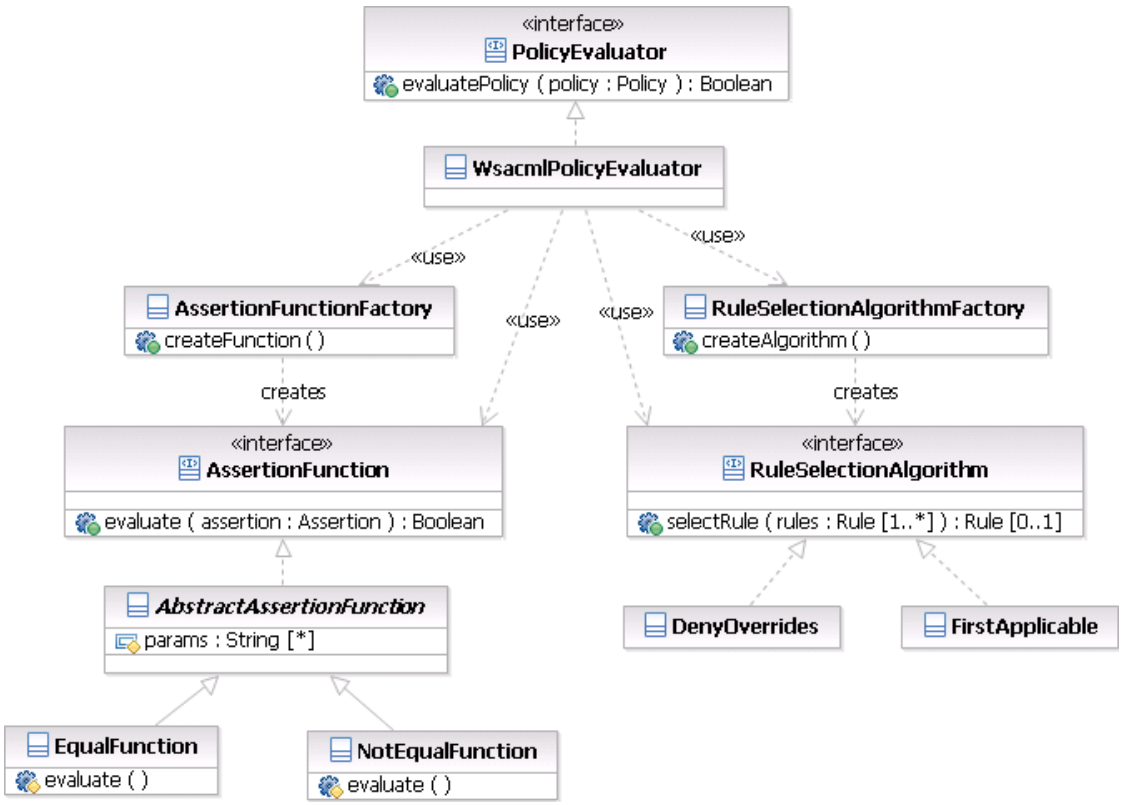


Abbildung 78: Das Domänenmodell des PolicyEvaluator

In der Abbildung werden zwei dieser Algorithmen dargestellt. Der `Deny-Overrides`-Algorithmus wählt aus einer Menge von zutreffenden Regeln diejenige aus, welche einen Zugriff verweigert. Der `First-Applicable`-Algorithmus hingegen wählt die zuerst zutreffende Regel aus, in der Reihenfolge in der sie deklariert sind. Durch das *Abstract Factory*-Muster sind diese Funktionen beliebig erweiterbar. Für eine Zusicherung wird entsprechend ihrer Spezifikation eine geeignete Funktion zur Auswertung der Zusicherung ausgewählt. Da die Funktionen als Klassen implementiert sind, kann auch hier das *Abstract Factory*-Muster eingesetzt werden. Die in der Abbildung dargestellte Funktion `Equals` überprüft, ob die beiden Argumente der Zusicherung gleich sind, während die `NotEquals`-Funktion auf Ungleichheit prüft.

8.3.1.2 Dynamische Aspekte

Neben den statischen Zusammenhängen werden nachfolgend die dynamischen Aspekte mittels UML-Sequenzdiagrammen dargestellt.

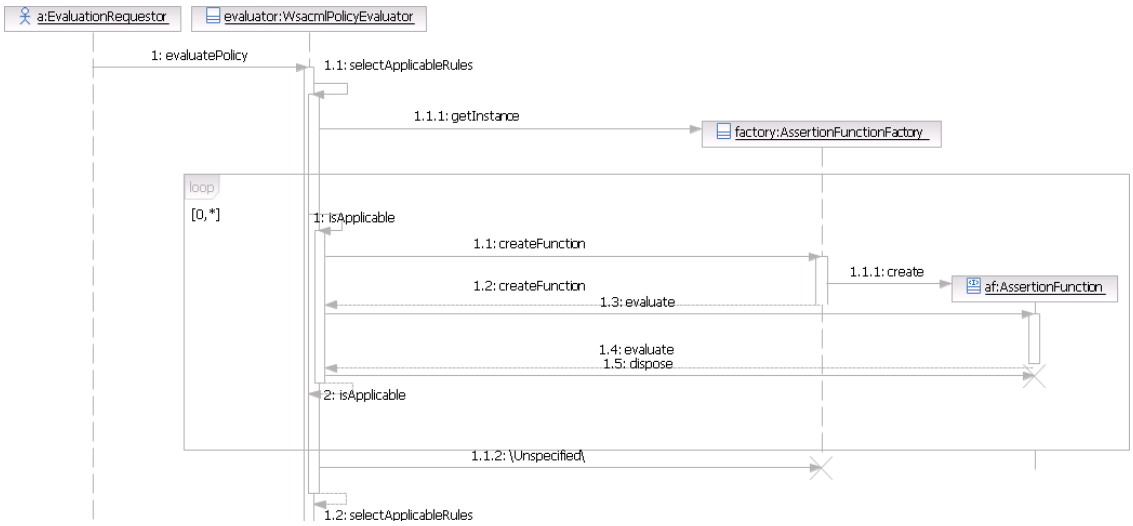


Abbildung 79: Funktionsweise des PolicyEvaluator (1)

Abbildung 79 und Abbildung 80 zeigen die Funktionsweise des `Policy-Evaluator` in einem UML-Sequenzdiagramm. Aus der beim Aufruf des `PolicyEvaluator` als Parameter übergebenen Policy werden zunächst alle für den Aufrufkontext zutreffenden Regeln ausgewählt. Dazu wird jede Regel der Policy betrachtet und die jeweiligen Zusicherungen werden mittels der entsprechenden Funktion ausgewertet. Eine Regel ist genau dann anwendbar, wenn alle ihre Zusicherungen positiv ausgewertet werden können. Als Ergebnis dieses

Verfahrens stehen ein oder mehrere zutreffende Regeln zur Auswertung zu Verfügung. Falls keine Regel zutreffen sollte, wird eine Ausnahme ausgelöst und der Dienstkonsument muss geeignete Maßnahmen treffen, diese zu behandeln.

In einem zweiten Schritt, dargestellt in Abbildung 80, wird eine Regel aus der Menge der zutreffenden Regeln ausgewählt. Dafür wird mittels des in der Policy spezifizierten Auswahlalgorithmus eine entsprechende Implementierung des Algorithmus instantiiert. Dieser wählt entsprechend seiner Implementierung die geeignete Regel aus. Anschließend kann anhand des in der Regel spezifizierten Effektes die Policy ausgewertet werden.

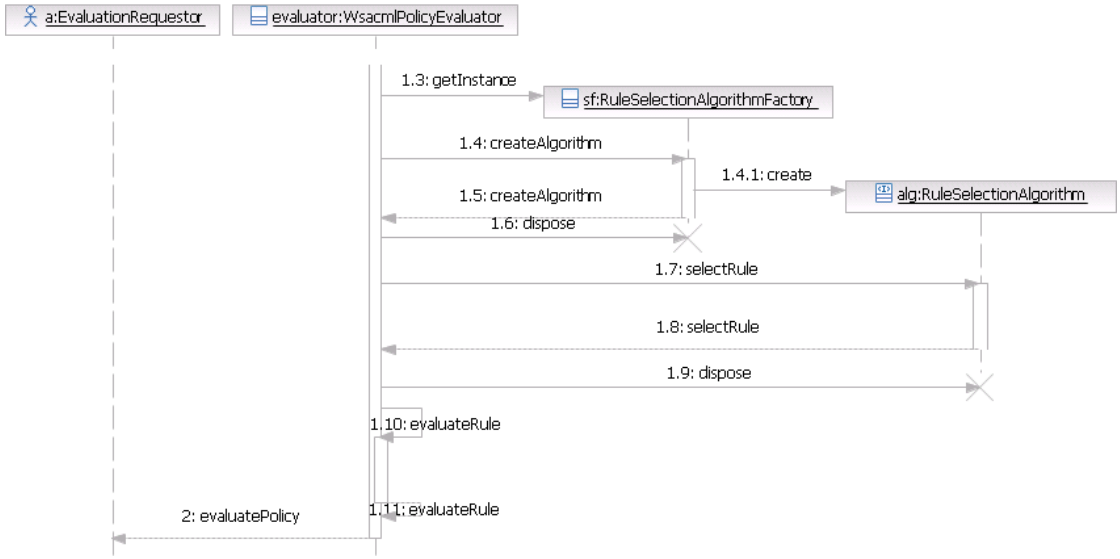


Abbildung 80: Funktionsweise des PolicyEvaluator (2)

8.3.1.3 Die Schnittstelle des PDP

Aus dem internen Aufbau des PDP lassen sich die öffentlichen Schnittstellen spezifizieren. In Abbildung 81 werden die öffentlichen Methoden der Schnittstelle, relevante Geschäftsobjekte sowie interne Methoden des PDP in einem UML-Klassendiagramm dargestellt. Der PDP bietet lediglich eine öffentliche Methode in seiner Schnittstelle an, welche seine gesamte Funktionalität widerspiegelt. Die `verifyAuthorization`-Operation nimmt als Parameter eine Instanz der Klasse `AuthorizationRequest`. Diese kapselt die Information, die notwendig sind, um die Anfrage zu entscheiden. Der boolesche Rückgabewert der Operation gibt an, ob der Zugriff auf die Ressource bei einem positiven Wert erlaubt beziehungsweise bei negativem Wert untersagt wird.

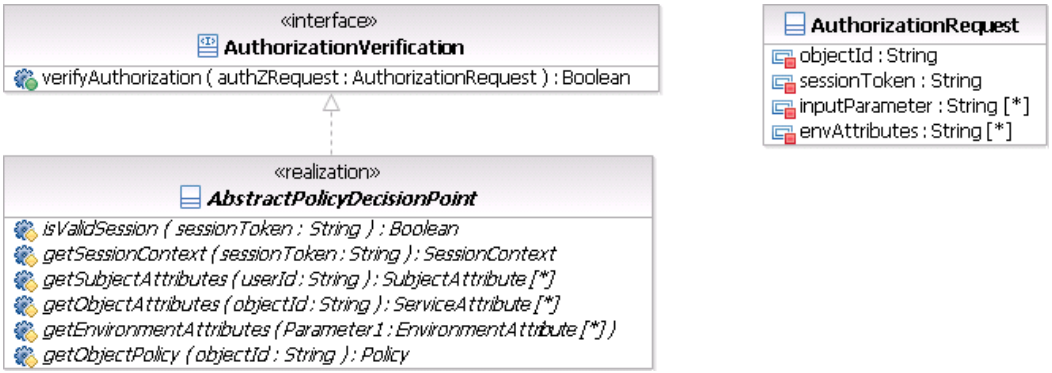


Abbildung 81: Die Schnittstelle des PolicyDecisionPoint

Zu den von den im Parameter gekapselten Daten gehören die eindeutige Identifikation des angefragten Objektes (`objectId`), das heißt, der aufgerufenen Webservice-Operation, sowie das die Sitzung des Dienstnutzers identifizierende Sicherheitstoken (`sessionToken`). Zusätzlich können weitere Informationen über eventuelle an die Webservice-Operation übergebende Parameter (`inputParameter`) oder zusätzliche Umgebungsattribute des Kontextes des Operationsaufrufes (`envAttributes`) spezifiziert werden. Zusätzlich können weiter Methoden identifiziert werden, die der PDP benötigt, um die geforderten Informationen aus anderen Datenquellen zusammenzutragen. Die `isValidSession`-Methode nutzt das übergebene Sicherheitstoken, um die Gültigkeit der Sitzung des Dienstnutzers zu überprüfen. Bei einer gültigen Sitzung kann mit der `getSessionContext`-Methode der gesamte Sitzungskontextes des Dienstnutzers abgefragt werden. Die darin enthaltenen Daten können von der `getSubjectAttributes`-Methoden genutzt werden, um die Attribute des Dienstnutzers zu erfassen. Weitere Attribute des aufgerufenen Objektes, des Umgebungszustands des Aufrufes werden zusätzlich von der `getObjectAttributes`- beziehungsweise der `getEnvironmentAttributes`-Methode bereitgestellt. Die für die aufgerufene Webservice-Operation spezifizierte Policy kann durch die `getObjectPolicy`-Methode abgefragt werden.

8.3.2 PolicyManager und PolicyStore

Wie bereits erwähnt, werden die Datenquellen um *Wrapper*-Komponenten ergänzt, welche einen dedizierten Zugangspunkt zur Datenquelle anbieten. Abbildung 82 zeigt den strukturellen Aufbau der `PolicyManager`-Komponente und den Aufbau der damit verbundenen `PolicyStore`-Komponente.

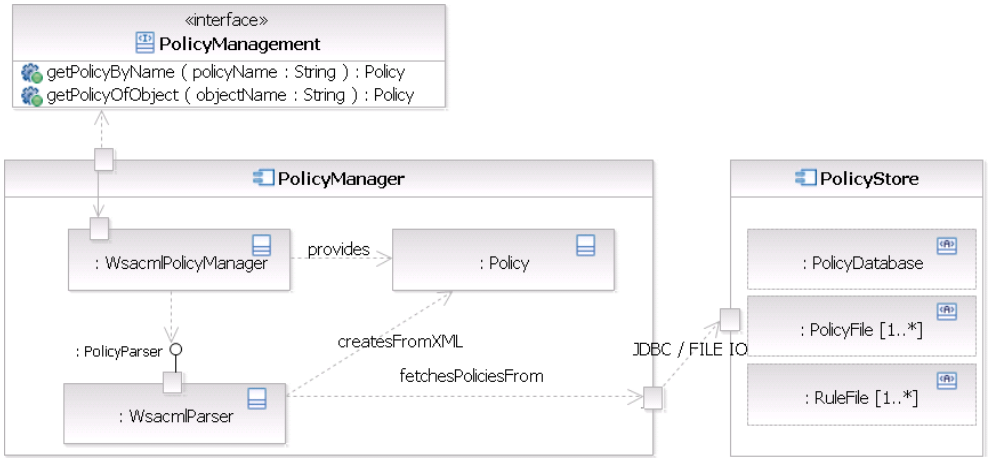


Abbildung 82: Die PolicyManager-Komponente

Die `PolicyStore`-Komponente ist für die Speicherung der Policy-Dateien verantwortlich. Da die Policies und auch die Regeln in XML-basiertem Format vorliegen, existieren mehrere Alternativen zur Speicherung der Policies. Zum einen können Datenbanken eingesetzt werden. Eine weitere Alternative ist die Speicherung in Form von XML-Dateien. Dies führt jedoch zu ineffizienten Suchmechanismen. Die dritte Alternative ist die gemischte Verwendung von relationalen Datenbanken und Dateiablage. Die Datenbank wird zum Indizieren der Dateien verwendet, wodurch die Policy- und Regel-Dateien effizient gefunden werden können. Die `Policies` und `Rules` werden dagegen in Dateien in einem bestimmten Verzeichnis gespeichert. Dies ermöglicht eine Konfiguration zur Laufzeit der Zugriffskontroll-Architektur. Die `PolicyManager`-Komponente hat die Aufgabe, auf Anfrage eine Policy als Geschäftsobjekt bereitzustellen. Hierzu ist es notwendig, die in XML-Format abgelegten Policy-Dokumente in eine objektorientierte Repräsentation umzuwandeln. Dazu wird ein spezieller XML-*Parser* genutzt. Da nicht nur Policies, sondern auch Regeln in XML-Format spezifiziert sein können, muss der *Parser* entsprechend flexibel entworfen werden.

Abbildung 83 zeigt das Domänenmodell der WSACML-Policy in objektorientierter Darstellung als UML-Klassendiagramm. Wie deutlich zu erkennen ist, lässt sich die in Kapitel 5.2 eingeführte WSACML-Syntax direkt auf ein objektorientiertes Modell abbilden, welches die Implementierung stark vereinfacht.

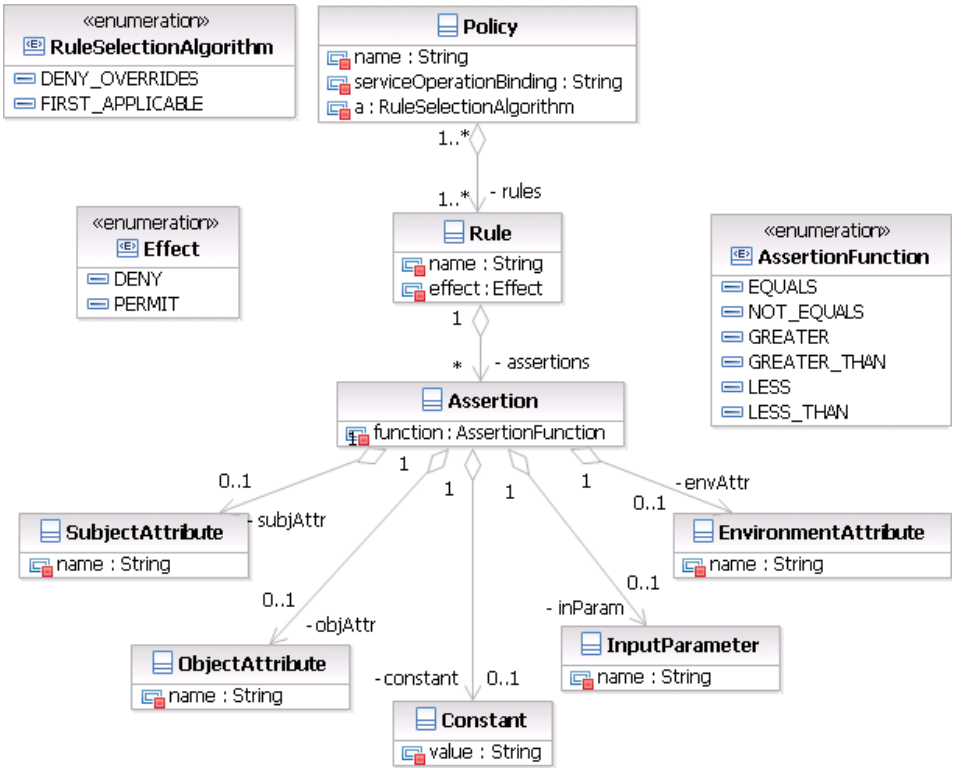


Abbildung 83: Modellierung der WSACML mit IBM-Werkzeugen

Jedes Element der abstrakten Syntax kann auf eine Klasse abgebildet werde. Die Attribute der abstrakten Sprachelemente werden auf Attribute der Klassen abgebildet. Das hier dargestellte Objektmodell ist eine Repräsentation des plattformunabhängigen Metamodells (PIM) ausgeprägt mittels Modellierungstechnologie der IBM-Werkzeuge. Damit wird ein unmittelbarer Einsatz im Kontext der modellgetriebenen Policy-Entwicklung vereinfacht.

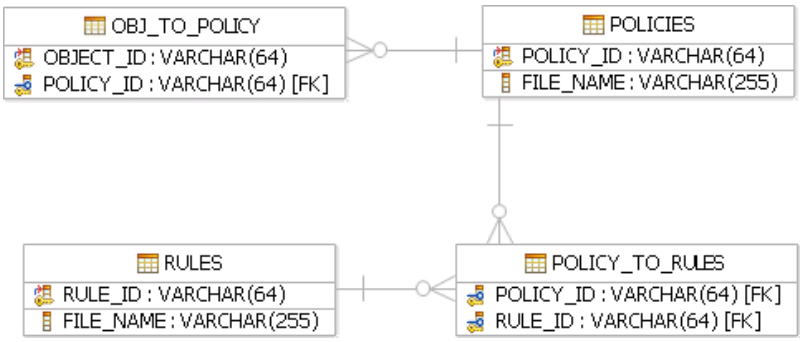


Abbildung 84: Databank-Schema des PolicyStore

In Abbildung 46 und Abbildung 47 wurden bereits Beispiele für die in XML formulierten WSACML-Policies gezeigt. Abbildung 84 zeigt das relationale Datenmodell des `PolicyStore`, welche diese XML-Dateien indiziert. Für ein

effizientes Auffinden der Policies wurde ein Datenbankschema erstellt, welches Policies mit Objekten verbindet. Diese Beziehung ist in der Relation `OBJ_TO_POLICY` festgehalten. Da eine Policy aus mehreren Regeln besteht und die Regeln wiederum in verschiedenen Policies verwendet werden können, wurden die Assoziationstabelle `POLICY_TO_RULES` angelegt, welche Assoziation zwischen Policies und Regeln festhält. Die Relationen `POLICIES` und `RULES` halten für eine durch das Attribut `ID` eindeutig bestimmte Policy bzw. Regel den Namen der Datei `FILE_NAME`, in der die Policy bzw. Regel in XML-Syntax spezifiziert sind. Der Dateiname wird dabei relativ zu einem konfigurierbaren Ablageplatz für WSACML-Dateien angegeben.

8.3.3 UserDirectoryManager und UserDirectory

Das Benutzerverzeichnis stellt die verbindliche Datenquelle für alle Informationen rund um die Benutzer dar. Hier werden die digitalen Identitäten vorgehalten, die durch die Subjektattribute mit Zugriffskontroll-Policies verbunden sind. Für das Benutzerverzeichnis gibt es eine Vielzahl fertiger Produkte und Komponenten, auf die in der Implementierung zurückgegriffen werden kann. Vor allem LDAP-basierte Benutzerverzeichnisse eignen sich ideal für die Verwaltung von Benutzerdaten. Für den vereinfachten Zugriff wird lediglich eine Datenzugriffskomponente eingesetzt, welche von dem konkret eingesetzten Produkt abstrahiert und PDP eine einheitliche Schnittstelle zur Verfügung stellt. Der `Directory-Manager` entkoppelt den PDP von spezifischen Zugriffstechnologien für die Verzeichnisdienste.

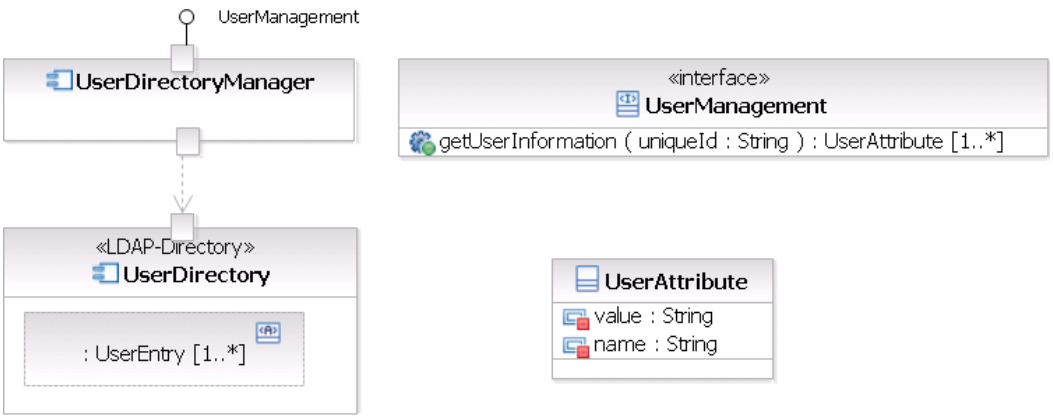


Abbildung 85: Die Benutzerverwaltung

Abbildung 85 zeigt den Aufbau des `DirectoryManagers` und die von ihm angebotene Schnittstelle. Die Schnittstelle stellt eine Methode zur Verfügung, mit welcher anhand der eindeutigen Benutzerkennung die mit dem Benutzer verbundenen Informationen aus dem Benutzerverzeichnis bereitgestellt werden. Die Informationen werden dabei in der `UserAttribute`-Klasse gekapselt.

8.3.4 SessionManager und TokenRepository

Das `TokenRepository` speichert die für eine gültige Sitzung eines Benutzers relevanten Informationen. Als Speicherkomponente kann hier auf ein bestehendes Datenbankmanagementsystem (DBMS) zurückgegriffen. Lediglich eine Datenbank für die relevanten Informationen muss angelegt werden. Zusätzlich wird auch hier eine Datenzugriffskomponente, in Information 22 als `SessionManager` gekennzeichnet, als zusätzliche Abstraktionsschicht zwischen PDP und `TokenRepository` eingeführt. Dies ermöglicht den Austausch des `TokenRepository`, ohne die PDP-Komponente ändern zu müssen.

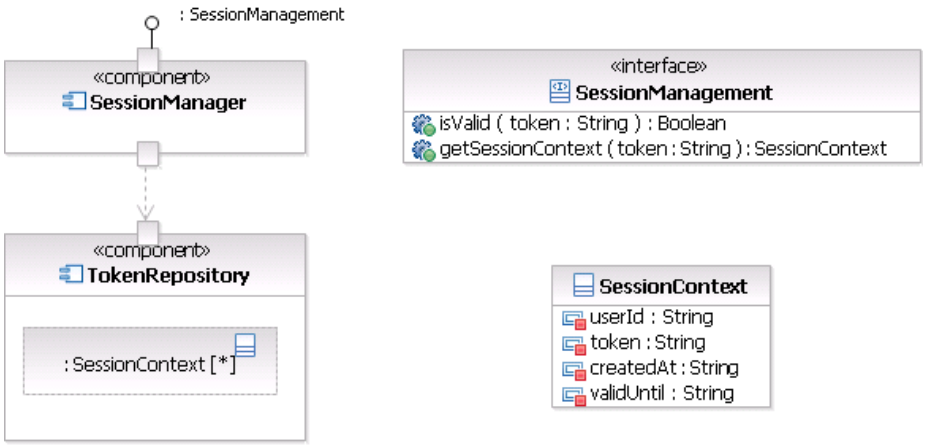


Abbildung 86: Die SessionManager-Komponente

Abbildung 86 zeigt den internen Aufbau des `SessionManager` und die von ihm bereitgestellte Schnittstelle. Die `isValid`-Methode der `SessionManagement`-Schnittstelle überprüft anhand eines Sicherheitstokens, ob die Sitzung des Dienstnutzers gültig ist (positiver Rückgabewert) oder bereits abgelaufen ist (negativer Rückgabewert). Die `getSessionContext`-Methode stellt anhand des Sicherheitstokens den gesamten Sitzungskontext des Dienstnutzers zur Verfügung.

Abbildung 87 zeigt das im `TokenRepository` implementierte Datenmodell. Das Modell besteht aus einer einzelnen Relation mit vier Attributen. Das Attribut `USER_ID` identifiziert den Benutzer, dessen aktueller Sitzungskontext abgelegt wurde. Des Weiteren wird für den Benutzer bei der Authentifizierung erzeugte Sicherheitstoken im Attribute `TOKEN` abgelegt. Da dies eindeutig im Kontext der Anwendung ist, dient es gleichzeitig als Primärschlüssel der Relation.

Abbildung 87: Datenbankschema des TokenRepository

Da eine Sitzung eines Benutzers meist zeitlich begrenzt ist, wird zusätzlich der Start- und Endzeitpunkt `CREATED_AT` beziehungsweise `VALID_UNTIL` der Sitzung abgelegt. Wie lange eine Sitzung gültig ist, kann im `SessionManager` konfiguriert werden.

8.3.5 Der SecureServiceAgent als Policy Enforcement Point

Zur Autorisierung auf Ebene der Basisdienste kommt das Entwurfsmuster *Secure Service Agent* (SSA) zur Realisierung des PEP zum Einsatz.

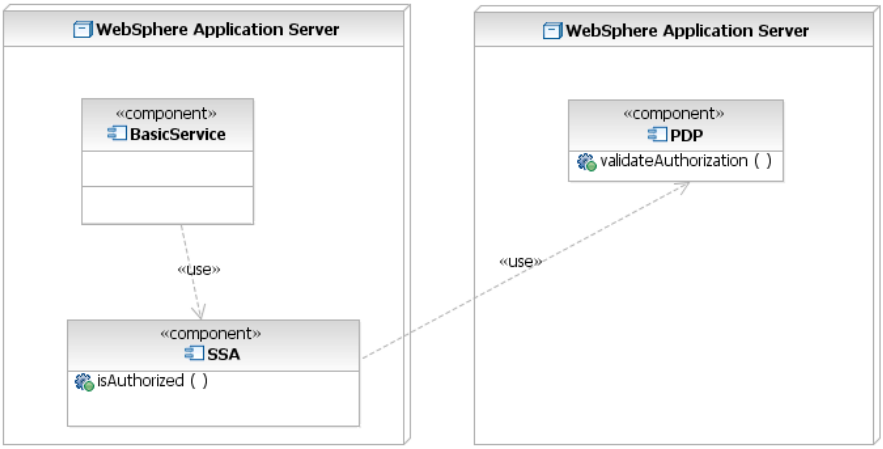


Abbildung 88: Der SecureServiceAgent

Hierbei handelt es sich um eine Komponente, die in jedem *Application Server* installiert wird und die Anfragen der Fachkomponenten zur Autorisierungsprü-

fung entgegen nimmt. Der SSA prüft die syntaktische Korrektheit und leitet sie bei Erfolg an den PDP zur Auswertung. Dieser beantwortet die Anfrage nach der Auswertung mit einem booleschen Wert, welcher der PEP an die Komponente zurückgibt. Die eigentliche Anfrage geschieht also im Kontext des PEP und nicht des jeweiligen Basisdienstes. Hierbei wird bereits klar, dass der Einbau der Zugriffskontroll-Architektur eine Nachrüstung der am Geschäftsprozess beteiligten Basisdienste nach sich zieht, weil jede Komponente den PEP selbst aufrufen und den Rückgabewert interpretieren muss. Diese Entwurfsentscheidung gewährleistet allerdings, dass die Zugriffskontrolle orthogonal zur Fachfunktionalität des Geschäftsprozesses und somit portabel im Bezug auf den eingesetzten *Application Server* ist. Da ein Geschäftsprozess aus einer Vielzahl von Basisdiensten komponiert sein kann und diese wiederum in unterschiedlichen *Application Server* als Anwendungsdomänen ausgeführt werden können, muss dafür Sorge getragen werden, dass die Aufrufe, die sich an einen PDP richten, vertrauensvoll sind. Aus diesem Grund verfügt jeder *Application Server*, von dem Basisdienste verwenden werden, über eine eigene PEP-Komponente, die sich ihrerseits gegenüber dem PDP authentisiert.

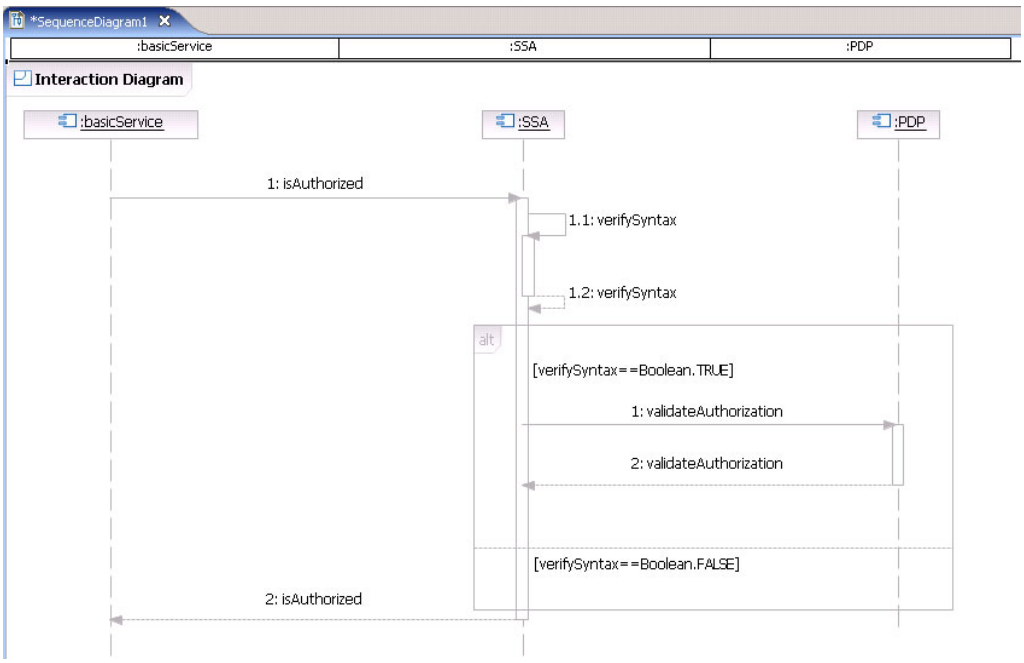


Abbildung 89: Kommunikationssequenzen des SecureServiceAgent

Da bei der Kommunikation des SSA mit dem PDP die Domänengrenze des *Application Server* überschritten wird, muss hier auf die Wahrung der Sicherheit während der Kommunikation geachtet werden. Zur Authentifizierung und verschlüsselten Nachrichtenkommunikation mit dem PDP werden daher X.509-

Zertifikate eingesetzt. Abbildung 89 zeigt den Ablauf zwischen einem Basisdienst, dem SSA und dem PDP in Form eines Sequenzdiagramms. Man erkennt, dass der Basisdienst den SSA über die Operation `isAuthorized` kontaktiert. Hierbei werden die bereits erwähnten Parameter übergeben. Der SSA überprüft diese zunächst auf syntaktische Korrektheit und leitet diese Parameter an den PDP zur Auswertung weiter. Beim Aufruf des PDP über `validateAuthorization` fügt der SSA sein Sicherheitstoken zur Authentifizierung bei. Nach der Auswertung gibt der PDP einen booleschen Wert an den SSA zurück, der diesen Wert dann seinerseits an den Basisdienst zurückgibt.

Abbildung 90: Nachrichtenverschlüsselung

Da der SSA und der PDP nicht im selben *Application Server* betrieben werden, findet die Kommunikation zwischen beiden über das an sich ungeschützte Netzwerk statt. Daher bietet sich der Einsatz von WS-Security (vgl. Kapitel 3.2.2) zur Nachrichtenschlüsselung und Punkt-zu-Punkt-Authentifizierung (vgl. Abbildung 19) an. Weil die Menge der in der Organisation betriebenen *Application Server* und damit analog die Anzahl der *Secure Service Agents* begrenzt ist, bietet sich der Einsatz von X.509-basierten Zertifikaten an. Jeder SSA und der PDP sollten mit einem solchen Zertifikat ausgestattet werden, bei dem bei Einsatz einer *Public Key Infrastructure* (PKI) auch automatisch ein gegenseitiges Vertrauen (engl. *Trust*) ermöglicht wird. Abbildung 90 zeigt exemplarisch eine SOAP-Nachricht, die ein SSA an den PDP schickt. Im Kopf der Nachricht wird das Zertifikat als `BinarySecurityToken` mitgeführt und über `mustUnderstand` festgelegt, dass der Empfänger diese Information auswerten muss. Auch erkennt man an der SOAP-Nachricht die übergebenen Parameter `token`, was dem Sicherheitstoken entspricht und den Identifikator der Webservice-Operation (`webServiceID`), für die eine Zugriffskontroll-Entscheidung getroffen werden soll. Die Konfiguration der für den Einsatz von WS-Security notwendigen Zertifikats-Infrastruktur gestaltet sich im Kontext IBM WebSphere-Produktreihe relativ einfach. Über grafische Konfigurationsmenüs kann ein

Administrator die Parametrisierung vornehmen, wie in [DM08] beschrieben wird.

8.3.6 Die Gesamtarchitektur

Die einzelnen Bausteine müssen nun zu einer Gesamtarchitektur zusammengefügt werden. Die Gesamtarchitektur ergänzt die Facharchitektur um die Zugriffskontroll-Architektur. Abbildung 91 zeigt die Verteilung der Komponenten auf einzelne Knoten.

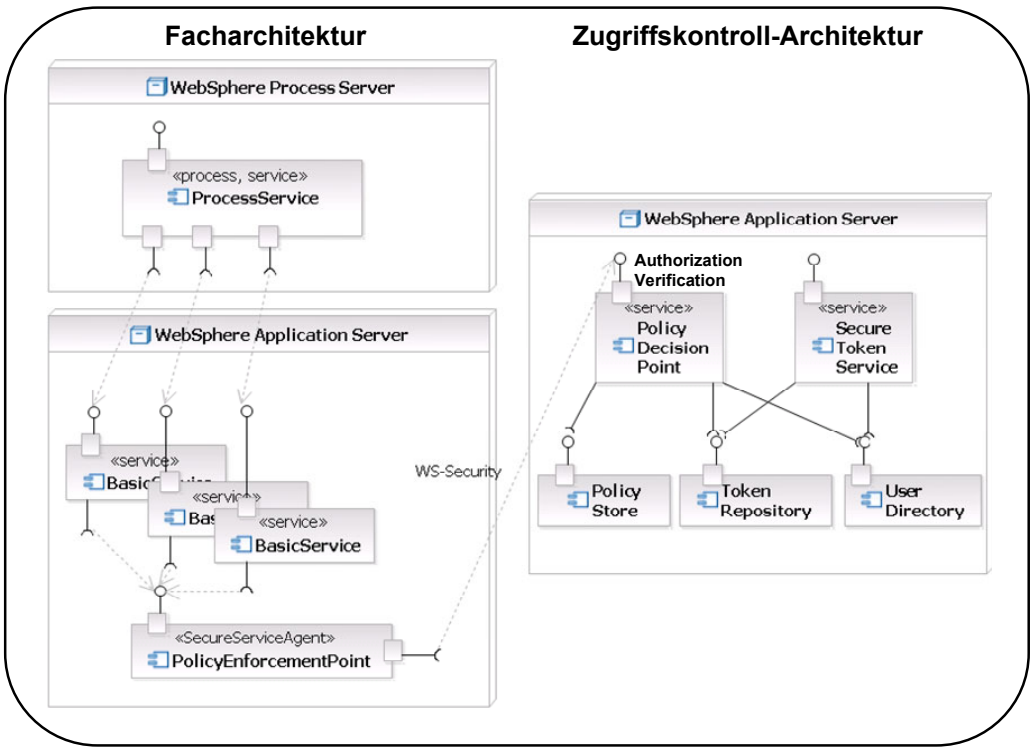


Abbildung 91: Die Gesamtarchitektur

Im linken Teil der Abbildung ist die Facharchitektur dargestellt. Exemplarisch für die Menge der BPEL-*Engines* ist oben links ein IBM WebSphere Process Server dargestellt, auf dem die in der Facharchitektur betriebenen BPEL-Prozesse *deployed* werden. Die Basisdienste werden auf mehreren *Application Server* betrieben, für die stellvertretend ein IBM WebSphere Application Server eingezeichnet ist. In jedem der *Application Server* befindet sich ein `SecureServiceAgent` als *Policy Enforcement Point*. Dieser führt die SOAP- und WS-Security-basierte Kommunikation mit der Zugriffskontroll-Architektur durch, die im rechten Teil der Abbildung dargestellt

wird. Sie setzt sich aus den in Abbildung 75 dargestellten Komponenten zusammen.

8.3.7 Evaluation der Performanz

Bei der Erweiterung der Facharchitektur um Zugriffskontrolle muss für jede Zugriffskontroll-Anfrage zusätzliche Laufzeit eingeplant werden, da zusätzliche Webservice-Aufrufe erfolgen, die wiederum zu Datenbank-Aufrufen und Berechnungen führen. All dies wirkt sich vom Grundsatz her verlängernd auf die Ausführungszeit aus. Anhand des BPEL-basierten Beispielprozesses *Determine Training Status* wird nachfolgend untersucht, welches Ausmaß die Erweitung um Zugriffskontrolle auf eine Webservice-basierte Dienstkomposition hat.

Der Versuchsaufbau wurde wie folgt gewählt:

- Die BPEL-basierte Dienstkomposition *Determine Training Status* nutzt zur Erstellung des Ergebnisses vier unterschiedliche Basis-Webservices. Diese werden in fünf Aktionen aufgerufen. Der BPEL-Prozess wird auf einer Instanz des IBM WebSphere Process Server (WPS) betrieben.

- Jeder der Basis-Webservices wurde um Zugriffskontrolle erweitert. Dabei findet eine Zugriffskontroll-Anfrage an die PEP-Komponente statt, die in derselben Instanz des IBM WebSphere Application Server (WAS) wie die Dienst-implementierenden Komponenten der Basisdienste betrieben wird.

- Der PEP nutzt die Webservice-Schnittstelle der Zugriffskontroll-Architektur zur Autorisierungsprüfung. Die Kommunikation auf Webservice-Ebene erfolgt mittels SOAP und WS-Security.

In dieser Untersuchung sollen die durchschnittliche Ausführungszeit des BPEL-Prozesses sowie die der von ihm aufgerufenen Basis-Webservices als Indikator der Performanz dienen.

Abbildung 92 zeigt die Ergebnisse des Versuches in einem Balkendiagramm. Auf der x-Achse sind der BPEL-Prozess („DTS Process“) sowie die durch ihn aufgerufenen Basisdienste aufgetragen. Die y-Achse zeigt die durchschnittlichen Ausführungszeiten in Sekunden. Die vertikal schraffierten Balken (jeweils links) repräsentieren die Ausführungszeit ohne Zugriffskontroll-Anfrage. Die gepunkte-

ten Balken (jeweils mittig) zeigen die durchschnittlichen Ausführungszeiten nach der Erweiterung um Zugriffskontroll-Mechanismen. Die horizontal schraffierten Balken (jeweils rechts) stellen die Ausführungszeiten beim Einsatz von *Caching*-Mechanismen dar, worauf nachfolgend eingegangen wird. Die Grafik zeigt, dass sich durch die Hinzufügung von Zugriffskontroll-Anfragen die durchschnittliche Ausführungszeit des BPEL-Prozesses deutlich verlängert.

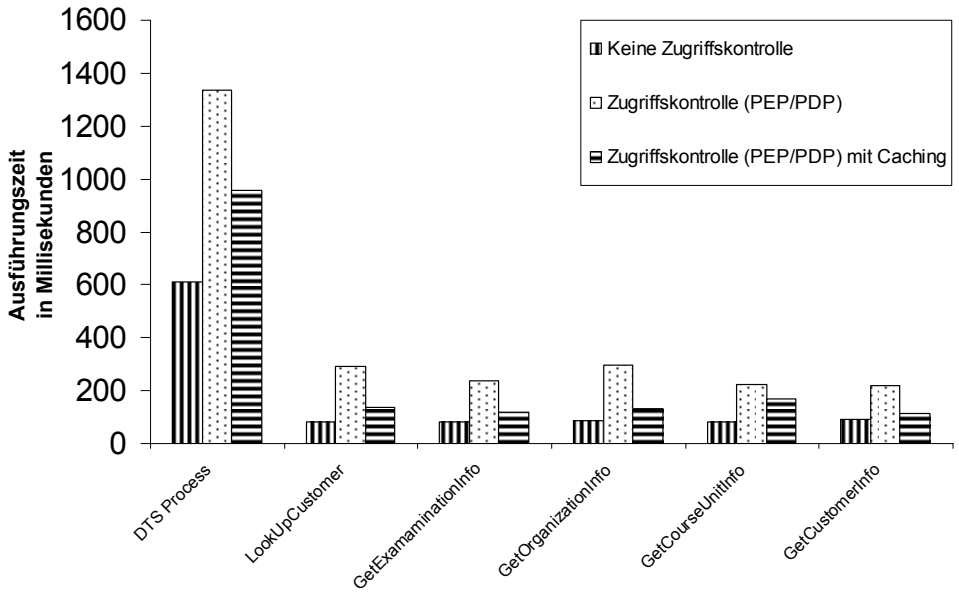


Abbildung 92: Evaluation der Performanz

Die Ausführungszeit des BPEL-Prozesses setzt sich aus den Aufrufzeiten der Basisdienste sowie seines internen Ablaufes zusammen. Dabei sind die Dienstaufrufe zu den Basisdiensten ursächlich für den Hauptanteil der Ausführungszeit. Bei der Erweiterung der Basisdienste um Zugriffskontroll-Anfragen vergrößert sich ihre Ausführungszeit. Die Vergrößerung basiert einerseits auf dem zusätzlichen Kommunikationsaufwand: Es findet eine Kommunikation der Fachkomponente mit dem PEP und zwischen PEP und PDP statt. Letztere findet wird mittels WS-Security verschlüsselt durchgeführt. Der PDP muss die entsprechende Zugriffskontroll-Policy auswählen, einlesen (engl. *Parsing*) und schließlich auswerten. Zur Verbesserung wurden daher der PEP und der PDP jeweils um einen Zwischenspeicher (engl. *Cache*) erweitert: Sie speichern die Ergebnisse vergangener Anfragen für einen definierten Gültigkeitszeitraum und sparen daher redundante Anfragen ein. Dadurch können sowohl Kommunikationspfade als auch Datenbankzugriffe und aufwendige Policy-Auswertungen eingespart werden. Die Auswirkungen von *Caching*-Mechanismen auf die durchschnittliche

Ausführungszeit werden durch die horizontal schraffierten Balken (jeweils rechts) dargestellt. Wie gut zu erkennen ist, lassen sich die Ausführungszeiten durch die zusätzlichen *Caches* deutlich reduzieren.

Bei der Evaluation wurde deutlich, dass der wesentliche Anteil bei der Verlängerung der Ausführungszeit in der Kommunikationsbeziehung zwischen PEP und PDP mittels SOAP und WS-Security über das Netzwerk hinweg liegt. Daher sind beim kommunikationssparenden Einsatz von *Caching*-Mechanismen auch unmittelbar positive Effekte nachweisbar. Hinsichtlich der Evaluationsergebnisse gilt es festzuhalten, dass sie unter Laborbedingungen gemessen wurden. Für die Übertragbarkeit in reale Umgebungen sind folgende Hinweise zu beachten: Üblicherweise wird die Zugriffskontroll-Architektur als infrastrukturelle Basisdienste von gesonderten Abteilungen eines Unternehmens betrieben, während die Fachfunktionalität (Webservices, BPEL-Prozesse) oft direkt von Fachabteilungen betrieben werden. Das führt dazu, dass aus Sicht des verteilten Software-Systems eine Distanz im Unternehmensnetzwerk zwischen PEP und PDP vorliegt. Beim üblichen Einsatz von Ethernet-basierten Netzen liegt letztlich ein gemeinsam genutzer Kommunikationskanal vor, der neben Zugriffskontroll-Anfragen auch für jegliche IP-basierte Kommunikation genutzt wird. Dabei kann es zu erhöhten Latenzen bis hin zu Paketverwerfungen kommen, was sich zusätzlich erhöhend auf die Ausführungszeit der Zugriffskontroll-Anfrage auswirkt. Eine Simulation bei unterschiedlicher Dienstgüte im Netzwerk zwischen PEP und PDP kann mittels Netzwerk-Paketdaten-Generatoren (engl. *Network Traffic Generator*) durchgeführt werden. Auf eine detaillierte Untersuchung wurde durch den Autor unter anderem mangels Gerät verzichtet. Ein ebenfalls für den Praxiseinsatz relevanter Untersuchungsgegenstand ist das Verhalten beim Einsatz von Virtualisierungstechniken. Im Zeitalter immer leistungsfähiger, aber auch energiehungriger *Server*-Systeme werden immer erhäufiger mehrere bestehende *Server* virtualisiert und gemeinsam auf einem jeweils deutlich leistungsfähigeren *Server* betrieben. Dadurch ergeben sich unter anderem eine Reduktion des Energieverbrauchs und der Wärmeerzeugung. Ebenso kann die durch die Hardware bereitgestellte Leistung bei schwankenden Ressourcen-Anforderungen durch statistisch begründete Überbuchung besser genutzt werden, wodurch sich zusätzliche Einsparungsmöglichkeiten für den Betreiber ergeben. Auf der anderen Seite muss jedoch betrachtet werden, welche Auswirkungen sich bei Spitzenlast auf die betriebenen Dienste ergeben. Dies kann zu zusätzlichen Verlängerungen in der Ausführungszeit bei der Zugriffskontrolle führen.

8.4 Resümee

In diesem Kapitel wurde mit der Zugriffkontroll-Architektur der zentrale Beitrag dieser Arbeit prototypisch mit IBM-Werkzeugen implementiert. Dabei wurde auf eine Einbettung in das sich durch die ganze Arbeit ziehende Hochschul-Szenario geachtet. Da es sich bei der Zugriffskontroll-Architektur um eine Eigenentwicklung handelt, wurde darauf geachtet, dass als Policy-Sprache unmittelbar die WSACML zum Einsatz gebracht wird. Dies erspart einerseits weitere Transformationsschritte und zeigt auf der anderen Seite, dass die WSACML alle notwendigen Informationen enthält, die für eine Zugriffskontroll-Entscheidung benötigt werden. Auch wenn in dieser Implementierung fast ausschließlich IBM-Werkzeuge zum Einsatz kommen, ist nachvollziehbar, dass die Portabilität des *Secure Service Agents* auch in JEE-Produkten anderer Hersteller unmittelbar gegeben ist. Dies wurde durch die Realisierung als Standard-konforme *Enterprise Java Bean* sichergestellt. Es ist klar, dass sich die Laufzeit eines Dienstaufrufs verlängert, der eine Zugriffskontroll-Entscheidung anfordert. Wie in der Evaluation gezeigt wurde, lässt sich dies jedoch durch *Caching*-Mechanismen auf ein moderates Niveau reduzieren, was einer praktischen Einsetzbarkeit der vorgestellten Lösung daher keinen Abbruch tut.

9 Zusammenfassung und Ausblick

Im abschließenden Kapitel dieser Arbeit werden die erzielten Ergebnisse zusammengefasst und hinsichtlich der in Kapitel 3.1 definierten Anforderungen bewertet.

9.1 Beiträge der Arbeit

Ziel der Arbeit ist es, Software-Entwickler im Webservice-Umfeld in die Lage zu versetzen, Webservices um Zugriffskontrolle zu ergänzen. Hierbei sollen anhand des Nutzungsszenarios entsprechende Zugriffsberechtigungen ergänzt werden können, deren Einhaltung zur Laufzeit überprüft werden. Um das Ziel zu erreichen, liefert die Arbeit die nachfolgenden Beiträge.

9.1.1 Dienstorientierte Zugriffskontroll-Architektur

Der erste der beiden Hauptbeiträge der Arbeit liegt in der Spezifikation einer dienstorientierten Zugriffskontroll-Architektur, die die dienstorientierte Facharchitektur erweitert. Sie ist maßgeblich durch den *Policy Decision Point* (PDP) nach außen repräsentiert, der die Schnittstelle der Zugriffskontroll-Architektur gegenüber der Facharchitektur darstellt. Als unmittelbares Gegenstück zum PDP wird ein universell wiederverwendbarer, also portabler *Policy Enforcement Point* (PEP) als Bindeglied zwischen Facharchitektur und Zugriffskontroll-Architektur entwickelt. Der interne Aufbau der entworfenen Zugriffskontroll-Architektur versteht sich als Muster, das es ermöglichen soll, bestehende Lösungen und kommerzielle Sicherheitsprodukte zu integrieren. Der PEP ist in zwei Richtungen lose gekoppelt: auf der einen Seite in Richtung der *Application Server*, die als austauschbare *Container*-Elemente für die Bereitstellung der fachfunktionalen Webservices benötigt werden. Auf diese Weise wird eine Herstellerunabhängigkeit erreicht, insbesondere hinsichtlich der *Application Server*. Andererseits sollte die Verbindung zwischen der fachfunktionalen Seite und der Zugriffskontroll-Architektur ebenfalls lose gekoppelt werden. Dies wird durch die Spezifikation von einheitlichen Dienstschnittstellen erreicht, welche zusätzlich die Komplexität innerhalb der Zugriffskontroll-Architektur für die Entwickler auf der fachfunktionalen Seite verschatten. Es muss lediglich gegen diese definierten und stabil gehaltenen Schnittstellen entwickelt werden.

9.1.2 Zugriffskontroll-Modell und Policy-Sprache

Der zweite Hauptbeitrag liegt in der Spezifikation eines Zugriffskontroll-Modells und einer zugehörigen Sprache zur Darstellung von Zugriffskontroll-Policies für Webservice-Architekturen. Policies können als Konfigurationsdaten verstanden werden, die nicht „hart“ im fachfunktionalen Programmcode fixiert sind. Analysiert man bestehende Zugriffskontroll-Modelle hinsichtlich ihrer Einsetzbarkeit in dienstorientierten Architekturen, so zeigt sich, dass bestehende Zugriffskontroll-Modelle nicht unmittelbar übertragbar sind. Daher wird zunächst ein Zugriffskontroll-Modell entwickelt, das die Besonderheiten von Webservice-orientierten Architekturen berücksichtigt, insbesondere die Struktur und die Granularität der zu schützenden Objekte, aber auch die explizite Dienstkomposition innerhalb der dienstorientierten Architektur. Aufbauend auf dem Zugriffskontroll-Modell wird eine Sprache zur Formulierung entsprechender Zugriffskontroll-Policies für den Webservice-Kontext entwickelt. Diese sind unabhängig in Hinblick auf die tatsächlich eingesetzten Komponenten in der Zugriffskontroll-Architektur.

Die beiden vorgenannten Hauptbeiträge werden anschließend durch zwei nachfolgend vorgestellte Erweiterungen ergänzt.

9.1.3 Modellgetriebene Entwicklung von Zugriffskontroll-Policies

Die erste Erweiterung befasst sich mit den Zugriffskontroll-Policies, die in einen modellgetriebenen Software-Entwicklungsprozess eingebunden werden. Ziel dabei ist, bereits frühzeitig neben der Fachfunktionalität die zugehörigen Nutzungsbedingungen in Form von Zugriffskontroll-Policies formal spezifizieren zu können. Diese sollen während der einzelnen Phasen der Software-Entwicklung aufgegriffen werden und schrittweise modellgetrieben verfeinert und konkretisiert werden. Hierbei werden die Grundlagen der modellgetriebenen Architektur (engl. *Model-Driven Architecture*, MDA) eingesetzt.

9.1.4 Semantische Integration von Benutzerdaten

Die zweite Erweiterung befasst sich mit der Integration verteilt vorliegender Benutzerdaten. Trotz der fachlichen Heterogenität der einzelnen Anwendungs-

systeme nutzen sie Benutzerverzeichnisse, um Zugriffskontroll-Informationen zu speichern. Um diese „Identitätsinseln“ zu integrieren, wird in der Arbeit eine semantische Integration der Benutzerdaten mittels Ontologien vorgenommen. Dadurch werden die unterschiedlichen syntaktischen Repräsentationen in den Benutzerverzeichnissen homogenisiert und eine Synchronisation über die Verzeichnisse hinweg wird vereinfacht.

9.1.5 Tragfähigkeitsnachweise

Der Tragfähigkeitsnachweis der vorgestellten Ergebnisse wurde durch prototypische Realisierungen erbracht. Die Demonstration der Anwendbarkeit in der Praxis erfolgte durch Implementierungen im Kontext von Forschungs- und Industrieprojekten, unter anderem im Projekt "Karlsruher integriertes InformationsManagement" [UKA-KIM]. In Kapitel 8 wurde eine prototypische Implementierung der in dieser Arbeit entworfenen Zugriffskontroll-Architektur ausführlich vorgestellt.

9.2 Diskussion der Ergebnisse

In den folgenden Abschnitten werden die erzielten Ergebnisse einer Bewertung unterzogen. Diese Bewertung wird dabei entlang des in Kapitel 3.1 definierten Kriterienkatalogs strukturiert.

9.2.1 Allgemeine Anforderungen

Zunächst werden zwei allgemeine Anforderungen diskutiert, die sich auf alle Beiträge der Arbeit ausprägen lassen.

9.2.1.1 Allgemeingültigkeit

Anforderung A1.1 zielt darauf ab, allgemeingültige Lösungen zu entwerfen. Sie sollen unabhängig von einer spezifischen Fachdomäne einsetzbar sein. Die einzige Einschränkung, die getroffen wurde, war die Beschränkung auf Webservice-basierte, dienstorientierte Architekturen. Die in Kapitel 4 vorgestellte Zugriffskontroll-Architektur weist vorgenannte Einschränkungen nicht auf. Im Gegensatz zu den in Kapitel 3.2.1.1 vorgestellten *Firewall*-basierten Lösungen besteht auch die Problematik der Positionierung des vorgeschalteten Nachrich-

tenfilters nicht. Der eigentliche Grundgedanke einer *Firewall*-basierten Lösung, nämlich die Trennung zwischen „innen“ und „außen“, lässt sich gerade im Webservice-Kontext nicht aufrechterhalten. Die klassischen Systemgrenzen spielen durch die Bereitstellung von Webservices für die Durchführung von Zugriffskontrolle nicht mehr die entscheidende Rolle; durch die dienstorientierte Architektur entsteht ein gemeinsames „Groß-System“. Daher müsste vor jeden Webservice eine entsprechende *Firewall* geschaltet werden, was vom Aufwand her betrachtet indiskutabel wäre. Auch und gerade im Vergleich mit allgemeingültigen Zugriffskontroll-Architekturen und -Modellen hat die vorgestellte Lösung Vorteile: Während XACML bewusst die Erweiterbarkeit bereits im Namen trägt (vgl. Kapitel 3.2.1.2), führt diese Art der Allgemeingültigkeit zu einem immensen Mehraufwand, wenn es um die Anpassung der Zugriffskontroll-Architektur und des Policy-Modells für den Einsatz in Webservice-Architekturen geht.

9.2.1.2 Einbeziehung des Fachentwicklers

Wie in Kapitel 3.4 vorgestellt, werden Zugriffskontroll-Policies oft erst nachgelagert, während des *Deployments* erstellt. Dabei zeigt sich, dass die Sicherheitsexperten, die die Sicherheitsprodukte betreiben, inhaltlich von den Fachabteilungen und Fachentwicklern entkoppelt sind. Das resultierende Problem ist, dass die Policy-Erstellung oft nicht auf formalen Artefakten beruht. Die frühzeitige Einbeziehung der Policy-Erstellung in den (fachlichen) Software-Entwicklungsprozess führt zu einer besseren Verzahnung von Fachfunktionalität und zugehörigen Zugriffsberechtigungen. Während diese Verknüpfung die Konsistenz verbessert, besteht die Gefahr einer Komplexitätssteigerung für die Fachentwickler. Dies wird durch den vorgestellten Ansatz vermieden. Es werden weiterhin klassische UML-Modelle eingesetzt, die sowohl bei Entwicklern als auch Fachabteilungen hinreichend verstanden werden. Am Beispiel der erweiterten UML-Aktivitätsdiagramme lässt sich der Sachverhalt verdeutlichen: Die traditionelle Darstellung von Rollen und Anwendungsfällen wird mit Mitteln der UML um den intuitiv verständlichen Nutzungskontext erweitert, der die Beziehung zwischen Rolle und Anwendungsfall charakterisiert. Dieses formal spezifizierte Element macht dann den Bezug zwischen Fachfunktionalität (Anwendungsfall) und Zugriffskontrolle (Nutzungskontext und Rolle) deutlich. Die Aufwandssteigerung für den Fachentwickler ist dabei hinreichend gering, da die Zugriffskontroll-relevanten Artefakte trotz allem durch Sicherheitsexperten weiterentwickelt werden.

9.2.2 Entwurf der Zugriffskontroll-Architektur

Neben den allgemeinen Anforderungen gibt es auch solche, die sich auf einen der vorgestellten Beiträge beziehen. Nachfolgend wird zunächst die Zugriffskontroll-Architektur betrachtet.

9.2.2.1 Interoperable Dienstschnittstellen

Ein Ziel beim Entwurf der Zugriffskontroll-Architektur war es, ihre Funktionalität dienstorientiert bereitzustellen. Das bedeutet, dass die Komplexität hinter den Dienstschnittstellen verschattet werden soll. Daher ist der Entwurf der Schnittstellen von besonderer Bedeutung. Da keine Festlegung auf ein konkretes Produkt erfolgen soll, weder für den Bereich der Facharchitektur (also eines beispielsweise eines bestimmten *Application Server*) noch hinsichtlich der einzusetzenden Sicherheitsprodukte, muss darauf geachtet werden, dass eine maximale Interoperabilität erreicht wird. Dies ist zunächst auf der syntaktischen Ebene sicherzustellen. Im Kontext Webservice-basierter Architekturen stellt das als Anforderung die Nutzung von WSDL zur Schnittstellenspezifikation und SOAP als Kommunikationsprotokoll. Beides wurde durch die vorgestellte Architektur so umgesetzt. Damit ergibt sich ein Vorteil zu kommerziellen Produkten, bei denen eine herstellerübergreifende Interoperabilität regelmäßig verhindert wird (vgl. Kapitel 3.2.3). Abgesehen von der Bereitstellung von mit WSDL beschriebenen Webservice-Operationen ist ebenfalls wichtig, dass diese Schnittstellen stabil gehalten werden. Dies wurde durch die Fixierung des Zugriffskontroll-Modells erreicht, das die Mengen und Relationen abschließend festlegt.

9.2.2.2 Portabilität

Laufzeitumgebungen wie *Application Server* oder BPEL-*Engines* sind in Bezug auf Webservice-orientierte Architekturen lediglich Mittel zum Zweck. Sie werden benötigt, um Fachkomponenten mit Dienstschnittstellen zu versehen beziehungsweise Dienste zu komponieren. Vom Grundsatz her sollten sie austauschbar sein. Daher ist es sehr sinnvoll, die Zugriffskontroll-Architektur nicht auf einzelne solcher Laufzeitumgebungen zu fixieren. Bestehende Lösungen wie der *Intercepting Web Agent* (IWA, vgl. Kapitel 3.2.1.3) haben hier ihre Schwachpunkte: Bei jedem Produktwechsel, und sei es auch nur bei einem

Versionssprung, besteht die Gefahr, dass die tiefsitzende Verdrahtung mit der Laufzeitumgebung angepasst werden muss. Während durch die Einbindung in die Nachrichten-*Queues* eine starke Abschirmung entsteht, muss auf der anderen Seite die fehlende Portabilität dagegengehalten werden. Durch den in dieser Arbeit vorgestellten *Secure Service Agents* wird eine auf die Fachdomäne ausgerichtete Lösung entworfen: Durch die Realisierung als Komponente im Nutzer-Kontext des *Application Server* wird die Portabilität im Gegensatz zu Nachrichten-*Queue*-basierten Lösungen wie dem IWA gesteigert. Andererseits wird im Gegensatz zur Umsetzung als *Secure Service Proxy* (SSP) eine Redundanz vermieden und damit der Pflegeaufwand des sicherheitsrelevanten Codes reduziert.

9.2.2.3 Integrationsfähigkeit

Im Sicherheitsbereich gibt es eine große Zahl kommerzieller Produkte zur Durchführung von Zugriffskontrolle. Ihre Entwicklung begann lange vor der Einführung dienstorientierter Architekturen, und sie werden erst sukzessive angepasst. Der Einsatz von Eigenentwicklungen im Sicherheitsbereich ist mit Vorsicht zu genießen [Ec06], daher wurde beim Entwurf der Zugriffskontroll-Architektur darauf geachtet, dass sich bestehende Sicherheitsprodukte integrieren lassen. Dies gilt sowohl für die Benutzerverzeichnisse, für die üblicherweise LDAP-basierte Verzeichnisse eingesetzt werden, als auch für die die Zugriffskontrolle durchführenden Sicherheitsprodukte selbst. Da die Produkte sich intern teilweise deutlich unterscheiden, ist es wichtig, dass die Zugriffskontroll-Policies zunächst plattformunabhängig spezifiziert werden können und anschließend in die Zielplattform transformiert werden. Dies wird durch den modellgetriebenen Ansatz zur Policy-Entwicklung ermöglicht.

9.2.2.4 Umsetzbarkeit

Die Umsetzbarkeit der vorgestellten Zugriffskontroll-Architektur wurde in unterschiedlichen Projekten nachgewiesen. Dabei wurde sowohl eine Eigenentwicklung durchgeführt (vgl. Kapitel 8) als auch in Projekten mit der Industrie bestehende Sicherheitsprodukte integriert. Dabei lag das Augenmerk darauf, sowohl die bestehenden Fachkomponenten (Anwendungssysteme) als auch die bestehenden Sicherheitsprodukte weiter in Betrieb halten zu können. Zur Realisierung des Zugriffskontroll-Diensts wurden daher die bestehenden Produkte mit den definierten Schnittstellen umhüllt (engl. *Wrapping*). Die Fachfunktionalität

der bestehenden Systeme wurde mittels *Application Server* an Webservice-Schnittstellen bereitgestellt und mit den Zugriffskontroll-Diensten verbunden. Dieses Vorgehen erlaubt den Parallelbetrieb von konventioneller Systemumgebung und dienstorientierter Architektur.

9.2.3 Zugriffskontroll-Modell und Policy-Sprache

9.2.3.1 Plattforumunabhängigkeit und Webservice-Bezug

Für den Bereich des Zugriffskontroll-Modells wurde analog zur Architektur eine Plattformunabhängigkeit gefordert. Dies bedeutet, dass sich das Modell rein auf die zu schützenden Objekte (Webservices) bezieht und keine Abhängigkeiten zu den später eingesetzten Sicherheitsprodukten haben soll. Durch diese Abstraktion auf den inhaltlichen Kern wird auch die frühzeitige Einbindung in den Software-Entwicklungsprozess vereinfacht. Während rollenbasierte Zugriffskontroll-Modelle (RBAC) einen nicht hinreichenden Umfang haben (vgl. Kapitel 3.3), wurde mit dem vorgestellten Modell der ursprüngliche Kern der Zugriffskontrolle als Grundlage genommen: die Subjekt/Objekt-Relation. Dabei wurden Rollen-Konzepte integriert, allerdings wird im Gegensatz zu RBAC-basierten Modellen die Geschäftsrolle auf eine Menge an Subjektattributen abgebildet. Während die Plattformunabhängigkeit durch das Ausblenden von bestehenden Sicherheitsprodukten zum Zeitpunkt der Modellentwicklung erreicht wird, entsteht der geforderte Webservice-Bezug durch die Formalisierung der zu schützenden Objekte, den Webservices. Mit dem vorgestellten Zugriffskontroll-Modell entsteht eine Erweiterung der klassischen Subjekt/Objekt-Relation, die hinsichtlich der Subjekte auf die Geschäftsmodellierung berücksichtigt und bezüglich der Objekte bereits die Webservice-orientierte Architektur unmittelbar fokussiert. Damit entstehen Vorteile gegenüber bestehenden Modellen, die zwar flexibel anpassbar sind, was jedoch zunächst noch durchgeführt werden muss.

9.2.3.2 Einbindung in den Software-Entwicklungsprozess

Um das Zugriffskontroll-Modell nutzbar zu machen, wird zunächst eine zugehörige Policy-Sprache benötigt, die die Konzepte des Modells umsetzen muss. Die durch die Plattformunabhängigkeit entstandene Abstraktion von den technischen Spezifika der eingesetzten Sicherheitsprodukte ermöglicht es, bereits frühzeitig im Software-Entwicklungsprozess Zugriffskontrolle formal zu berücksichtigen.

Dabei greift die vorgestellte Lösung im Gegensatz zu den in Kapitel 3.4 vorgestellten Lösungen bereits in der Analyse-Phase ein. Dies ermöglicht es, auch die Kompetenz der in dieser Phase aktiven Fachabteilungen mit einzubeziehen. Durch die konsequente Nutzung der UML und der in ihrer Spezifikation verankerten Erweiterungsmöglichkeiten ist die Werkzeugunterstützung gut umsetzbar. Über die Erweiterung durch UML-Profile ist eine Festlegung auf ein konkretes Werkzeug nicht erforderlich.

9.2.3.3 Umsetzbarkeit

In der Forschungsgruppe C&M wurde die Entwicklung einer dienstorientierten Architektur für den universitären Kontext vorangetrieben. Dabei wurden (fachliche) Dienste entworfen, die im Rahmen der Rechnerunterstützung eingebunden werden können. Neben der fachlichen Beschreibung, die unter anderem in [Kr07b] vorgestellt wurde, wurden parallel entsprechende Zugriffskontroll-Policies mit der in dieser Arbeit entwickelten *Web Services Access Control Markup Language* (WSACML) spezifiziert. Diese bildeten den Ausgangspunkt einerseits für Modelltransformationen in bestehende Sicherheitsprodukte hinein oder auch als unmittelbar auswertbare Policy für die eigenentwickelte Zugriffskontroll-Architektur.

9.2.4 Modellgetriebene Entwicklung von Zugriffskontroll-Policies

Einerseits erlaubt die Abstraktion der Komplexität bestehender Sicherheitsprodukte das Einbeziehen von weiteren Wissensträgern aus den Fachabteilungen, andererseits sind zur unmittelbaren Weiterverarbeitung der abstrakten Policies Automatisierungen von Vorteil. Hierzu wurde in dieser Arbeit die Umsetzung des Vorgehens der modellgetrieben Architektur vorgestellt. Nach der Erstellung eines plattformunabhängigen (abstrakten) Modells wird dieses zunächst in ein plattformspezifisches Modell transformiert und abschließend in lauffähigen Code (beziehungsweise auswertbare Policies) überführt. Der Grundgedanke des Vorgehens ist die sukzessive Anreicherung der vorhandenen Kerninformation. In dieser Arbeit wird für das plattformunabhängige Metamodell exemplarisch die Transformation in das plattformspezifische Metamodell eines gegebenen Sicherheitsprodukts entwickelt. Das Vorgehen ist prinzipiell auch auf andere Produkte übertragbar. Es fand bereits Einsatz im Kontext eines Projekts bei einem großen

Automobilhersteller. Hier zeigte sich auch der Vorteil auch darin, den Fachabteilungen mit Hilfe der plattformunabhängigen Sprache die tatsächlich umgesetzten Zugriffskontroll-Policies zu visualisieren und damit auch Rückflüsse hinsichtlich notwendiger Verbesserungen zu erreichen.

9.2.5 Semantische Integration von Benutzerdaten

Benutzerdaten stellen eine wesentliche Grundlage für die Durchführung von Zugriffskontroll-Entscheidungen dar. In der Form von digitalen Identitäten spiegeln sie die handelnden Subjekte wider. Bei einer systemübergreifenden Integration, wie sie im Kontext dienstorientierter Architekturen stattfindet, müssen die verteilt vorliegenden Benutzerdaten ebenfalls integriert werden. Dies kann auf verschiedene Arten durchgeführt werden, wie die Analyse des Stands der Technik und Forschung zeigt (vgl. Kapitel 3.5). Die in dieser Arbeit vorgestellte Lösung stellt die explizite Behandlung der Semantik in den Vordergrund. Dabei wird eine Personen-Ontologie entwickelt, die einerseits ein Kernmodell darstellt und andererseits durch die inhärent modellierten Abhängigkeiten die Taxonomien der einzelnen Datenquellen auf ein gemeinsames Niveau vereinheitlicht. Der vorgestellte Entwurf einer Synchronisations-Architektur greift diese Ontologie auf und arbeitet ereignisgesteuert (*Publish/Subscribe*-Prinzip). Die Ankopplung der bestehenden Datenquellen erfolgt analog den Prinzipien der dienstorientierten Architektur über wohldefinierte Dienstschnittstellen. Dieses Konzept wurde prototypisch in einem Industrieprojekt umgesetzt und verbesserte die eingesetzte individuelle Punkt-zu-Punkt-Synchronisation.

9.2.6 Resümee

Die Bewertung der Ergebnisse entlang des aufgestellten Kriterienkatalogs zeigt, dass der in Kapitel 3 motivierte Handlungsbedarf erfolgreich bearbeitet wurde. Die in Kapitel 1.3 aufgeworfenen Fragestellungen wurden beantwortet und die formulierten Ziele erreicht. Bei der Bearbeitung der spezifizierten Problemstellung wurden weiterführende Fragen bezüglich der entwickelten Konzepte aufgeworfen. Einige dieser Fragen werden nachfolgend zusammengefasst.

9.3 Ausblick

In dieser Arbeit wurden umfangreiche Konzepte zur Realisierung von Zugriffskontrolle in dienstorientierten Architekturen entwickelt. Hinsichtlich der Ergebnisse lassen sich folgende weitergehende Herausforderungen identifizieren.

- In dieser Arbeit lag der Fokus der modellgetriebenen Entwicklung von Zugriffskontroll-Policies bei der Überführung von plattformunabhängigen Modellen in plattformspezifische und damit letztlich in effektiv auswertbaren Code. Die Verknüpfung mit den Geschäftsprozessen selbst wurde über Nutzungskontexte realisiert, die mit Diensten assoziiert werden. Eine unmittelbare Einbindung in die Geschäftsprozessmodellierung wurde mit [KW+08] motiviert, sollte jedoch weiter ausgebaut werden. Insbesondere auch die Werkzeugunterstützung muss weiter ausgebaut werden.

- Die Verknüpfung der Zugriffskontroll-Policies als nichtfunktionale Eigenschaften eines Dienstes (Webservices) mit seiner fachlichen Beschreibung kann über das Dienstverzeichnis erfolgen, wie in Kapitel 4 skizziert wurde. Während die Betrachtung des Lebenszyklusses eines fachlichen Dienstes mit dem zunehmenden Einsatz von dienstorientierten Architekturen zunimmt, muss in analoger Weise auch die Fortentwicklung der zuhgehörigen Zugriffskontroll-Policies sichergestellt werden. Hierbei wird dem Dienstverzeichnis eine größere Bedeutung zukommen: Es bietet Anknüpfungspunkte für Rechnerunterstützung bei der Umsetzung von Geschäftsprozessen und kann als unmittelbare Verknüpfung zu den Zugriffskontroll-Policies verwendet werden.

- Für die Realisierung von Zugriffskontrolle wird im Rahmen dieser Arbeit der Kontext einer in sich geschlossenen Unternehmung betrachtet. Aspekte wie föderiertes Identitätsmanagement (engl. *Federated Identity Management*, FIM) in unternehmensübergreifenden Formen der Zusammenarbeit wurden in dieser Arbeit nicht untersucht, da zunächst unternehmensintern die Voraussetzungen für dienstorientierte Architekturen geschaffen werden müssen. Darauf können unternehmensübergreifende Geschäftsprozesse aufsetzen, bei denen Einzelaktivitäten auf Dienste unterschiedlichen Unternehmen abgebildet werden. Insbesondere die Frage, wie eine Vertrauensbeziehung hinsichtlich eines Subjekts dargestellt werden kann, wird derzeit untersucht [Ho07].

- In dieser Arbeit wird thematisiert, wie die Entwicklung von Zugriffskontroll-Policies ähnlich systematisch durchgeführt werden kann wie die Software-Entwicklung selbst. Hierzu wird ein modellgetriebenes Verfahren und dessen Einbindung in den Software-Entwicklungsprozess vorgestellt. Dies stellt den Weg „von oben nach unten“ dar. Unternehmen werden durch gesetzliche Vorgaben immer stärker verpflichtet, auch den umgekehrten Weg zu betrachten. Die Frage, die es bei sogenannten Sicherheits-Auditierungen zu klären gilt, ist, ob der Istzustand einer IT-Landschaft dem Sollzustand entspricht. Dabei ist es sinnvoll, dass die Modelle der Geschäftsmodellierung mit denen der Systemumsetzungen sich möglichst direkt in Überdeckung bringen lassen. Durch das Konzept der klaren Trennung von Geschäftsrollen einerseits und der Abbildung auf Attribute auf der Systemseite andererseits wurde diesbezüglich in dem in dieser Arbeit vorgestellten Zugriffskontroll-Modell bereits eine Richtung vorgegeben.

Die exemplarisch aufgeführten Fragen motivieren weitere Forschungsarbeiten im Kontext der Sicherheit in dienstorientierten Architekturen. Mit der häufig prognostizierten und zwischenzeitlich auch im industriellen Kontext unmittelbar erkennbaren Bedeutungszunahme dienstorientierter Architekturen wird die Komplexität der Lösungen in diesem Bereich weiter zunehmen. Dabei werden insbesondere Mechanismen der modellgetriebenen Software-Entwicklung, wie sie in dieser Arbeit für die Erstellung von Zugriffskontroll-Policies angewendet wurden, weiter an Bedeutung gewinnen.

Anhang

Abkürzungsverzeichnis

ABAC	*Attribute-Based Access Control* (dt. attributbasierte Zugriffskontrolle)
ACDF	*Access Control Decision Function*
ACEF	*Access Control Enforcement Function*
AES	*Advanced Encryption Standard*
API	*Application Programming Interface*
B2B	*Business To Business*
BMP	*Bean-Managed Persistence*
BPD	*Business Process Diagram*
BPEL	*Business Process Execution Language*
BPMI	*Business Process Management Initiative*
BPMN	*Business Process Modeling Notation*
BRC	*Business-Related Component* (dt. Fachkomponente)
CIM	*Computation-Independent Model* (dt. rechnerunabhängiges Modell)
CMP	*Container-Managed Persistence*
CORBA	*Common Object Request Broker Architecture*
COTS	*Commercial Of The Shelf*
DAC	*Discretionary Access Control* (dt. benutzerbestimmbare Zugriffskontrolle)
DAML+OIL	*DARPA Agent Markup Language + Ontoloty Interference Layer*
DARPA	*Defense Advanced Research Projects Agency*
DB	Datenbank (engl. *Database*)
DDoS	*Distributed Denial of Service*
DEN	*Directory-Enabled Networks*
DIB	*Directory Information Base*
DIT	*Directory Information Tree*
DMZ	Demilitarisierte Zone (engl. *Demilitarized Zone*)

DN	*Distinguished Name*
DSML	*Directory Service Markup Language*
EAI	*Enterprise Application Integration*
eEPK	erweiterte Ereignisgesteuerte Prozesskette
EJB	*Enterprise Java Bean*
ERP	*Enterprise Resource Planning*
ESB	*Enterprise Service Bus*
FIM	*Federated Identity Management* (dt. föderiertes Identitätsmanagement)
GI	Gesellschaft für Informatik e.V.
GUI	*Graphical User Interface* (dt. grafische Benutzerschnittstelle)
HTTP	*HyperText Transfer Protocol*
IBAC	*Identity-Based Access Control* (dt. identitätsbasierte Zugriffskontrolle)
ID	Identifikator (engl. *Identifier*)
IdM	Identitätsmanagement (engl. *Identity Management*)
IEEE	*Institute of Electrical and Electronics Engineers*
IETF	*Internet Engineering Task Force*
IP	*Intellectual Property* (dt. geistiges Eigentum)
ISO	*International Organization for Standardization*
IT	Informationstechnologie (engl. *Information Technology*)
ITIL	*IT Infrastructure Library*
ITSM	*IT Service Management*
IWA	*Intercepting Web Agent*
J2EE	siehe JEE
JAAS	*Java Authentication and Authorization Service*
JEE	*Java Platform, Enterprise Edition* (früher: J2EE)

JDK	*Java Development Kit*
JNDI	*Java Naming and Directory Interface*
KIM	Karlsruher Integriertes InformationsManagement
LBAC	*Lattice-Based Access Control* (dt. verbandbasierte Zugriffskontrolle)
LDAP	*Lightweight Directory Access Protocol*
LDIF	*LDAP Data Interchange Format*
MAC	*Mandatory Access Control* (dt. systembestimmte Zugriffskontrolle)
MDA	*Model-Driven Architecture* (dt. modellgetriebene Architektur)
MOF	*Meta Object Facility*
OASIS	*Organization for the Advancement of Structured Information Standards*
OCL	*Object Constraint Language*
OWL	*Web Ontology Language*
OWL-S	*Web Ontology Language for Services*
PAP	*Policy Administration Point*
PDP	*Policy Decision Point*
PEP	*Policy Enforcement Point*
PIM	*Platform-Independent Model* (dt. plattformunabhängiges Modell)
PITL	*Programming in the Large*
PITS	*Programming in the Small*
PSC	*Platform-Specific Code* (dt. plattformspezifischer Code)
PSM	*Platform-Specific Model* (dt. plattformspezifisches Modell)
POLP	*Principle of Least Privilege*
POLA	*Principle of Least Authority*
RBAC	*Role-Based Access Control* (dt. rollenbasierte Zugriffskontrolle)

RDN	*Relative Distinguished Name*
RFC	*Request For Comment*
RMI	*Remote Method Invocation*
ROI	*Return On Investment*
ROSI	*Return On Security Investment*
RPC	*Remote Procedure Call*
RSA	*Rational Software Architect*
RUP	*Rational Unified Process*
SAML	*Security Assertion Markup Language*
SHA	*Secure Hash Algorithm*
SLA	*Service Level Agreement* (dt. Dienstleistungsvereinbarung, DLV)
SPA	*Single Point of Administration*
SOA	*Service-Oriented Architecture* (dt. dienstorientierte Architektur)
SOAP	früher: *Simple Object Access Protocol*
SoC	*Separation of Concerns*
SOX	*Sarbanes-Oxley Act*
SPML	*Service Provisioning Markup Language*
SSA	*Secure Service Agent*
SSL	*Secure Sockets Layer*
SSO	*Single Sign-On*
SSP	*Secure Service Proxy*
STS	*Security Token Service*
TLS	*Transport Layer Security*
UDDI	*Universal Description, Discovery and Integration*
UML	*Unified Modeling Language*
URI	*Uniform Resource Identifier*
W3C	*World Wide Web Consortium*
WAS	IBM WebSphere Application Server
WID	IBM WebSphere Integration Developer
WPS	IBM WebSphere Process Server
WSACML	*Web Services Access Control Markup Language*
WSDL	*Web Services Description Language*

WSOA	Webservice-orientierte Architektur
WSS	*Web Services Security*
WUSKAR	Werkstatt Unternehmenssoftware Karlsruhe
WWW	*World Wide Web*
XACML	*Extensible Access Control Markup Language*
XML	*Extensible Markup Language*
XSD	*XML Schema Definition*
XSL	*Extensible Stylesheet Language*
XSLT	*XSL Transformation*

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[Ab05] S. Abeck: Kursbuch Informatik I: Formale Grundlagen der Informatik und Programmierkonzepte am Beispiel von Java, Universitätsverlag Karlsruhe, 2005.

[AB+06] M. Alam, R. Breu, M. Hafner: Modeling permissions in a (U/X)ML world, First International Conference on Availability, Reliability and Security (ARES'06), 2006.

[AC+04] G. Alonso, F. Casati, H. Kuno, V. Machiraju: Web Services – Concepts, Architectures and Applications, Springer-Verlag, Berlin/Heidelberg, 2004.

[AF+99] R. Arlein, J. Freire, N. Gehani, D. Lieuwen, J. Ordille: Making LDAP Active with the LTAP Gateway, Workshop on Databases in Tele-communication, September 1999.

[AL99] C. Adams, S. Lloyd: Understanding Public-Key Instrastructure, Macmillan Technical Publishing, 1999.

[Am94] E. Amoroso: Fundamentals of Computer Security Technology, Prentice Hall, 1994.

[Ar04] A. Arsanjani: Service-oriented modeling and architecture, IBM developerWorks, 2004.

[AS04] S. Aier, M. Schönherr (Hrsg.): Enterprise Application Integration – Serviceorientierung und nachhaltige Architekturen, GITO-Verlag, Berlin, November 2004.

[AZ+07] A. Arsanjani, L.-J. Zhang, M.Ellis, A. Allam, K. Channabasavaiah: Design an SOA solution using a reference architecture, IBM developerWorks, 2007.

[Ba00] H. Balzert: Lehrbuch der Software-Technik – Software-Entwicklung, Spektrum Akademischer Verlag, Heidelberg/Berlin, 2000.

[Ba07] A. Barnitzke: Erst Identity Management hilft SOA auf die Beine, Computer Zeitung Nr. 49, Dezember 2007.

[BB+05] N. Bieberstein, S. Bose, M. Fiammante, K. Jones, R. Shah: Service-Oriented Architecture Compass: Business Value, Planning, and Enterprise Roadmap, IBM Press, Oktober 2005.

[BC04] G. Brown, R. Carpenter: Successful Application of Service-Oriented Architecture Across the Enterprise and Beyond, Intel Technology Journal, Volume 08, Issue 04, November 2004.

[Be00] T. Berners-Lee: Semantic Web – XML2000, W3C Talks, 2000.

[BE+04] K. Ballinger, D. Ehnebuske, C. Ferries, M. Gudgin, C. K. Liu, M. Nottingham, P. Yendluri: Basic Profile Version 1.1 Final Material, Web Services Interoperability Organization, April 2006.
URL: http://www.ws-i.org/Profiles/BasicProfile-1.1.html

[BH+04] M. Born, E. Holz, P. Kath: Softwareentwicklung mit der UML 2, Addison-Wesley-Verlag, München, 2004.

[Bi77] K. J. Biba: Integrity Considerations for Secure Computer Systems, MTR-3153, Mitre Corporation, April 1977.

[Bl03] J. Bloomberg: Enterprise IdM - Essential SOA Prerequisite, Zapthink Zapflash Document ID: ZapFlash-06192003, September 2003.

[Bl05] D. Blum: Concepts and Definitions (Identity Management Glossary), Burton Group Identity and Privacy Strategies, September 2005.

[BL73] D. E. Bell, L. J. LaPadula: Secure Computer Systems: Mathematical Foundations and Model, Technical Report, MITRE Corp., Mai 1973.

[BM+96] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad: A System of Patterns. Pattern-Oriented Software Architecture, Volume 1, Wiley & Sons, 1996.

[BN89] D.F.C. Brewer, M.J. Nash: The Chinese Wall Security Policy, IEEE Symposium on Research in Security and Privacy, Mai 1989.

[Br07] F. Brandt: Interoperabilität im Identitätsmanagement bei Dienstkompositionen, Diplomarbeit, Universität Karlsruhe (TH), 2007.

[BS+05] R. Bebawy, H. Sabry, S. El-Kassas, Y. Hanna, Y. Youssef: Nedgty: Web Services Firewall, IEEE International Conference on Web Services (ICWS'05), 2005.

[BV03] S. Bechhofer, R. Volz: WonderWeb OWL Ontology Validator, 2003.
URL: http://phoebus.cs.man.ac.uk:9999/OWL/Validator

[BW+02] F.E. Bustamante, P. Widener, K. Schwan: A Case for Proactivity in Directory Services, Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing (HPDC), 2002.

[CA-SM-CG] CA: SiteMinder Concepts Guide Version 4.6, Netegrity, 2001.

[CA-SM-DB] CA: eTrust SiteMinder Datenblatt, CA Darmstadt, 2005.

[CA-SM-PA] CA: SiteMinder Software Development Kit v6.0 SP4, Java API Reference, 2005.

[CA-SM-TWP] CA: Netegrity SiteMinder 5.5, Technical White Paper, 2003.

[Ch04] F. Chong: Identity and Access Management, Microsoft Architects Journal, Issue 3, 2004.

[CJ+02] Z. Cui, D. Jones, P. O'Brien: Issues in Ontology-based Information Integration, IJCAI Seattle / USA, 2002.

[CW87] D.D. Clark, D.R. Wilson: A Comparison of Commercial and Military Computer Security Policies, IEEE Symposium on Security and Privacy, 1987.

[Da07] K. Daniel: Collaborative Engineering. Anwendungen, Architekturen und Schnittstellen, Vdm Verlag Dr. Müller, 2007.

[DC+02] L. Ding, H. Chen, L. Kagal, T. Finin: DAML Person Ontology, 2002.
UTL: http://daml.umbc.edu/ontologies/ittalks/person

[DH08] J. Dinger, H. Hartenstein: Netzwerk- und IT-Sicherheitsmanagement, Universitätsverlag Karlsruhe, 2008.

[Di06] M. Diodati: User Authentication, Burton Group Identity and Privacy Strategies, Juni 2006.

[DJ+05] W. Dostal, M. Jeckle, I. Melzer, B. Zengler: Service-orientierte Architekturen mit Web Services, Spektrum Akademischer Verlag, 2005.

[DM08] A. Dikanski, K. Molitorisz: Autorisierungsprüfung für Webservices mit IBM-Werkzeugen, Studienarbeit, Universität Karlsruhe (TH), 2008.

[DR06] Y. Deswarte, M. Roy: Privacy-Enhancing Access Control Enforcement, W3C Workshop on Languages for Privacy Policy Negotiation and Semantics-Driven Enforcement, September 2006.

[Du06] Dudenredaktion: Die deutsche Rechtschreibung - Das umfassende Standardwerk auf der Grundlage der neuen amtlichen Regeln (August 2006):, Band 1, Bibliographisches Institut, Mannheim, 24. Auflage, Juli 2006.

[EA04] C. Emig, S. Abeck: Werkstatt Unternehmenssoftware Karlsruhe (WUSKAR) - Ein Beitrag zur praxisorientierten Informatik-Ausbildung an Hochschulen, Hochschulmagazin UNIKATH, Karlsruhe, 2004.

[EA+04] M. Endrei, J. Ang, A. Arsanjani, S. Chua, P. Comte, P. Krogdahl, M. Luo, T. Newling: Patterns - Service-Oriented Architecture and Web Services, IBM Redbooks, April 2004.

[EB+07a] C. Emig, F. Brandt, S. Abeck, J. Biermann, H. Klarl: An Access Control Metamodel for Web Service-Oriented Architecture, IEEE Conference on Software Engineering Advances (ICSEA'07), Cap Esterel, August 2007.

[EB+07b] C. Emig, F. Brandt, S. Kreuzer, S. Abeck: Identity as a Service - Towards a Service-Oriented Identity Management Architecture, 13th EUNICE Open European Summer School and IFIP TC6.6 Workshop on Dependable and Adaptable Networks and Services (EUNICE'07), Twente, Juli 2007.

[Ec06] C. Eckert: IT-Sicherheit, Konzepte – Verfahren – Protokolle, Oldenbourg-Verlag, 4. Auflage, 2006.

[EK+07] C. Emig, K. Krutz, S. Link, C. Momm, S. Abeck: Model-Driven Development of SOA Services, Forschungsbericht, Universität Karlsruhe (TH), April 2007.

[EK+08] C. Emig, S. Kreuzer, S. Abeck, J. Biermann, H. Klarl: Model-Driven Development of Access Control Policies for Web Services, Forschungsbericht, Universität Karlsruhe (TH), 2008.

[EL+06] C. Emig, K. Langer, K. Krutz, S. Link, C. Momm, S. Abeck: The SOA's Layers, Forschungsbericht, Universität Karlsruhe (TH), 2006.

[EL+07] C. Emig, K. Langer, J. Biermann, S. Abeck: Semantic Integration of Identity Data Repositories, Kommunikation in verteilten Systemen (KiVS), 15. GI/ITG-Fachtagung, Bern, Februar 2007.

[Em07] C. Emig: Identity as a Service in Service-Oriented Architecture, Vortrag auf der European Identity Conference, München, Mai 2007.

[EM+05] C. Emig, C. Momm, J. Weißer, S. Abeck: Programming in the Large based on the Business Process Modeling Notation, Jahrestagung der Gesellschaft für Informatik (GI), Bonn, September 2005.

[EM+06] M. El Kharbili, T. Mathes, P. Perera, H. Schandua, T. Stiller, K. Langer, C. Emig, S. Abeck: Werkstatt Unternehmenssoftware Karlsruhe (WUSKAR): User Provisioning Processes in Identity Management addressing SAP Campus Management, Forschungsbericht, Universität Karlsruhe (TH), 2006.

[ER02] T. Erler, M. Ricken: UML, Verlag Moderne Industrie, Bonn, 2002.

[Er05] T. Erl: Service-Oriented Architecture: Concepts, Technology and Design, Prentice Hall, 2005.

[ES03] K. Eilebrecht, G. Starke: Patterns kompakt. Entwurfsmuster für effektive Software-Entwicklung, Spektrum Akademischer Verlag, 2003.

[ES+06] C. Emig, H. Schandua, S. Abeck: SOA-aware Authorization Control, IEEE Conference Software Engineering Advances (ICSEA'06), Tahiti, November 2006.

[EW+06] C. Emig, J. Weißer, S. Abeck: Development of SOA-Based Software Systems - an Evolutionary Programming Approach, IEEE Conference on Internet and Web Applications and Services (ICIW'06), Guadeloupe, Februar 2006.

[Fe04] D. Feuerhelm: Metaoperationen und Metadaten im Internet-basierten Wissenstransfer, Dissertation, Verlag dissertation.de, Berlin, 2004.

[Ge05] G. Gebel: Virtual Directory Services, Burton Group Identity and Privacy Strategies, 2005.

[GH+95] E. Gamma, R. Helm, R. Johnson and J. Vlissides: Entwurfsmuster : Elemente wiederverwendbarer objektorientierter Software, Addison-Wesley Publishing Company, 1995.

[GI-IB] Gesellschaft für Informatik: Informatik-Begriffsnetz, 2008.
URL: http://www.tfh-berlin.de/~giak

[GI-SWA] Gesellschaft für Informatik: Webseite der Fachgruppe Software-Architektur, 2008.
URL: http://sdq.ipd.uka.de/research/fgswarch

[GK04] G. Gamm, S. Körnig: Transparente Schichten – Perspektiven der Informatisierung des Wissens, Proceedings of the First International Workshop on Philosophy and Informatics, Köln, März 2004.

[GK+03] K. Geihs, R. Kalckloesch, A. Grode: Single Sign-On in Service-Oriented Computing, ICSOC 2003, LNCS 2910, 2003.

[GN07] G. Gebel, M. Neuenschwander: User Authorization, Burton Group Identity and Privacy Strategies, März 2007.

[Go97] C.H. Goh. Representing and Reasoning about Semantic Conflicts in Heterogeneous Information Sources, MIT, 1997.

[Gr05] P. Grund-Ludwig: Identitätsmanagement ist Voraussetzung für den Aufbau serviceorientierter Architekturen, Computer Zeitung Nr. 42, Oktober 2005.

[Gr93] T. Gruber: A translation approach to portable ontology specifications, Knowledge Acquisition, Volume 5, Issue 2, Juni 1993.

[HA+99] H.-G. Hegering, S. Abeck, B. Neumair: Integriertes Management vernetzter Systeme – Konzepte, Architekturen und deren betrieblicher Einsatz, dpunkt.verlag, Heidelberg, 1999.

[HB04] H. Haas, A. Brown: Web Services Glossary, W3C Working Group Note, Februar 2004.
URL: http://www.w3.org/TR/ws-gloss

[He06] M. Henning: The Rise and Fall of CORBA, ACM Queue Magazine „Component Technologies“, Vol. 4, No. 5, Juni 2006.

[HG00] F. Hakimpour, A. Geppert: Resolving Semantic Heterogeneity in Schema Integration – An Ontology Based Approach, Department of Geography, Universität Zürich, 2000.

[HH07] T. Hasen, M. Herren: SOAP Firewall, IT Security Journal, 2007.

[HH+06] A. Hess, B. Humm, M. Voß: Regeln für serviceorientierte Architekturen hoher Qualität, in Informatik Spektrum, 29(6), Springer-Verlag, Dezember 2006.

[HM+06] T. Höllrigl, A. Maurer, F. Schell, H. Wenske, H. Hartenstein: Dienstorientiertes Identitätsmanagement für eine Pervasive University, Jahrestagung der Gesellschaft für Informatik (GI), Dresden, September 2006.

[Ho07] W. Hommel: Architektur- und Werkzeugkonzepte für föderiertes Identitäts-Management, Dissertation, Verlag Dr. Hut, München, 2007.

[HP-JENA] Hewlett Packard Development Company: JENA – A Semantic Web Framework for Java, 2005.
URL: http://jena.sourceforge.net

[HR-XML] Webseite des HR-XML-Konsortiums
URL: http://www.hr-xml.org

[IBM-RSA] IBM-Produktwebseite: Rational Software Architect (RSA).
URL: http://www-306.ibm.com/software/awdtools/swarchitect

[IBM-TDI] IBM-Redbook: Robust Data Synchronization With IBM Tivoli Directory Integrator, IBM Corp., 2006.

[IBM-WAS] IBM-Produktwebseite: WebSphere Application Server (WAS).
URL: http://www-306.ibm.com/software/webservers

[IBM-WID] IBM-Produktwebseite: WebSphere Integration Developer (WID).
URL: http://www-306.ibm.com/software/integration/wid

[IBM-WPS] IBM- Produktwebseite: WebSphere Process Server (WPS).
URL: http://www-306.ibm.com/software/integration/wps

[ISO89] International Organization for Standardization: ISO 7498-2 Information Processing Systems - Open Systems Interconnection - Basic Reference Model - Part 2: Security Architecture, 1989.

[ITU-X.812] ITU X.812: Information technology - Open Systems Interconnection - Security frameworks for open systems: Access control framework, 1996.

[JB+99] I. Jacobson, G. Booch, J. Rumbaugh: The unified software development process, Addison-Wesley, 1999.

[JH94] K.M. Jackson, J. Hruska: Computer Security Reference Book, Butterworth-Heinemann-Verlag, Oxford, 1994.

[JS03] G. Jahn, R. Stamms: Identity Management und Zentraler Verzeichnisdienst - Eine Infrastruktur für ein deutlich verbessertes Informationsmanagement für Hochschulen, IBM Business Consulting, 2003.

[Ka03] G. Karjoth: Access Control with IBM Tivoli Access Manager, ACM Transactions on Information and System Security, Vol. 6, No. 2, Mai 2003.

[KB+05] D. Krafzig, K. Banke, D. Slama: Enterprise SOA, Prentice Hall, 2005.

[Ke02] W. Keller: Enterprise Application Integration, dpunkt.verlag, Heidelberg, 2002

[Ke07] <kes> – Die Zeitschrift für Informations-Sicherheit: Lexikon der Informationssicherheit, 2007.
URL: http://www.kes.info

[KL04] D. Kossmann, F. Leymann: Web Services, Informatik Spektrum, Band 27, 117-128, Springer-Verlag, 2004.

[Kl06] M. Klein: Automatisierung dienstorientierten Rechnens durch semantische Dienstbeschreibungen, Dissertation, Universitätsverlag Karlsruhe, 2006.

[Ko04] J. Kobielus: Service-Oriented Architecture: Developing the Enterprise Roadmap, Burton Group Application Platform Strategies, Februar 2004.

[Ko06] N. Koch: Transformations Techniques in the Model-Driven Development Process of UWE, Model-Driven Web Engineering Workshop (MDWE 2006), Palo Alto, USA, Juli 2006.

[Kr07a] S. Kreuzer: Model-driven Development of Access Control Policies within Web Service-based Architecture, Diplomarbeit, Universität Karlsruhe (TH), 2007.

[Kr07b] K. Krutz: Integriertes Modell zur Entwicklung und Suche von Web-Diensten, Dissertation, Verlag dissertation.de, Berlin, 2007.

[KS96] V. Kashyap, A. Sheth: Schematic and semantic semilarities between database objects: A context-based approach, International Journal on Very Large Data Bases, 5(4):276–304, 1996.

[Ku05] M. Kuppinger: Identity Management – Basis für sichere Geschäftsprozesse, 2005.

[KW+08] H. Klarl, C. Wolff, C. Emig: Abbildung von Zugriffskontroll-Aussagen in Geschäftsprozessmodellen, Modellierung 2008, Konferenz des GI Querschnittsfachausschusses der Modellierungsfachgruppen, Berlin, März 2008.

[La06] K. Langer: Ontology-Based Integration of Directories for Identity Management, Diplomarbeit, Universität Karlsruhe (TH), 2006.

[La69] B.W. Lampson: Dynamic protection structures, AFIPS conference proceedings, Fall Joint Computer Conference, 1969.

[Le03] F. Leymann: Web Services - Distributed Applications without Limits, Business, Technology and Web, Leipzig, 2003.

[Le98] J. Lewis: Enterprise Directory Infrastructure - Meta-Directory Concepts and Functions, Burton Group Identity and Privacy Strategies, Juli 2004.

[Li03] D.S. Linthicum: Next Generation Application Integration - From simple Information to Web Services, Addison Wesley, Dezember 2003.

[Lo03] T. Lodderstedt: Model-Driven Security, Dissertation, Universität Freiburg, 2003.
URN (NBN): urn:nbn:de:bsz:25-opus-12532

[LS99] E.C. Lupu, M. Sloman: Conflicts in Policy-Based Distributed Systems Management, IEEE Transactions on Software Engineering, Vol. 25, No. 6, pp. 852-869, November 1999.

[Ma01] C. Mayerl: Eine integrierte Dienstmanagement-Architektur für die qualitätsgesicherte Bereitstellung von Netz- und Systemdiensten, Dissertation, Shaker-Verlag, Aachen, 2001.

[Me07] O. Mehl: Modellgetriebene Entwicklung managementfähiger Anwendungssysteme, Dissertation, Verlag dissertation.de, Berlin, 2007.

[ME+05] M. Manz, C. Emig, S. Abeck: Werkstatt Unternehmenssoftware Karlsruhe (WUSKAR): Synchronization of Directory Services with the Event Propagation Framework, Forschungsbericht, Universität Karlsruhe (TH), 2005.

[MP04] W. Martin, U. Parthier: Das prozeßorientierte Unternehmen, Strategic Bulletin EAI 2004, Annecy/Sauerlach, 2004.

[MS-DCOM] Microsoft Corp. - Distributed Component Object Model (DCOM), Juli 1997
URL: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnanchor/html/dcom.asp

[Ni03] N. Nikols: Meta-Directory Services, Burton Group Identity and Privacy Strategies, 2003.

[Ni04] N. Nikols: Directory Products evolve towards Identity Services, Burton Group Identity and Privacy Strategies, November 2004.

[Ni06] N. Nikols: Enabling Identity Data Services, Burton Group Identity and Privacy Strategies, März 2006.

[NM01] N.F. Noy, D.L. McGuinness: Ontology Development 101: A Guide to Creating Your First Ontology, Stanford University, Stanford, CA, 94305, 2001.

[No05] Z. Nochta: Zertifikatsbasierte Zugriffskontrolle in verteilten Informationssystemen, Dissertation, Universitätsverlag Karlsruhe, 2005.

[OASIS-SAML2.0] OASIS: Security Assertion Markup Language (SAML) 2.0, 2005.
URL: http://www.oasis-open.org/specs/index.php#samlv2.0

[OASIS-UDDI3.0] OASIS: Universal Description, Discovery and Integration (UDDI) 3.0.2, Februar 2005.
URL: http://www.oasis-open.org/committees/uddi-spec

[OASIS-WS-Sec1.1] A. Nadalin, C. Kaler, R. Monzillo, P. Hallam-Baker (Hrsg.): Web Services Security (WS-Security), Version 1.1, Februar 2006.

[OASIS-WST1.3] A. Nadalin, M. Goodner, M. Gudgin, A. Barbir, N. H. Granqvist (Hrsg): WS-Trust 1.3, Dezember 2006.

[OASIS-XACML2.0] T. Moses (Hrsg): eXtensible Access Control Markup Language (XACML) Version 2.0, Februar 2005.
URL: http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml#XACML20

[Oe04] B. Oestereich: Die UML 2.0 Kurzreferenz für die Praxis, Oldenburger Wissenschaftsverlag, München, 2004.

[OMG-BPMN] Business Process Management Initiative (BPMI): Business Process Modeling Notation (BPMN), Version 1.0, BPMI.org, Mai 2004.

[OMG-CORBA] Object Management Group: Common Object Request Broker Architecture (CORBA) 3.0.3, März 2004.

[OMG-MDA] Object Management Group: Model Driven Architecture. URL: http://www.omg.org/mda

[OMG-MDA-Guide] Object Management Group: MDA Guide Version 1.0.1, Juni 2003.

[OMG-MOF] Object Management Group: Meta Object Facility (MOF) Specification, Juli 2005.

[OMG-OCL2] Object Management Group: Object Constraint Language, Version 2.0, Mai 2006.

[OMG-UML2-Infra] Object Management Group: Unified Modeling Language: Infrastructure, Version 2.1.1, Februar 2007.

[OMG-UML2-Super] Object Management Group: Unified Modeling Language: Superstructure, Version 2.1.1, Februar 2007.

[OR03] M. Owen, J. Ray: BPMN and Business Process Management – Introduction to the New Business Modeling Standard, Popkin Software, 2003.

[Os05] U. Ostler: Safety first in Service-orientierten Architekturen, silicon.de Portal, September 2005.

[OW+03] B. Oestereich, C. Weiss, C. Schroeder, T. Weilkiens, A. Lenhard: Objektorientierte Geschäftsprozessmodellierung mit der UML, dpunkt.verlag, Heidelberg, 2003.

[Pa02] C. Partridge: The Role of Ontology in Semantic Integration, OOPSLA 2002, Seattle.

[PM03] A. Pashalidis, C.J. Mitchell: A Taxonomy of Single Sign-On Systems, Springer-Verlag, 2003.

[PM06] R. Petrasch, O. Meimberg: Model Driven Architecture, dpunkt.verlag, Heidelberg, 2006.

[Ra02a] J. Ramachandran: Designing Security Architecture Solutions, Wiley Computer Publishing, 2002.

[Ra02b] E. Ray: Einführung in XML, O'Reilly-Verlag, 2002.

[RACER] Racer DIG Reasoner, Juli 2006.
URL: http://www.racer-systems.com/de/index.phtml
URL: http://dig.sourceforge.net

[Re05] M. Reiter: Infrastruktur für die Serviceorientierung steht, Computer Zeitung Nr. 19, Mai 2005.

[Re06] M. Reiter: Webservices lösen Corba langfristig ab, Computer Zeitung Nr. 48, November 2006.

[RFC-2045] The Internet Engineering Task Force: RFC 2045, Multipurpose Internet Mail Extensions, (MIME) Part One: Format of Internet Message Bodies, 1996.

[RFC-2251] The Internet Engineering Task Force: RFC 2251, Lightweight Directory Access Protocol (v3), 1997.

[RFC-2753] The Internet Engineering Task Force: RFC 2753, A Framework for Policy-based Admission Control, 2000.

[RFC-2849] The Internet Engineering Task Force: RFC 2849, The LDAP Data Interchange Format (LDIF) – Technical Specification, 1997.

[RH06] R. Reussner, W. Hasselbring: Handbuch der Software-Architektur, dpunkt.verlag, Heidelberg, 2006.

[RH+05] J.-P. Richter, H. Haller, P. Schrey: Serviceorientierte Architektur, Informatik Spektrum im Springer-Verlag, Oktober 2005.

[Ro03] G. Dreo-Rodosek: A Generic Model for IT Services and Service Management, In Integrated Network Management VIII — Managing It All, 2003, Kluwer Academic Publishers, IFIP/IEEE, Colorado Springs, USA, März, 2003.

[Sa01] R. Sandhu: Future Directions in Role-Based Access Control Models, MMM-ACNS 2001, LNCS 2052, pp. 22-26, 2001.

[Sa93] R. Sandhu: Lattice-Based Access Control Models, IEEE Computer 26 (11), 1993.

[SB+98] R. Studer, V. R. Benjamins, D. Fensel: Knowledge Engineering - Principles and Methods, Data Knowledge Engineering, Volume 25, No 1-2, 1998.

[Sc02] A.-W. Scheer: ARIS - Vom Geschäftsprozess zum Anwendungssystem. 4. Auflage, Springer-Verlag, Berlin, 2002.

[Sc06] H. Schandua: Anpassung bestehender Identitätsmanagement-Lösungen an serviceorientierte Architekturen, Diplomarbeit, Universität Karlsruhe (TH), 2006.

[SC+96] R. Sandhu, E.J. Coyne, H.L. Feinstein, C.E. Youman: Role-Based Access Control Models, IEEE Computer 29(2): 38-47, IEEE Press, 1996.

[SMI05] Stanford Medical Informatics: Protégé Ontology Editor, 2005. URL: http://protege.stanford.edu

[SN+05] C. Steel, R. Naggapan, R. Lai: Core Security Patterns, Prentice Hall, Oktober 2005.

[Sp00] G. Sparks: An Introduction to UML - The Business Process Model, Sparx Systems, 2000.

[SS04] S. Staab, R. Studer [Hrsg.]: Handbook on Ontologies, International Handbooks on Information Systems, Springer-Verlag, 2004.

[SS94] R.S. Sandhu, P. Samarati: Access Control - Principles and Practice, IEEE Communications, September 1994.

[St05] H. Störrle: UML 2 erfolgreich einsetzen, Addison-Wesley Verlag, München, 2005.

[St06a] R. Stalder: Die Zukunft der Softwareentwicklung, in: Blickpunkt KMU 2/2006, März 2006.

[St06b] T. Stiller: Single Sign-On in serviceorientierten Architekturen, Diplomarbeit, Universität Karlsruhe (TH), 2006.

[St73] H. Stachowiak: Allgemeine Modelltheorie, Springer-Verlag, Wien, 1973.

[SU06] Standord University: Knowledge Systems Laboratory Ontology Editor, Juni 2006.
URL: http://www-ksl-svc.stanford.edu:5915

[SUN-Jini] Sun Microsystems – Jini Network Technology v2.1, Oktober 2005.
URL: http://www.sun.com/software/jini

[SUN-NI] Sun Microsystems: Network Identity – Unlocking the Values of Web Services, 2003.

[SUN-RCL] Sun: Retro Change Log Plug-In.
URL: http://docs.sun.com/source/816-6698-10/replicat.html

[SV+05] T. Stahl, M. Völter: Modellgetriebene Softwareentwicklung, 1. Auflage, dpunkt.verlag, Heidelberg, 2005.

[Ta05] H. Tao: A XACML-based Access Control Model for Web Service, IEEE Conference on Wireless Communications, Networking and Mobile Computing, 2005.

[TE+04] S. Tuttle, A. Ehlenberger, R. Gorthi, J. Leiserson, R. Macbeth, N. Owen, S. Ranahandola, M. Storrs, C. Yang: Understanding LDAP Design and Implementation, IBM Redbooks, 2004.

[TS07] G. Starke, S. Tilkov: SOA-Expertenwissen, dpunkt.verlag, Heidelberg, Juni 2007.

[UKA-KIM] Universität Karlsruhe: Karlsruher Integriertes Informations-Management – Webseite des Projekts, 2008.
URL: http://www.kim.uni-karlsruhe.de

[UMBC-PO] UMBC Ebiquity Research Group: Person Ontology.
URL: http://ebiquity.umbc.edu/ontology/person.owl

[VS+05] M. Völter, T. Stahl, J. Bettin, A. Haase, S. Helsen: Model-Driven Software Development, Wiley-Verlag, Mai 2007.

[W3C-OWL] W3C Recommendation: OWL Web Ontology Language Overview, Februar 2004.

[W3C-RDF] W3C Recommendation: Resource Description Framework (RDF), Februar 2004.

[W3C-RDFS] W3C Recommendation: RDF Vocabluary Description Language 1.0: RDF Schema, Februar 2004.

[W3C-SOAP1.2] W3C Recommendation: SOAP Version 1.2, April 2007.
URL: http://www.w3.org/TR/soap

[W3C-XML-Enc] W3C: XML-Encryption Syntax and Processing, Februar 2002.
URL: http://www.w3.org/TR/xmlenc-core

[W3C-XML-Sig] W3C: XML-Signature Syntax and Processing, Februar 2002.
URL: http://www.w3.org/TR/xmldsig-core

[W3C-WSDL1.1] W3C: Web Services Description Language (WSDL), Version 1.1, Mai 2001.
URL: http://www.w3.org/TR/wsdl

[W3C-WSDL2.0] W3C: Web Services Description Language (WSDL), Version 2.0, Juni 2007.
URL: http://www.w3.org/TR/wsdl20

[W3C-WSP1.5] W3C: Web Services Policy Framework, Version 1.5, September 2007.
URL: http://www.w3.org/TR/ws-policy

[Wa04] H. Walther: The actual status of Identity Management, IM Information Management & Consulting, Heft 19, Mai 2004.

[WA05] X. Wang, J.S. Almeida: DLG2 - A Graphical Presentation Language for RDF and OWL, 2005.
URL: http://charlestoncore.musc.edu/docs/dlg2.html

[We05] J. Weißer: University SOA – Building a Transcript of Records Service, Studienarbeit, Universität Karlsruhe (TH), 2005.

[Wh04] S.A. White: Introduction to BPMN, IBM Cooperation, 2004.

[Wi05] P. Windly: Digital Identity, O'Reilly-Verlag, 2005.

[WK03] J.B. Warmer, A. Kleppe: The Object Constraint Language, Pearson Education, Boston, 2003.

[Wi95] R. Wies: Policies in Integrated Network and Systems Management, Dissertation, Shaker Verlag, Aachen, 1995.

[WSFED1.1] H. Lockhart, S. Andersen et. al.: Web Services Federation Language (WS-Federation), Dezember 2006.

[WSSP1.1] G. Della-Libera, M. Gudgin et. al: Web Services Security Policy Language (WS-SecurityPolicy), Juli 2005.

[WV+01] H. Wache, T. Vögele, U. Visser, H. Stuckenschmidt, G. Schuster, H. Neumann, S. Hübner: Ontology-Based Integration of Information - A Survey of Existing Approaches, Intelligent Systems Group, Center for Computing Technologies, Universität Bremen, 2001.

[YT05] E. Yuan, J. Tong: Attribute Based Access Control (ABAC) for Web Services, IEEE International Conference on Web Services (ICWS'05), Orlando, Juli 2005.

[Zi80] H. Zimmermann: OSI Reference Model – The ISO Model of Architecture for Open Systems Interconnection, IEEE Transactions on Communications, Vol. COM-28, No. 4, April 1980.

[Zö05] S. Zörner: LDAP für Java-Entwickler, entwickler.press Software & Support Verlag, 2005.

[ZT+05] O. Zimmermann, M. Tomlinson, S. Peuser: Perspectives on Web Services, Springer-Verlag, 2. Auflage, 2005.

Die angegebenen URLs wurden am 06. Mai 2008 auf Gültigkeit geprüft.