Hochschule Mittweida (FH)
University of Applied Sciences

DIPLOMARBEIT

Stanislav Bilik

Konzeption und Realisierung eines Backend-Systems für ein Dokumenten-Management-System auf Basis der serviceorientierten Architektur

2010

Hochschule Mittweida (FH)
University of Applied Sciences
Fachbereich Informationstechnik & Elektrotechnik

DIPLOMARBEIT

Konzeption und Realisierung eines Backend-Systems für ein Dokumenten-Management-System auf Basis der serviceorientierten Architektur

Autor: Stanislav Bilik

Studiengang: Multimediatechnik
Seminargruppe: MK05w1

Erstprüfer: Prof. Dr.-Ing. Frank Zimmer
Zweitprüfer: Dipl.-Wi.Inf. Ron Kuhnert

Mittweida, Januar 2010

Bibliografische Angaben

Bilik, Stanislav:
Konzeption und Realisierung eines Backend-Systems für ein Dokumenten-Management-System auf Basis der serviceorientierten Architektur
83 Seiten, 34 Abbildungen und 4 Anlagen mit Quelltexten,
Hochschule Mittweida (FH), Fachbereich Informationstechnik & Elektrotechnik
Diplomarbeit, 2010

Referat:

Ziel der Diplomarbeit ist es, ein serviceorientiertes Backend- System für ein Dokumenten-Management-System auf der Basis von open-EIS Plattform zu entwerfen und zu implementieren. Am Anfang werden die erforderlichen Grundlagen behandelt, worauf sich dann die Analyse der Anforderungen stützt und den Ausgangspunkt für den Entwurf definiert. Mit dem Einsatz von geeigneten Mitteln wird dieser Entwurf realisiert und die entsprechende Implementierung umgesetzt. Zum Schluss wird das komplette Backend- System den notwendigen Testabläufen unterworfen und eine Auswertung der Gesamtarbeit durchgeführt.

Danksagung

Ich möchte mich bei allen bedanken, die mir während der Erstellung dieser Arbeit geholfen haben. Zuerst richtet sich der Dank an meinen Professor Frank Zimmer für die tatkräftige Unterstützung bei der Anfertigung meiner Diplomarbeit sowie die überaus freundliche Bekanntschaft während des gesamten Studiums. Vielen Dank für die hilfreichen Anregungen und die Toleranz. Ebenfalls schulde ich ein herzliches Dankeschön meinem betrieblichen Betreuer Ron Kuhnert für sein kompetentes Engagement, welcher mir in dieser Phase zur Seite gestanden hat. Ich danke den Geschäftsführern, Herrn Bauch und Herrn Nebel, des Unternehmens community4you GmbH für sehr spannende und interessante Zeit, die ich sowohl während des Studiums als auch zu seinem Abschluss verbracht habe. Nicht zuletzt möchte ich meinen Eltern danken, die mich immer moralisch unterstützt und mir den Rücken gestärkt haben. Im Besonderen danke ich meinem Vater Juri, der stets mit einer enthusiastischen Anteilnahme die Fortschritte meiner Diplomarbeit verfolgte. Ein liebevoller Dank geht auch an meine Frau Nataliya, die in den guten wie auch in den schlechten Zeiten zu mir gestanden hat.

Inhaltsverzeichnis

I. Abbildungsverzeichnis

II. Abkürzungsverzeichnis

API	Application Programming Interface
CVS	Concurrent Versions System
DDL	Data Definition Language
DDM	Dynamic Data Model
DML	Data Manipulation Language
DMS	Dokumenten-Management-System
DO	Data Object
DRM	Digital Rights Management
HTTP	Hypertext Transfer Protocol
JDBC	Java Database Connectivity
JPEG	Joint Photographic Experts Group
JSP	JavaServer Pages
JVM	Java Virtual Machine
MDA	Model Driven Architecture
ODBC	Open Database Connectivity
PDF	Portable Document Format
RMI	Remote Method Invocation
POJO	Plain Old Java Object
QoS	Quality of Service
RCP	Rich Client Platform
REST	Representational State Transfer
RPC	Remote Procedure Call
SAML	Security Assertion Markup Language
SLA	Service Level Agreement
SOA	serviceorientierte Architektur
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
SWT	Standard Widget Toolkit
UDDI	Universal Description, Discovery and Integration
UI	User Interface
UML	Unified Modeling Language
URI	Uniform Resource Identifier

URL	Uniform Resource Locator
WS	Webservice
WSIL	Web Services Inspection Language
WSDL	Web Service Description Language
XML	Extensible Markup Language
bzw.	beziehungsweise
deut.	deutsch
etc.	und so weiter (lat. et cetera)
evtl.	eventuell
o.b.	oben beschrieben
o.e.	oben erwähnt
s.	siehe
s.g.	so genannt
u.a.	unter anderem
vgl.	vergleiche
z.B.	zum Beispiel

1. Einleitung

„Wir ertrinken in Informationen, aber hungern nach Wissen.“
(John Naisbitt)

Mit der industriellen Herstellung der *Personal Computer* Ende der 70er bzw. Anfang der 80er Jahre des 20. Jahrhunderts und deren Einsatz an vielen Arbeitsplätzen hat sich die elektronische Datenverarbeitung mit rasender Geschwindigkeit verbreitet. Die Rechner wurden damals in den Unternehmen oft für die Erstellung von Dokumenten eingesetzt, wobei diese bei der Fertigstellung meistens nur ausgedruckt wurden. Es war eine Umstellung von den analogen Schreibmaschinen auf digitale Rechner. Durch die später veränderte IT- Infrastruktur ließen sich Rechner miteinander vernetzen, was zur Folge einer Zentralisierung der Dokumente führte. Die klassischen Dokumentenablagen werden zunehmend durch die rechnergestützten Ablagen ersetzt. Infolge der ständig wachsenden Informationsmengen erhöht sich die Nachfrage nach einer zentralen Verwaltungs- und Archivierungsstelle der Dokumente für eine oder sogar mehrere Institutionen, denn Wissen bedeutet Vorsprung und Sicherheit. Durch den verstärkten Wettbewerb in letzten Jahren sind das schnelle Erkennen von Problemen und Wünschen der Kunden sowie das Reagieren auf die Marktveränderungen überlebenswichtig geworden. Doch nicht das Wissen allein, sondern der zeit- und ortunabhängiger Zugriff auf die Informationen ist entscheidend. Beim Einsatz eines Dokumenten-Management-Systems in einer vernetzten IT- Infrastruktur liegen seine Vorteile im gemeinsamen Zugriff für jeden Berechtigten, jederzeit, auf stets aktuelle und vollständige Informationsmengen. An diese Denkweise anknüpfend möchte ich mich in dieser Diplomarbeit mit der Thematik eines Systems für die Verwaltung von Dokumenten auseinandersetzen.

1.1. Motivation

In vielen großen Unternehmen lässt sich seit einiger Zeit ein Trend beobachten, bei welchem die Informationstechnik des Unternehmens auf serviceorientierte Architekturen umgestaltet wird. [vgl. be10] Die community4you GmbH verfolgt auch den Gesichtspunkt des dezentralen, servicegetriebenes

Architekturkonzepts und bietet ihrerseits eine Lösung, open-EIS, in Form eines Enterprise Informationssystems für Wissensmanagement, eLearning und Kommunikation dar. Dieses System basiert auf einer flexiblen und mehrschichtigen serviceorientierten Architektur zum effizienten Aufbau moderner Anwendungen. Diese Architektur bildet die Basis für die Erstellung individueller Lösungen mit fachlich getrennten Anwendungselementen. Somit wird eine schnellere Reaktion auf veränderte Geschäftsprozesse und mehr Flexibilität bei den Systemen erreicht. Die community4you GmbH besitzt ein Dokumenten- Management-System, welches jedoch nicht auf der SOA basiert. Aus diesem Grund wird das Ziel dieser Arbeit sein, einen prototypischen Entwurf eines serviceorientierten Backend-Systems für die Verwaltung von Dokumenten auf der Basis von open-EIS zu konzipieren und diesen anschließend zu implementieren.

1.2. Überblick

Im nachfolgenden Kapitel werden einige Grundlagen zum Thema Dokumenten-Management-System (DMS) erläutert. Dabei handelt es sich um die Definition des Begriffes DMS, den Funktionsumfang sowie den prinzipiellen Aufbau des Systems. Weiterhin werden die Vorteile und die Nachteile eines DMS angesprochen und die möglichen Systemarchitekturen näher beschrieben. Ein weiterer umfangreicher Bereich dieses Kapitels widmet sich der serviceorientierten Architektur. Anschließend wird die open-EIS Plattform vorgestellt. Das Kapitel 3 befasst sich mit der Anforderungsanalyse an das Backend-System und setzt die Rahmenbedingungen für diese Arbeit fest. In dem Kapitel werden detailliert die Anforderungen mit Hilfe von geeigneten Diagrammen beschrieben. Nachdem die Analyse der Anforderungen an das zu entwickelnde System definiert wurde, wird im Kapitel 4 ein Entwurf für die Umsetzung eines serviceorientierten Backend-Systems konzipiert. Das Kapitel 5 verwendet die Ergebnisse aus dem Entwurf und beschreibt die Phase der Implementierung des benötigten Systems. Nach der Fertigstellung der Implementierung wird im Kapitel 6 dargestellt, wie und mit welchen Mitteln diese Implementierung getestet wurde. Zuletzt wird ein Resümee über den Stand der Arbeit gezogen.

2. Grundlagen

In dem Kapitel Grundlagen werden die Begriffsdefinitionen und die Vorteile des Einsatzes eines Dokumenten-Management-Systems beschrieben. Weiterhin beschäftigt sich dieses Kapitel mit den unterschiedlichen Architekturarten und Funktionen sowie dem prinzipiellen Aufbau dieser Systeme. Ferner wird das Prinzip der serviceorientierten Architektur sowie die darauf basierende open-EIS Plattform dargestellt, welche die Grundlage für diese Arbeit liefern. Zum Schluss des Kapitels werden einige Vorgehensmodelle in der Softwareentwicklung näher behandelt, welche die Planung und Durchführung der Projekte erleichtern.

2.1. Dokumenten-Management-System

Das Dokumenten-Management-System stellt einen großen Schwerpunkt dieser Arbeit dar. Aus diesem Grund ist es notwendig, dieses Thema umfassend zu betrachten. Nachfolgend werden die mit dieser Thematik zusammenhängenden Begrifflichkeiten geklärt.

2.1.1. Definitionen

Ein Dokument ist in der Lage Information aller Art zusammenzuführen und zu einem bestimmten Zeitpunkt festzuhalten, unabhängig davon ob es eine Rechnung, ein Vertrag, ein Videomaterial oder eine einfache URL-Adresse ist. Im Angesicht der vielen unterschiedlichen Dokumenten lassen sich jedoch zwei Grundtypen in Bezug auf den Zweck eines Dokumentes herausbilden. Zum einen soll ein Dokument als Nachweis einer Tatsache und zum anderen als Träger von Informationen dienen. Es gibt aber auch eine Reihe von Dokumenten, welche beiden Zwecken gleichzeitig dienen. [vgl. Ka99] S. 27-28

Für die Verwaltung von Dokumenten in einem DMS muss ein Dokument bestimmten Konventionen für dessen Beschreibung folgen [vgl. Go04] S. 34:

- Das eigentliche **Dokument**: wird gespeichert, falls notwendig vorher noch in ein anderes Format konvertiert.
- Der eindeutige **Schlüssel** (Key) ist für jedes Dokument notwendig, um es wieder finden zu können.

- **Metadaten** eines Dokumentes beschreiben es und helfen bei der Suche.
- **Strukturinformationen** beschreiben den Aufbau des Dokumentes.
- **Regeln** legen die Lebensdauer und Lebenszyklus, sowie Zugriffsrechte fest

Nach GÖTZER handelt es sich bei Dokumenten-Management-Systemen um „… die Logik der Verwaltung von Dokumenten, deren Status, Struktur, Lebenszyklus und Inhalt. Dokumente werden beschrieben, klassifiziert und in einer bestimmten logischen Struktur eingeordnet, damit sie einfach wieder gefunden werden können.“ [Go04] S. 5 Das Management von Dokumenten beschäftigt sich somit im Laufe des gesamten Dokumentenlebenszyklus mit der Erstellung und Bearbeitung, Kontrolle der Zugriffsrechte, Publikation, Verteilung und der Vernichtung von Dokumenten. KRÄNZLE definiert ein DMS als eine Anwendung mit einem objektorientierten Ansatz folgenderweise: „… DMS - ist eine Software, die der aufgabengerechten Erzeugung, Bereitstellung, Steuerung, Weiterleitung und Archivierung von Dokumenten im Rahmen von organisatorischen Prozessen dient. Dokumente sind dabei alle informatorischen Objekte - seien sie auf Papier oder als elektronische Objekte wie Dateien, Verzeichnisse oder zusammengesetzte Objektstrukturen - die Informationen für die jeweiligen betrieblichen Prozesse zur Verfügung stellen.“ [Kr95] S. 27 Unter dem zu entwickelnden Backend für ein DMS wird auch eine objektorientierte Anwendung verstanden. Dieses Backend- System umfasst die Anwendungsbereiche, welche sich mit der Geschäftslogik und der Datenhaltung beschäftigen. Das folgende Kapitel beschreibt die Funktionen eines DMS, welche für die Verwaltung von Dokumenten verwendet werden.

2.1.2. Funktionen

In einer abstrakten Sicht auf die Funktonalitäten für die Dokumentenverwaltung eines Systems spricht man im Sinne von RIGGERT von den sieben *V*: Verarbeiten, Verwahren, Verhindern, Verfügen, Verändern, Verwalten und Vernichten. Darunter soll verstanden werden, dass ein Dokument erstellt, archiviert und von unbefugtem Zugriff geschützt wird. Zu einem späteren Zeitpunkt soll

dieses Dokument wieder gefunden und bearbeitet oder auch bei Nichtverwendung aus dem System entfernt werden. [vgl. ri09] S. 28

2.1.2.1. Allgemeine Funktionen

Die Dokumenten-Management-Systeme werden für unterschiedliche Zwecke mit unterschiedlichsten innerbetrieblichen Abläufen in den Unternehmen eingesetzt. Aus diesem Grund gibt es nicht *ein* System mit allen erdenklichen Funktionalitäten, sondern viele Systeme mit den angepassten Arbeitsabläufen für verschiedene Einsatzgebiete. Alle diese Systeme verwenden allerdings im Kern die typischen Funktionen, welche nachfolgend zusammengefasst wurden. [vgl. Go04] S. 36-37

- **Verwaltung**: von Dokumenten und ihren Metainformationen, Nutzerrechte sowie Ablagestruktur
- **Indexierung**: Vergabe von Suchkrieterien/Metainformationen
- **Suche**: detaillierte Suche nach den Dokumenten mit Hilfe von Metainformationen bzw. Volltextsuche über die Dokumente
- **Ausgabe**: Bildschirmdarstellung, Drucken und Weiteleitung (z.B. über E-Mail)
- **Ablage**: Mechanismen zur Sicherung, Archivierung und Wiederherstellung
- **Zugriffskontrolle**: Die Rechteprüfung bei dem Zugriff auf die Dokumente
- **Check in/out**: Übertragung einer Arbeitsversion in das Datenverzeichnis (Repository) zu einer endgültigen Version und umgekehrt
- **Workflow**: Organisation des Verarbeitungsflusses der Dokumenten

An dieser Stelle werden einige der o.e. Funktionalitäten näher erläutert, da sie im Kern vieler DMS zu finden sind und somit die Grundfunktionalitäten darstellen.

Indizierung: Das Dokument in einem DMS besteht aus den Nutz- (Dokumentinhalt) und den Metadaten. Metadaten sind Informationen zur Beschreibung des Dokumentes, welche für die Suche verwendet werden können, um ein schnelles Auffinden zu ermöglichen. Indizierung beschreibt also den Prozess

der Vereinbarung von Metainformationen eines Dokumentes und kann sowohl manuell als auch automatisch erfolgen. [vgl. Go04] S. 42

Suche: Nach dem die Dokumente erfolgreich in das System integriert wurden, lassen sich diese über die Suche im System lokalisieren. Die Ergebnisse der Suche sollten unter anderem nach verschiedenen Kriterien und Datenquellen sortiert werden können. Die Verwendung der booleschen Operatoren für die Verknüpfung der Suchbegriffe verfeinert den Suchprozess. Die Suche kann sich nach Metadaten (strukturierte Suche) oder auch nach Nutzdaten (Volltextsuche) richten. [vgl. Go04] S. 48-50

Zugriffskontrolle: Die Zugriffskontrolle umfasst die Prüfung der Nutzerrechte für den Zugriff auf die Dokumente. Des Weiteren steuert diese Kontrolle die Speicherung bzw. Wiedereinspielung der Dokumentenversionen. [vgl. Go04] S. 45-46

Check in/out: Beim Check-in werden die ausgewählten Dokumente entsprechend eigener Indizierung unter Beibehaltung der vorherigen Version gespeichert. Die Referenz auf das Dokument wird durch den Index in einer Metadatenbank gespeichert. Der Check-out macht ein Dokument beschreibbar und zeigt anderen Benutzern eine Sperre des Dokumentes durch einen Bearbeitungsvorgang an. [vgl. Gu02] S. 21

2.1.2.2. Klassifizierung der Funktionen

Aus den allgemeinen Funktionen lässt sich eine Klassifizierung der einzelnen Funktionen ableiten. Diese Klassifizierung wird vorgenommen, um das DMS so abstrakt wie möglich beschreiben zu können. Die Funktionen werden in die folgenden Bereiche aufteilt: *Eingabe*, *Ausgabe*, *Ablage* und *Administration*. Zur Eingabe gehört Erstellung/Erfassung, Indexierung und Klassifizierung von Dokumenten. Die Ausgabe beschäftigt sich mit der Suche, Darstellung, Weiterleitung und dem Drucken der Dokumente. Die Ablage beinhaltet u.a. die Statusverwaltung, Zugriffskontrolle und stellt damit den Kern des Systems dar. Die Funktionalitäten wie die Verwaltung der Nuterrechte sowie der Ablagestruktur sind für den Bereich Administration vorgesehen. [vgl. Go04] S. 38

2.1.3. Aufbau von DMS

Nach der Klassifizierung der Funktionen im Unterkapitel 2.1.2.2 in die Bereiche Eingabe, Ablage, Ausgabe und Administration lässt sich eine Struktur zum Aufbau eines DMS aufzeichnen. Diese Struktur beschreibt außerdem noch den Arbeitsablauf zwischen diesen Bereichen, der auch als Workflow bezeichnet wird, sowie die über das ganze System übergreifende Administration.

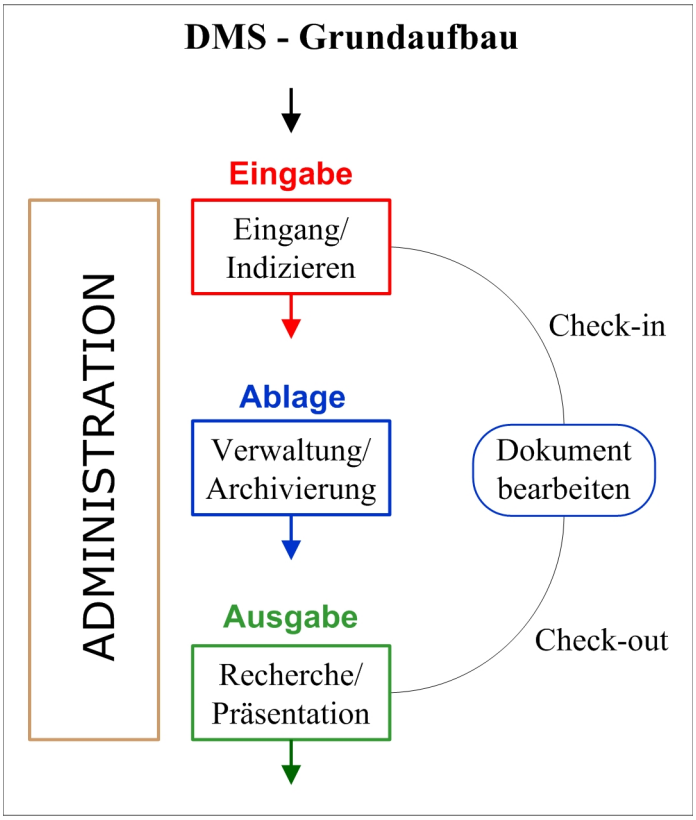

Abb. 1: Grundaufbau eines DMS

[Go04] S.38

2.1.3.1. Eingabe

Bei der Eingabe werden die Dokumente dem System zugeführt und indiziert, wobei eine Gültigkeitsüberprüfung hinsichtlich der Dokumentenarten und ihren Formaten stattfindet. [vgl. Go04] S. 39-40 Zusätzlich findet eine weitere Überprüfung statt, welche die Zugriffsrechte des Nutzers untersucht, um sicherzustellen, dass das Dokument von einem autorisierten Nutzer erstellt und dem System zugeführt wird. Bei dem Indizierungsvorgang ist die Anzahl der inhalts-

bezogenen Suchbegriffe für ein Dokument maßgebend, wie schnell und effektiv der Suchprozess für ein Dokument verläuft.

2.1.3.2. Ablage

Dieser Funktionsbereich befasst sich mit der Verwaltung und Archivierung der Dokumente in einem System. Bei der Verwaltung werden die Dokumente mit den zugehörigen Suchbegriffen in der Datenbank gespeichert. Dieser Vorgang wird auch als Check-in bezeichnet, was gleichbedeutend mit dem Einspielen ist. Dabei ist zu beachten, dass die Dokumente nur auf diesem Weg abgelegt werden dürfen. Wird ein Dokument zu einem späteren Zeitpunkt gebraucht, so kann das Dokument über einen Suchvorgang wieder gefunden werden. Falls eine Änderung des Dokumentes erforderlich ist, so wird bei diesem Vorgang, auch Check-out genannt, der Schreibzugriff auf das Dokument für die anderen Nutzer gesperrt. [vgl. Go04] S. 45-46 Die meisten Dokumenten-Management-Systeme verwalten auch den Status sowie deren logische Übergänge eines Dokumentes, weil über diese die Zugriffsrechte vergeben werden können. Nicht selten finden die verteilten Dokumenten-Management-Systeme ihre Anwendung. Die Dokumente werden auf mehreren Systemen im Netz verteilt, was die Lasten auf den Servern vermindert und die Zugriffszeiten niedrig hält. Somit kann der Zugriff auf alle Dokumente aus einer beliebigen Stelle im Netz erfolgen. [vgl. Go04] S. 46-48 Manchmal reicht es nicht aus, mehrere Dokumente einzeln zu verwalten, weil diese eine logische Einheit bilden. Dann werden diese Dokumente zu einem s. g. Container zusammengefasst und als eine Einheit verwaltet. Bei der Archivierung geht es um die Speicherung und das Wiederfinden der Dokumente samt aller Metainformationen. Der Schwerpunkt liegt in der Gewährleistung des schnellen Zugriffs auf die gesuchten Dokumente.

2.1.3.3. Ausgabe

Die Ausgabe umfasst die Suche nach einem Dokument und dessen Präsentation in unterschiedlichen Medienformaten. Um ein Dokument aus dem System zu finden, verwendet man entweder die strukturierte Suche oder die Volltextsuche. Bei der strukturierten Suche wird nach den vorgegebenen Suchbegriffen, Attributen oder Klassifizierung gesucht. Die Volltextsuche durchsucht

dagegen das komplette Dokument nach den Suchbegriffen. Die beiden Suchmöglichkeiten arbeiten unabhängig von einander. Bei der Präsentation soll das Dokument auf dem Monitor angezeigt, mit Hilfe eines Druckers gedruckt oder weitergeleitet (z.B. als Email) werden. Viele Systeme verfügen über einen Web-Client, welcher die Kommunikation mit einem entfernten Server erlaubt. [vgl. Go04] S. 48-50

2.1.3.4. Administration

Die Administration ist ein zentraler Punkt für die Verwaltung der Zugriffsrechte und Benutzereinstellungen. Außerdem ermöglicht dieser Bereich die Einsicht auf die Auswertungen der Zugriffshäufigkeiten, Speicherbelegung und diversen Protokollen. Die Datensicherung sowie die Datenwiederherstellung gehören auch zu dem Funktionsumfang des Administrationsbereiches. [vgl. Go04] S. 52-54

2.1.4. Vorteile und Nachteile

Durch den Einsatz von Dokumenten-Management-Systemen kann eine Reihe von Faktoren sowohl qualitativ als auch quantitativ verbessert werden. Gegenüber den klassischen papiergebunden Archiven können die Zugriffs-, Ablage- und Transportzeiten deutlich verkürzt und eine bessere Sicherheit geboten werden. Mit der Digitalisierung der Dokumente wird eine höhere Produktivität durch die Wiederverwendbarkeit erreicht und die Möglichkeit der integrierten Verarbeitung gegeben. Des Weiteren ist die Einbindung von multimedialen Informationsdaten gestattet. Die digitalisierten Dokumente verursachen bei der Archivierung weniger Kosten hinsichtlich des Platzbedarfs und des Archivpersonals. Mit dem dezentralen Zugriff wird eine gleichzeitige Nutzung der Dokumente von mehreren Personen erreicht, was zu einer Beschleunigung von Entscheidungsprozessen sowie einer Wertsteigerung der Information durch eine höhere Verfügbarkeit führen kann. Die zentrale Datenhaltung in einem DMS führt auch zu einer höheren Konsistenz der Daten.

Selbstverständlich bringen auch die Dokumenten-Management-Systeme einige Nachteile mit sich. Als großer Nachteil ist zuerst die aufwendige Eingabe zu erwähnen. Die Dokumente müssen sortiert, digitalisiert und indexiert werden,

um diese in das System einspielen zu können. Durch die unterschiedlichen Eigenschaften von Dokumenten wird diese Eingabe erschwert. Weiterhin sind der Einsatz und die Pflege von solchen Systemen immer mit zusätzlichem Fachpersonal und erhöhten Kosten verbunden. Als weiterer Nachteil ist die eingeschränkte Mobilität bei der Benutzung von Dokumenten zu benennen. Zu einem ist der Nutzer an einen Computer gebunden, was den Besitz und die damit verbundenen Kosten nach sich zieht. Zu anderem sind es die Position und der Standort für das Lesen und Schreiben, welche bei den papierbehafteten Dokumenten deutlich vielfältiger sind. Ein DMS ist auch als digitale Papierunterlage für die Ablage von Diskussionen und Besprechungen eher ungeeignet. [vgl. Gu02] S. 14-16, [vgl. al09]

2.2. Schichtenarchitekturen

„Schichtenarchitekturen definieren Prinzipien zur Strukturierung von Software- Architekturen.“ [it10] Bei dieser Strukturierung wird das ganze System in Schichten aufgeteilt. Die Beziehungen zwischen den Schichten unterliegen einer bestimmten Vorschrift, bei welcher eine höher liegende Schicht nur eine darunter liegende Schicht verwenden kann. Der Vorteil einer Schichtenarchitektur liegt in der Reduzierung der Abhängigkeiten innerhalb eines Systems. Somit wird eine lose Koppelung der einzelnen Schichten erreicht, was zur Aufspaltung der Gesamtkomplexität des Systems führt. Typischerweise wird heute bei den Lösungsansätzen von DMS auf eine Schichtenarchitektur als Basis gesetzt. Im Folgenden werden die meist praktizierten Schichtenarchitekturen näher erläutert. [vgl. Ka99] S. 43

2.2.1. Zwei-Schichten-Architektur

Bei der Zwei-Schichten-Architektur (eng. two tier architecture) wird eine Anwendung in zwei logische Schichten, die Präsentationsschicht und die Datenhaltungsschicht, aufgeteilt. Mit dem englischen Wort *tier* wird eine Schicht bezeichnet. Dabei kann die Anwendungslogik entweder in der einen oder auch in der anderen Schicht verarbeitet werden. Meist wird sie jedoch in der Präsentationsschicht vorgefunden. Um der o.e. Vorschrift zu entsprechen, wird die Datenhaltungsschicht unter der Präsentationsschicht platziert. Somit kann die In-

teraktion nur aus der Präsentationsschicht eröffnet werden. Aus diesem Grund wird diese Architektur auch als eine Client-Server-Architektur bezeichnet. Der Client ist der aktive Teil des Systems, welcher eine Aufgabe bzw. einen Dienst (s. Kapitel 2.3.1) vom Server anfordert. Nach der Bearbeitung der Aufgabe liefert der Server das Ergebnis an den Client zurück. Die Bezeichnungen Client und Server stehen in diesem Zusammenhang für die Teile einer Anwendung, welche sich auch auf einem physischen Rechner befinden können. Bei vernetzten Systemen werden die entsprechenden Schichten auf mehrere Rechner verteilt, um die Serverauslastungen so gering wie möglich zu halten. Beim Zugriff über die Client-Komponente sollte es gar nicht bemerkbar sein, dass sich eventuell mehrere Server die Aufgaben teilen und die zur Verfügung stehende Dienste anbieten. Der linke Bereich der Abb. 2 stellt diese Architektur in der vereinfachter Form dar. Im rechten Bereich der Abbildung wird diese Architektur in vernetzten Systemen aufgezeigt. [vgl. Ni96] S. 19-20, [vgl. Ka99] S. 43

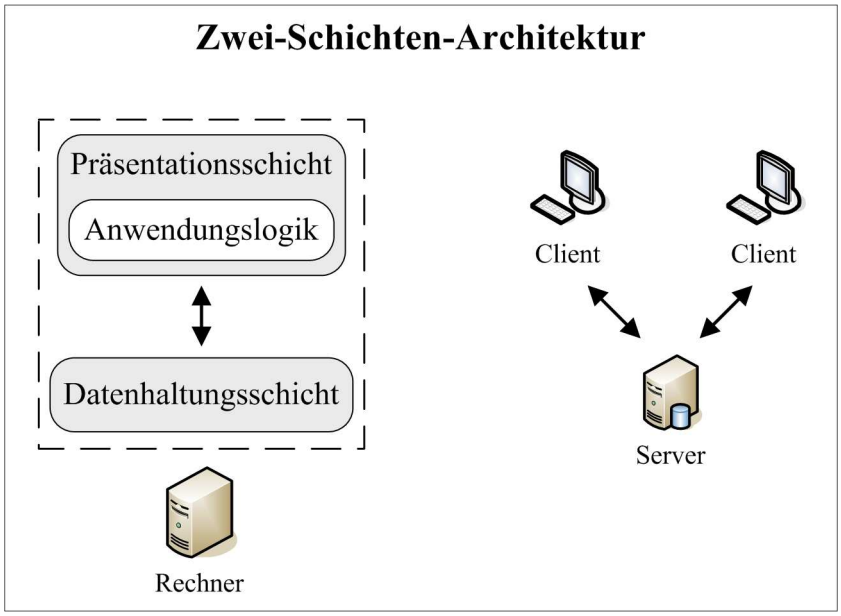

Abb. 2: Zwei-Schichten-Architektur

[vgl. Ni96] S. 19

2.2.2. Drei-Schichten-Architektur

Anders als bei der Zwei-Schichten-Architektur wird hier eine Anwendung softwareseitig in drei von einander unabhängige Schichten aufgeteilt:

- Präsentationsschicht
- Logikschicht
- Datenhaltungsschicht

Die Präsentationsschicht stellt die Informationen dar und nimmt die Benutzereingaben entgegen. Dazu existieren zwei Möglichkeiten der Realisierung. Ein Thin Client (deut. wörtlich *dünner Nutzer*) ist eine auf Ein- und Ausgabe beschränkte Anwendung. Ein Fat Client (deut. wörtlich *dicker Nutzer*) dagegen ist außerdem noch für die Verarbeitung der Ein- und Ausgabedaten zuständig.

Die Logikschicht, auch als *middle- tier* bekannt, befindet sich wie der Name schon sagt zwischen den beiden anderen Schichten und realisiert die eigentliche Logik der Anwendung. Diese Schicht ist die einzige, welche die Informationen an die Präsentationsschicht weitergibt. Dabei fungiert sie in einer Client-Server-Architektur als Server und die Präsentationsschicht als Client. Die Logikschicht ist wiederum die einzige, welche die Informationen aus der Datenhaltungsschicht erhält und verarbeitet. Bei diesem Kommunikationsablauf wird die Logikschicht zum Client und Datenhaltungsschicht zum Server.

Die Datenhaltungsschicht enthält die Daten, oder auch die Datenbank und ist für das Speichern und Laden verantwortlich.

Der Datenaustausch zwischen der Präsentationsschicht und der Logikschicht kann auf mehreren Wegen mit unterschiedlichen Technologien erfolgen, zum Beispiel über HTTP, XML-RPC, SOAP und Java- RMI. Dagegen werden beispielsweise die Schnittstellen wie JDBC oder ODBC für den Informationsaustausch zwischen der Logikschicht und der Datenhaltungsschicht verwendet.

Durch die Aufteilung und Entkoppelung der einzelnen Schichten wird diese Architektur sehr oft in der Praxis bei den vernetzten Systemen eingesetzt. Hierbei können die einzelnen Schichten auf unterschiedlichen physischen Rechnern ausgelagert werden. Die Abb. 3 zeigt den grundlegenden logischen Aufbau einer Anwendung in einer Drei-Schichten-Architektur. Ebenso erfolgt eine Gegenüberstellung dieser Architektur in vernetzten Systemen. [vgl. Ka99] S. 43-45

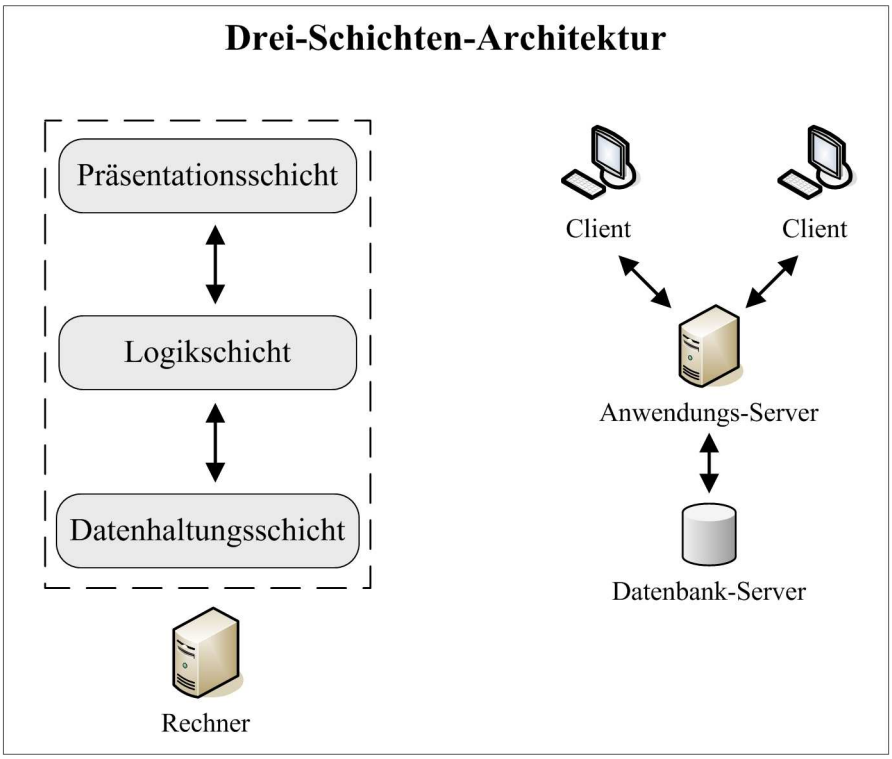

Abb. 3: Drei-Schichten-Architektur

[vgl. Ka99] S. 44

2.3. Serviceorientierte Architektur (SOA)

„Mache die Dinge so einfach wie möglich – aber nicht einfacher.“
(Albert Einstein)

Momentan gibt es keine eindeutige Definition, welche den Begriff SOA vollständig beschreibt. Nach MELZER könnte man sich unter SOA Folgendes vorstellen: „Serviceorientierte Architekturen, kurz SOA, sind das abstrakte Konzept einer Software-Architektur, in deren Zentrum das Anbieten, Suchen und Nutzen von Diensten über ein Netzwerk steht.“ [Me08] S. 9 In Verbindung mit einigen wichtigen Merkmalen der SOA wird dieser Begriff nach MELZER wie folgt definiert: „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.“ [Me08] S. 13

2.3.1. Webservice

Das Kernelement der serviceorientierten Architektur bildet mindestens ein Webservice bzw. ein Webdienst. WORLD WIDE WEB CONSORTIUM, auch W3C genannt, definiert den Begriff des Webservices wie folgt: Ein Webservice ist ein Softwaresystem, welches für die Unterstützung der kompatiblen Interaktion zwischen den Endgeräten über das Netzwerk entwickelt wurde. Dieser Webservice besitzt eine Schnittstelle, welche in einem Rechner- verarbeitenden Format (meistens WSDL, s. Kapitel 2.3.3.1) beschrieben wird. Andere Systeme interagieren mit dem Webservice durch seine Beschreibung unter Verwendung der SOAP- Nachrichten (s. Kapitel 2.3.3.2). Diese Nachrichten werden üblicherweise über das HTTP mit einer XML- Serialisierung in Verbindung mit anderen netzspezifischen Standards versandt. [vgl. ww10]

Einfach betrachtet kapselt ein Webservice eine beliebige Funktionalität ein, welche durch die Nutzer bzw. die Anwendungen über eine vordefinierte Service-Schnittstelle in einem Netzwerk oftmals verwendet werden kann. Dabei kann ein zusammengesetzter Webservice selbst aus mehreren Services bestehen.

2.3.2. Grundelemente der SOA

Die Abb. 4 zeigt die Grundelemente einer SOA in einem s.g. *SOA- Tempel* auf. Die Basis dieser gesamten Architektur bilden drei Stufen [vgl. je10]:

- **Standards**: Diese Architektur verwendet offene Standards (s. Kapitel 2.3.3.), welche für eine breite Akzeptanz beim Einsatz und bei der Verwendung der Architektur sorgen.
- **Sicherheit**: Die Realisierung der SOA bedarf der Berücksichtigung aller Sicherheitsfragen wie Authentifizierung, Autorisierung, Vertraulichkeit etc.
- **Einfachheit**: Die Forderung nach Einfachheit wird durch zwei Faktoren beschrieben. Zum einen soll die Technologie an sich einfach sein und zum anderen die Einfachheit ihrer Anwendbarkeit.

Auf dem Fundament stehen vier Säulen, welche die Merkmale einer SOA darstellen sollen. Eines der Hauptmerkmale ist die lose Kopplung der Webservices. Die Webservices werden bei Bedarf dynamisch gesucht, gefunden und eingebunden. Aus diesem Grund kann es erst zur Laufzeit festgelegt werden,

welcher Nutzer welchen Dienst aufruft. Um einen Dienst überhaupt ausführen zu können, muss zuerst nach ihm gesucht werden. Dafür ist ein Verzeichnisdienst (Repository) vorhanden, welcher alle verfügbaren Dienste registriert. Um den gefundenen Dienst nutzen zu können, muss der Dienstnutzer sich mit diesem verständigen. Die *SOA- Standards* (s. Kapitel 2.3.3.) definieren eine einheitliche Sprache, um den Verständnisproblemen aus dem Weg zu gehen. Auf diese Weise können unterschiedliche Anbieter ihre Dienste zur Verfügung stellen, ohne dass es zu Problemen bei der Kommunikation mit den Dienstnutzern kommt. Gleichzeitig erfolgt die Trennung von Schnittstelle und Implementierung der Dienste, was die Wiederverwendbarkeit der Dienste in unterschiedlichen Einsatzgebieten ermöglicht und die serviceorientierte Architektur sich somit einfach darstellen lässt. [vgl. Me08] S. 11-14

Die Verbindung des Fundamentes mit den vier tragenden Säulen zu einem Konstrukt, welche zusammen das Dach einer SOA stützen, stellt ein *SOA-Tempel* dar.

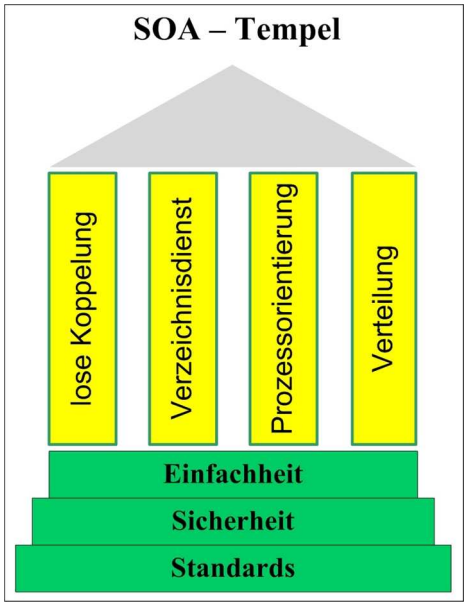

Abb. 4: SOA- Tempel

[Me08] S. 13

2.3.3. SOA- Standards

Die breite Akzeptanz der serviceorientierten Architektur wird durch die Verwendung von offenen Standards bekräftigt. Aus diesem Grund werden an dieser Stelle die wichtigsten Standards in dieser Architektur näher erläutert.

2.3.3.1. WSDL – Web Service Description Language

„Sage nicht immer was du weißt, aber wisse immer, was du sagst.“
(Matthias Claudius)

Das WSDL ist eine XML- basierte Sprache zur einheitlichen Beschreibung von Services, welche eine Syntax definiert, um die Service- Schnittstelle unabhängig vom Kontext darzustellen. Die WSDL- Dokumente realisieren somit eine Beschreibungsschicht, durch welche eine Trennung der Innen- und Außendarstellung von Services erzielt wird. [vgl. vB08] S. 20

Die Struktur von einem WSDL- Dokument kann in einen logischen und einen physischen Bereich getrennt werden. Der logische Bereich wird für den Zugriff zum Service in der Schnittstellendefinition definiert. Der physische Bereich spezifiziert die Verbindung zum Service in der Implementierungsdefinition. Die Schnittstellendefinition enthält die Elemente für den Datentyp, die Nachricht, die Schnittstelle und die Bindung.

- **Datentypelement**: Definition der zu verwendeten Datentypen in den Nachrichten (<wsdl: types>).
- **Nachrichtenelement**: Definition der zu verarbeitenden Nachrichten eines Webservices. Die Nachrichten können auch in logische Teile eingeteilt werden (<wsdl: message>, <wsdl: part>).
- **Schnittstellenelement**: Kapselung der Operationen (<wsdl: portType>). Die Operationen entsprechen den Methoden, welche dem Nutzer zur Verfügung gestellt werden (<wsdl: operation >).
- **Bindungselement**: Definition des Protokolls zwischen dem Anbieter und dem Nutzer für die einzelnen Schnittstellenelemente (<wsdl: binding>).

Die Implementierungsdefinition beschäftigt sich mit der Verbindung des Dokumentes mit der Serviceimplementierung und enthält die Elemente für den Endpoint und den Service.

- **Endpointelement**: Definition der Netzwerkadresse und des Portes für die Nutzung des Services (<wsdl: port>).
- **Serviceelement**: Definition einer logischen Menge von Endpointelementen (<wsdl: service>).

Das WSDL- Dokument weist eine spezifische Struktur auf, in welcher die Serviceelemente alle Endpointelemente zusammenfassen. Die Endpointelemente referenzieren wiederum die Bindungselemente. Für die einzelnen Schnittstellenelemente werden Bindungselemente definiert. Die Endpointelemente kapseln die Operationen. Diese Operationen werden über die Nachrichtenelemente verwendet, welche die vorgegebenen Datentypelemente einsetzen. Die aufgezählten sechs Hauptelemente werden auch in die abstrakten und konkreten Bereiche eines WSDL- Dokumentes klassifiziert. Die Bereiche mit den Elementen für den Datentyp, die Nachricht und die Schnittstelle bilden den abstrakten Teil, wogegen die Elemente für die Bindung, den Endpoint und den Service den konkreten Bereich darstellen. [vgl. vB08] S. 21

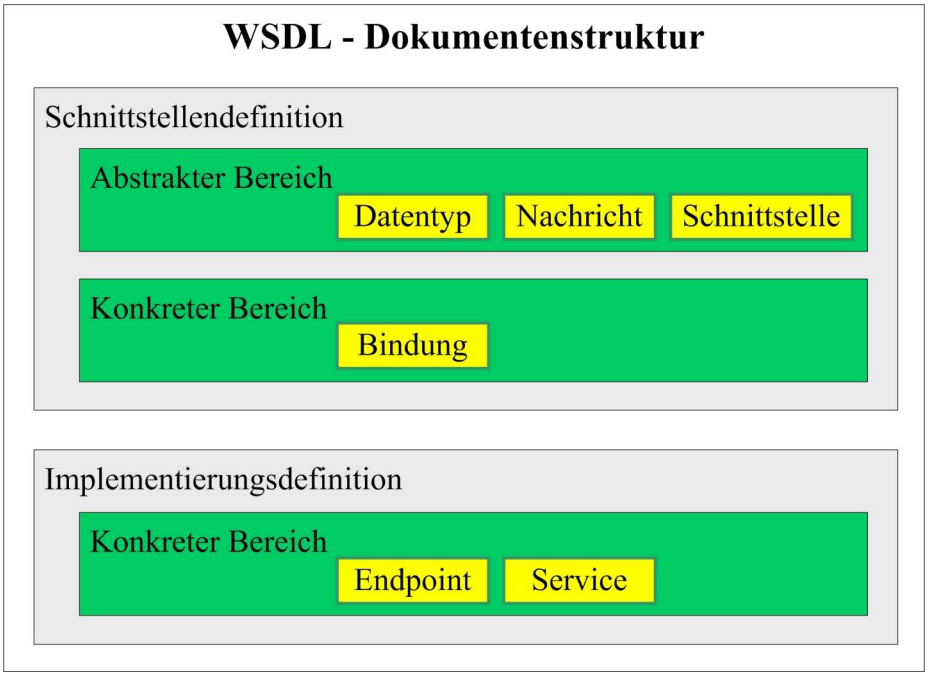

Abb. 5: WSDL- Dokumentenstruktur

[vgl. vB08] S. 20

2.3.3.2. SOAP – Simple Object Access Protocol

Das SOAP ist ein Protokoll für den Informationsaustausch mit Webservices und beschreibt gleichzeitig seine Einbettung in die meisten Transportprotokolle. [vgl. Ma07] S. 239 Die Nachrichten werden mittels dieses Protokolls in einer serviceorientierten Architektur auf Basis von Webservices übertragen. Auch eine Kommunikation per RPC (Remote Procedure Calls) kann auf der Grundlage von SOAP erfolgen. Da das Protokoll nicht an ein Betriebssystem oder eine

Programmiersprache gebunden ist, wird eine lose Kopplung der Services erreicht. Die SOAP- Spezifikation trifft keine Annahmen über den Kommunikationskanal, die Sprache und die Umhüllung einer Nachricht. Wichtig ist nur, dass der Sender und der Empfänger sich darüber einig sind, welches Verfahren eingesetzt wird und beidseitige Interpretationsmöglichkeit von diesem Verfahren besteht. [vgl. vB08] S. 23

Aufbau einer SOAP- Nachricht

Eine SOAP- Nachricht ist ein XML- Dokument, welche in den SOAP- Umschlag (Envelope) gekapselt wird und den SOAP- Kopf (Header) mit dem SOAP- Rumpf (Body) einschließt. Der Umschlag beinhaltet die Nachricht und ist zugleich das Wurzelelement in dem XML- Dokument. Der Kopf ist ein Bereich, welcher zusätzliche Informationen einer SOAP- Nachricht enthalten kann und kommt in der Nachricht höchstens einmal als Unterelement des Umschlages vor. Zum Beispiel können an dieser Stelle die Informationen über die Sicherheit oder auch über die Transaktionssteuerung ihren Platz finden. Im Rumpf werden die eigentlichen Nutzdaten übertragen. Dieser Bereich wird an zweiter Position als Unterelement des Umschlags in dem XML- Dokument platziert und darf genau einmal vorkommen. [vgl. vB08] S. 24

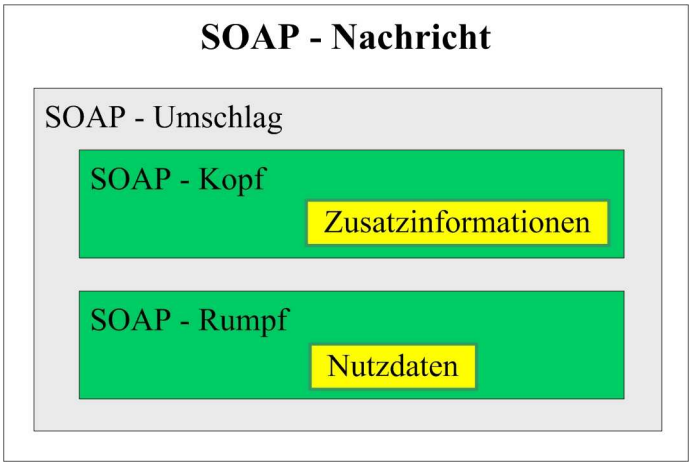

Abb. 6: Aufbau einer SOAP- Nachricht

[vgl. Me08] S. 79

Kommunikationsablauf

Der Sender (SOAP- Client) wählt einen Kommunikationskanal (z.B.: HTTP) aus und erzeugt eine Nachricht (XML- Dokument) in einem für den Empfänger (SOAP- Server) verständlichen Format. Bevor die Nachricht versendet werden

kann, muss sie durch den Sender serialisiert, also umhüllt werden, um auf der Empfängerseite diese Nachricht als eine SOAP- Nachricht erkennen zu können. Der Empfänger deserialisiert die Nachricht nach der Übertragung und kann den Inhalt interpretieren. Die Reaktion des Empfängers findet spiegelbildlich statt. [vgl. Me08] S. 77-78, [vgl. vB08] S. 24

Abb. 7: Kommunikationsablauf mit SOAP

[vgl. Me08] S. 78

SOAP- Alternativen

Die Bezeichnung eines einfachen Protokolls für den Objektzugriff kann bei SOAP als falsch angesehen werden. Die Vorteile von SOAP wie die Transportprotokollunabhängigkeit sowie der asynchrone Informationsaustausch führen zu der Komplexität der Spezifikation und der erschwerten Verwendung. Aus diesem Grund werden in kurzer Form zwei weitere Alternativen, das XML-RPC und REST, vorgestellt.

XML-RPC (Extensible Markup Language – Remote Procedure Calls) ist ein Protokoll und dient einem entfernten Methodenaufruf, welcher die XML-Nachrichten einzig über das Transportprotokoll HTTP in synchroner Übertragung versendet. Die kürzeren Elementnamen sowie das Ausschließen der XML-Namensräumen repräsentieren es als eine wirklich einfache Alternative zu SOAP. Das XML-RPC ist auch plattform- und sprachunabhängig. Des Weiteren hat RPC ein vereinfachtes Typsystem, mit dem nicht alle Typen direkt abgebildet werden können und somit mit mehr Aufwand bei der Implementierung verbunden ist. [vgl. Me08] S. 100-103

Das REST (REpresentational State Transfer) ist nur ein Architekturstil zur Definition der Verwendung von den vorhandenen Transportprotokollen und somit kein Protokoll in der Transportschicht. Im Gegensatz zu SOAP liegt der Fokus beim REST nicht in vielen verteilten Anwendungen, sondern nur in den Anwendungen für das WWW (World Wide Web). Eine Anwendung mit diesem Architekturstil kann jede einzelne Ressource mittels einer eindeutiger URL (Uni-

form Resource Locator) verwalten. Die Ressourcen können beliebige Objekte aus der realen Welt sein. Ihre Repräsentation wird mit HTTP-GET Operationen abgefragt und für die Manipulation werden HTTP-PUT, HTTP-POST und HTTP-DELETE eingesetzt. Die Aufrufe von den Webservices lassen sich leicht mit den HTTP Operationen erstellen und benötigen keine XML- Dokumente. [vgl. Me08] S. 103-105

2.3.3.3. UDDI – Universal Description, Discovery and Integration

In der serviceorientierten Architektur bilden die Verzeichnisdienste einen zentralen Punkt, weil diese Dienste eine Übersicht über die Ressourcen, insbesondere die Webservices, für die Nutzer und deren Anwendungen anbieten. Genau diese Verzeichnisdienste sind maßgeblich für die Realisierung einer losen Kopplung von Webservices, welche das dynamische Einbinden von Webdiensten ermöglichen. Durch die standardisierte Schnittstellen und die internen Datenstrukturen ermöglicht dieser Verzeichnisdienst eine strukturierte Suche nach einem Webservice. UDDI ist ein Verzeichnisdienst in der SOA. [vgl. vB08] S. 24

Die Kernaufgaben von einem UDDI- Verzeichnis sind das Registrieren und Auffinden von Diensten, welche von einem Anbieter bereitgestellt werden. Der Anbieter veröffentlicht seinen Dienst in Form einer WSDL- Beschreibung in einem Verzeichnis und der Nutzer ruft diese Schnittstellenbeschreibung nach der erfolgreichen Suche ab. Das UDDI ist ein XML- basierter Standard zur Definition der Sprachkonstrukte der Webservicebeschreibungen und der Regelung von Zugriffs- und Replikationsoperationen. [vgl. Me08] S. 137 Die registrierten Webservices in einem Verzeichnis werden nach der UDDI- Spezifikation in drei s.g. Entitäten kategorisiert.

- **White Pages**: Die Anbieter, welche ihre Dienste in einem Verzeichnis registriert haben, können hier Informationen über sich selbst angeben, um die eigene Identität den Nutzern mitzuteilen.
- **Yellow Pages**: Die registrierten Dienste werden nach bestimmten Klassifikationsmustern in Hinsicht auf ihren Zweck gruppiert, z.B.: Branchen oder Ländercodes. Die Klassifikation kann dabei einem Standard oder einer Anbieterspezifikation unterliegen.

- **Green Pages**: An dieser Stelle werden die Schnittstellenbeschreibungen (WSDL- Dokumente) von Webservices verzeichnet.

Da die Green Pages durch die menschlichen Nutzer verwendet werden, werden mit der Service Type Registration Informationen in maschinenlesbarer Form dargestellt, welche mit den Informationen den Green Pages identisch sind. [vgl. Me08] S. 138

Für die Beschreibung der o.e. Entitäten besitzt das UDDI eine standardisierte Datenstruktur aus den Komponenten, welche die Beziehungen zwischen den Anbietern, sowie deren Diensten und Schnittstellen wie folgt festlegt:

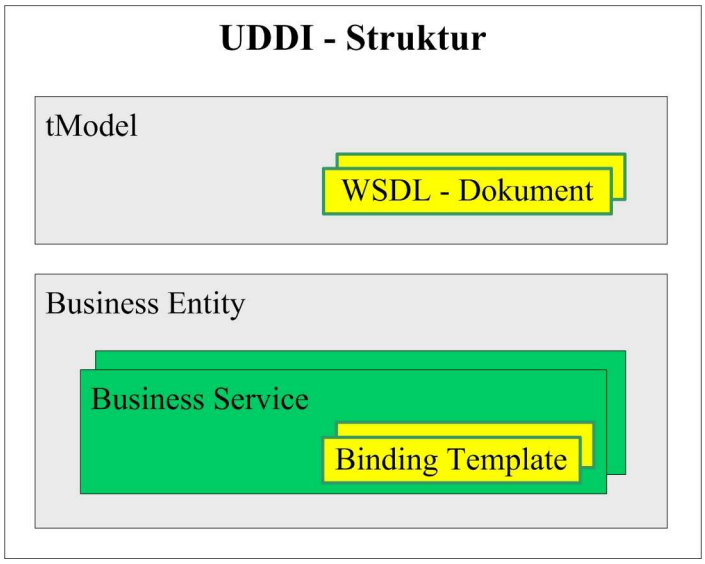

Abb. 8: UDDI – Datenstruktur

[vgl. vB08] S. 25

- **Business Entity**: dient der Modellierung von Anbietern und beinhaltet die Angaben der White Pages. Es werden außer den wichtigen Attributen wie der Name und die ID noch eine Liste von Business Service- Datenstruktur (alle Dienste des Anbieters) festgehalten.
- **Business Service**: stellt einen Webservice dar, welcher durch die Business Entity angeboten wird. Der Dienst wird allgemein entsprechend den Yellow Pages beschrieben. Ein Business Service kann auch ein zusammengesetzter Service aus mehreren einzelnen Services sein.

- **Binding Template**: enthält die technischen Informationen von Business Service und ist das Pendant zu den Green Pages. Das Binding Template stellt eine Referenz zu tModel, welches weitere technische Beschreibung des Dienstes liefert.
- **tModel**: beschreibt die Schnittstelle eines Services unabhängig von dessen Implementierung. Meistens werden für diese Beschreibung von Diensten die WSDL- Dokumente verwendet. Das tModel ist kein Unterelement der vorgestellten Strukturen und lässt sich somit für weitere ähnliche Dienste wieder verwenden. Des Weiteren besteht die Möglichkeit die Webservices auf technischer Ebene miteinander zu vergleichen. Zusätzlich kann definiert werden, wie ein Service anzusprechen ist und welche Spezifikationen und Konstrukte er verwendet.
- **Publisher Assertion**: beschreibt ein Verhältnis zwischen zwei Anbietern (Business Entities), z.B. mit dem Ziel als eine Organisation nach außen zu wirken. Dabei bedarf es einer beidseitigen Zustimmung.
- **Operational Info**: beinhaltet die Veränderungen im UDDI- Verzeichnis. Dabei werden in dieser Datenstruktur die Angaben (was, wann und von wem erstellt oder geändert) abgelegt.

Um auf die UDDI- Verzeichnisse zugreifen zu können, wurde eine UDDI-API zur Verfügung gestellt, welche die Basis für den Zugriff auf die SOAP-Nachrichten liefert. [vgl. vB08] S. 25-26

2.3.3.4. Web Service Inspection Language

Eine Alternative zu UDDI ist die WS- Inspection, welche deutlich einfacher aufgebaut ist. Dieses Konstrukt ist grundsätzlich anders, da es mit vielen dezentralen Verzeichnissen arbeitet. Dabei werden meistens alle Dienste eines Anbieters in einem Verzeichnis registriert und von dem jeweiligen Anbieter verwaltet. Der Grundgedanke der WS- Inspection liegt darin, dass die Nutzer bei denjenigen Anbietern nach passenden Diensten suchen, welchen sie auch ihr Vertrauen erweisen. Die Nutzer rufen mittels HTTP ein Dokument (*inspection.wsil*) ab, welches der Anbieter auf dem Server abgelegt hat. Dieses enthält eine Liste mit

allen Diensten des jeweiligen Anbieters sowie eine WSDL- Beschreibung. Die Spezifikation von WS- Inspection definiert das XML- Dokumentenformat für die WSIL- Datei und Regeln für ihr Auffinden. Weiterhin besitzt dieser Standard die entsprechenden Erweiterungen für UDDI und WSDL, um die benötigten Informationen in der WSIL- Datei zu verwenden. Die Datenstruktur ist einfacher als bei UDDI. Sie besteht hier aus drei Komponenten [vgl. Me08] S. 134-135:

- **Service**: definiert die Dienste und beinhaltet die Schnittstellenbeschreibung.
- **Inspection**: umfasst alle Service- und Link- Elemente.
- **Link**: verweist auf externe Datenquellen, z.B. andere WSIL- Dokumente oder auch UDDI- Verzeichnisse.

2.3.4. Das magische Dreieck

Das Rollenmodell einer SOA, auch *Das magische Dreieck* genannt, beschreibt die wichtigsten Webservice- Standards in Bezug auf die vordefinierten Rollen dieser Architektur. Im Folgenden werden die drei Rollen vorgestellt [vgl. vB08] S. 19:

- Dienstanbieter
- Dienstnutzer
- Dienstverzeichnis

2.3.4.1. Dienstanbieter

Eine Plattform zur Nutzung der Dienste und deren Zugriff über ein Netzwerk wird von einem Dienstanbieter zur Verfügung gestellt. Die Nutzer können allerdings nur die Dienste finden, welche in einem Dienstverzeichnis auch registriert sind. Der Dienstanbieter ist auch für den Betrieb und Wartung der gestellten Plattform zuständig. Eine weitere Aufgabe von ihm ist die Sicherheit der Plattform, sprich die Authentifizierung und die Authentisierung. Ein Anbieter muss aber nicht alle zur Verfügung gestellte Dienste selbst implementieren. Er kann auch fremde Dienste kapseln oder die vorhandenen Dienste in erweiterter Form wieder anbieten. Die Qualität des Services (QoS) ist ein wichtiger Punkt bei der Inbetriebnahme eines Webservices durch den Anbieter. Um dem Nutzer die Qualität des Webservices mitzuteilen, bedarf es eines so genannten SLA

(Service Level Agreement). Das SLA ist ein Vertrag zwischen einem Anbieter und einem Nutzer, welcher die technischen und wirtschaftlichen Vereinbarungen eines Services festlegt. [vgl. Me08] S. 16

2.3.4.2. Dienstnutzer

Der Dienstnutzer in einer SOA ist mit einem Verbraucher zu vergleichen, welcher eine bestimmte Leistung von einem Anbieter erwartet und diese in Anspruch nehmen möchte. Er muss nicht unbedingt dem Anbieter bekannt sein, um seine Dienste verwenden zu können. Der Anbieter muss jedoch die Schnittstellen der Dienste darlegen, um die Kommunikation über ein für beide bekanntes Protokoll zu ermöglichen. Der Nutzer ist auf die offenen Standards mehr als angewiesen. Er muss in der Lage sein den Dienst zu finden, das Wissen über den Aufruf zu erlangen und letztendlich die Kommunikation durchzuführen. [vgl. Me08] S. 17-18

2.3.4.3. Dienstverzeichnis

Der Dreh- und Angelpunkt in der Interaktion zwischen dem Anbieter und dem Nutzer in der serviceorientierten Architektur ist ein solches Verzeichnis. Die Hauptaufgabe dieses Verzeichnisses liegt beim Auffinden der gewünschten Dienste durch den Nutzer und das aktive Registrieren der zur Verfügung gestellten Dienste durch den Anbieter. Ein guter Vergleich ist das Verzeichnis von *gelben Seiten*. Der Unterschied liegt jedoch in der Vielfalt von Dienstverzeichnissen durch eine große Anzahl der Anbieter. Eine mögliche Alternative für ein einziges Verzeichnis mit unterschiedlichen Anbietern und vielen Diensten wird bereits praktiziert. [vgl. Me08] S. 16-17

2.3.4.4. Interaktion der Rollen

Am Anfang muss der Anbieter einen Dienst implementieren, um es überhaupt den Nutzern zur Verfügung stellen zu können. Nach der Fertigstellung des Dienstes wird eine Schnittstellenbeschreibung mittels WSDL im Dienstverzeichnis erstellt und veröffentlicht. Das Verzeichnis ist nach dem UDDI- Standard strukturiert. Damit wird der erstellte Dienst für die potenziellen Nutzer auf-

findbar gemacht. Die Nutzer senden an das Dienstverzeichnis ihre Suchanfragen, welche in Form einer SOAP- Nachricht vorliegen. Sobald der Nutzer den gewünschten Dienst gefunden hat, erlangt er den Verweis auf die vorher abgelegte Schnittstellenbeschreibung des Anbieters über WSDL, welche in einem Netzwerk durch ein URI exakt bestimmt werden kann. Im letzten Schritt fragt der Nutzer die Beschreibung bei dem Anbieter über SOAP ab und wird in die Lage versetzt, den Dienst ebenfalls über SOAP nutzen zu können. Diese drei Rollen fungieren als eigenständige Anwendungssysteme in der serviceorientierten Architektur. [vgl. vB08] S.19-20

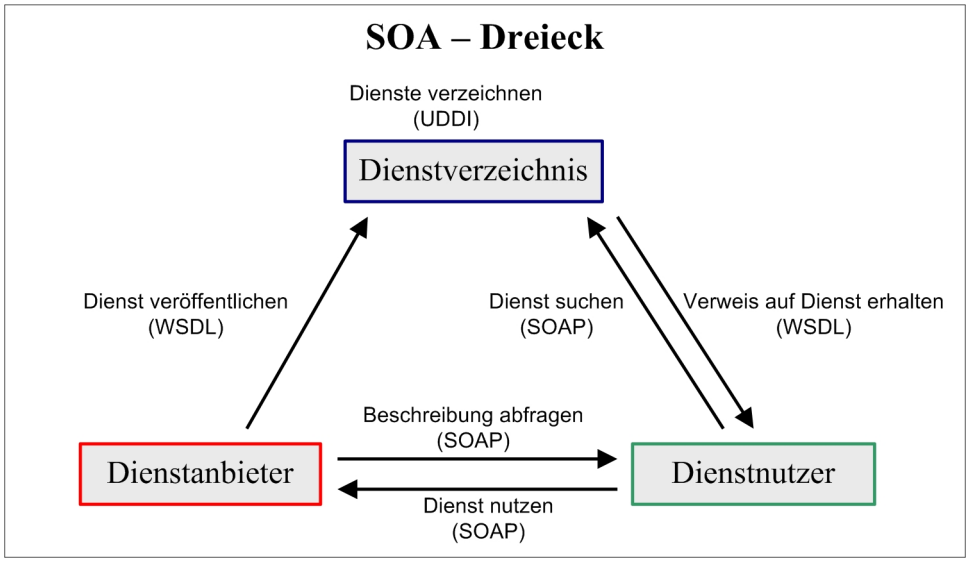

Abb. 9: Das magische Dreieck

[vgl. vB08] S.19

2.3.5. Sicherheit

Der Faktor Sicherheit spielt eine wesentliche Rolle bei den verteilten Systemen und insbesondere in der serviceorientierten Architektur mit dem Einsatz von Webservices. Der Zugriff auf die Informationen sollte nicht allen Beteiligten im gleichen Maße geboten werden, sondern z.B. klassifiziert nach den Nutzergruppen. Außerdem ist die sichere Übertragung der Nachrichten in solch einer Architektur zu gewährleisten. Aus diesen und anderen Überlegungen heraus lassen sich einige Anforderungen nach ihrer Art einordnen. [vgl. Jo08] S. 213-214

- **Authentifizierung**: dient der Nutzeridentifikation. Der Nutzer in SOA kann durchaus nicht nur ein Mensch, sondern auch netz-

werkfähiges Gerät oder ein externer Service- Nutzer sein. Die Aufgabe liegt also in der Bestimmung des Nutzers, für den er sich tatsächlich ausgibt

- **Autorisierung**: ist für die Prüfung der Rechte des Nutzers zuständig. Diese regelt, ob der Aufrufer den Service ausführen darf und eventuell auch die Ergebnisse ausgeliefert bekommt
- **Vertraulichkeit**: beschäftigt sich mit der Gewährleistung der Sicherheit der Daten bei dem Transport zwischen dem Anbieter und dem Nutzer. Es geht darum, dass die vertraulichen Informationen nicht durch das s.g. *Backdoor* zu den *Dritten* gelangen
- **Integrität**: stellt sicher, dass die Daten in sich erhalten und gültig bleiben. Die Integrität soll von der unzulässigen Manipulation der Daten schützen
- **Verfügbarkeit**: ist eine prozentuale Größe, welche sich anhand der Zeit definieren lässt. Sie gibt ein Maß für die Verfügbarkeit des Systems für die Nutzer in einem Zeitraum an
- **Accounting**: ist ein Begriff aus der Wirtschaft, welcher für die Protokollierung der Nutzung von Diensten verwendet wird

Um diesen Anforderungen an Sicherheit gerecht zu werden, wurden einige Standards in den Bereichen XML und Webservices ausgearbeitet. Für XML sind folgende Standards zu erwähnen:

- **XML Signature**: ermöglicht die digitale Unterschrift von XML- Dokumenten, um ihre Richtigkeit bzw. Gültigkeit nachzuweisen. Die Verwaltung von den digitalen Schlüsseln für diesen Standard wird von einem XML Key Management- Standard unterstützt.
- **XML Encryption**: ist in der Lage ein ganzes XML- Dokument oder auch nur Teile davon zu verschlüsseln
- **SAML** (Security Assertion Markup Language): ist eine XML- basierte Sprache zur Verwaltung und Austausch von Sicherheitsinformationen

Zu den wichtigen Webservices- Standards gehören zum jetzigen Zeitpunkt folgende [vgl. Jo08] S. 226-227:

- **WS- Security**: definiert die Art und den Ort für die Anwendung der einzelnen Sicherheitsmechanismen bei den Webservices in dem

SOAP- Protokoll. Mit diesem Standard werden SOAP- Nachrichten um sicherheitsrelevante Informationen und Verfahren erweitert.

- **WS- Security Policy**: wird verwendet, um die Sicherheitsanforderungen durch den Anbieter festzulegen. Somit kann der Anbieter bereits in der Schnittstellenbeschreibung seines Dienstes die Angaben über die unterstützten bzw. erwarteten Standards bestimmen.
- **WS- Trust:** wird für das Ausstellen, Erneuern und Validieren von Sicherheitsschlüsseln (*Security Token*) verwendet. Dieser Standard erweitert das *WS- Security* und verwendet die *WS- Security Policy*.

2.3.6. Vorteile und Nachteile

Ein großer Vorteil der SOA ist die Unabhängigkeit der Implementierung, was eine funktionale Zerlegung der Anwendung ermöglicht und eine prozessorientierte Betrachtungsweise beisteuert. [vgl. Me08] S. 9 Mit der serviceorientierten Architektur lassen sich die Geschäftsprozesse besser abbilden sowie durch die vorhandene Flexibilität dynamische Geschäftsmodelle erzeugen. Der Einsatz von offenen Standards liefert eine breite Akzeptanz von SOA. Durch den Einsatz von Diensten in SOA lassen sich in sich abgeschlossene Anwendungsmodule wieder verwenden und somit die Ausgaben reduzieren.

Als Nachteile von SOA sind zuerst die erforderlichen Investitionen zu erwähnen. Die Nutzer und die Entwickler müssen sich mehr mit den Geschäftsprozessen auseinander setzen. Allerdings ist der Einsatz von solchen Architekturen nur für eine große Nutzergruppe sinnvoll.

2.4. open-EIS Plattform

open-EIS ist eine Integrations- und Entwicklungsplattform mit Enterprise-Technologie. Dieser Technologie liegt der Client- Server- Prinzip zugrunde. Die Plattform basiert auf einer serviceorientierten Drei-Schichten-Architektur für den effizienteren Aufbau neuer, moderner, betriebssystem-unabhängiger und hochspezialisierter Unternehmensanwendungen in heterogenen Umgebungen. Sie

bildet somit die Basis für die individuelle Lösung, welche aus der Kombination von einzelnen, fachlich getrennten Anwendungsbausteinen besteht. Auf diese Weise wird eine schnelle Reaktion auf die veränderten Geschäftsprozesse mit flexiblen Anpassungen und Erweitrungen durch integrierte Mechanismen erreicht. Diese Plattform basiert auf den folgenden Grundkonzepten: [vgl. Co07] S.2

- Integration
- Modellierung
- Flexibilität
- Skalierbarkeit
- Effizienz
- Abstraktion

Unter Integration wird die Konnektivität zu Fremdsystemen sowie die Integration oder Adaption bestehender Altsysteme verstanden, weil diese Plattform als zentrale Integrationslösung verschiedener Unternehmensanwendungen eingesetzt werden kann.

Auf der Basis der modellgetriebenen Softwareentwicklung ist die plattformunabhängige Modellierung von wieder verwendbaren Lösungsentwürfen, welche die Grundlage für einen aufwandsärmeren und weniger fehleranfälligen zum Teil automatisierten Prozess der Entwicklung bildet, gestattet. Dabei können bereits auf der Modellebene die Entwürfe validiert und realisiert werden. Dieser Ansatz wird in der Softwareentwicklung auch als MDA- Prozess bezechet.

Die flexible Anpassungen und Erweiterungen von komplexen Geschäftsprozessen werden durch integrierte Mechanismen realisiert. Durch den Einsatz verschiedener Abstraktionsgrade können unterschiedliche Anforderungen spezieller Anwendungen realisiert und damit die effiziente Entwicklung verteilter Anwendungen ermöglicht werden.

Die Skalierbarkeit wird durch die flexible Architektur ermöglicht, welche eine performante Laufzeit der Anwendungen erzielen sowie den Wachstumsansprüchen von Anforderungen gerecht werden kann. Dies wird in open-EIS mit verschiedenen Cluster-Szenarien, SOA- Unterstützung sowie Datenkoordination mittels Replikation oder Servicetechnologien umgesetzt.

Die Effizienz der Plattform wird durch die verschiedenen Entwicklungsansätze, dynamisches Datenmodell (DDM) und Datenabstraktionsschicht (O/R-Mapping), für eine zeit- und kosteneffiziente Anwendungsentwicklung unterstützt.

Die Unterstützung von verschiedenen Enterprise-Technologien und deren Interaktion innerhalb eines Systems wird durch verschiedene Abstraktionsebenen in der Datenabfrage, SOA- Konzept, Modellierung und Benutzerschnittstellenanbindung realisiert.

Wie bereits oben erwähnt basiert open-EIS auf einer Drei-Schichten-Architektur, welche aus der Präsentations-, Logik- und Datenhaltungsschicht besteht. Die Präsentationsschicht übernimmt alle Aufgaben des UI (User Interfaces), die Logikschicht führt die Anwendungslogik aus und die Datenhaltungsschicht ist für das Speichern und Laden der Daten zuständig. Die Kommunikation zwischen diesen logischen Schichten wird über die vordefinierten Schnittstellen realisiert, jedoch werden nur die Anfragen der darüber liegenden Schicht beantwortet. Genauer gesagt, darf die darüber liegende Schicht die Anfragen lediglich an die darunter liegende Schicht versenden. [vgl. Co07] S. 2-3 Die Abb. 10 zeigt einen Auszug aus der Struktur dieser open-EIS Plattform, welche in den folgenden Kapiteln näher erläutert wird.

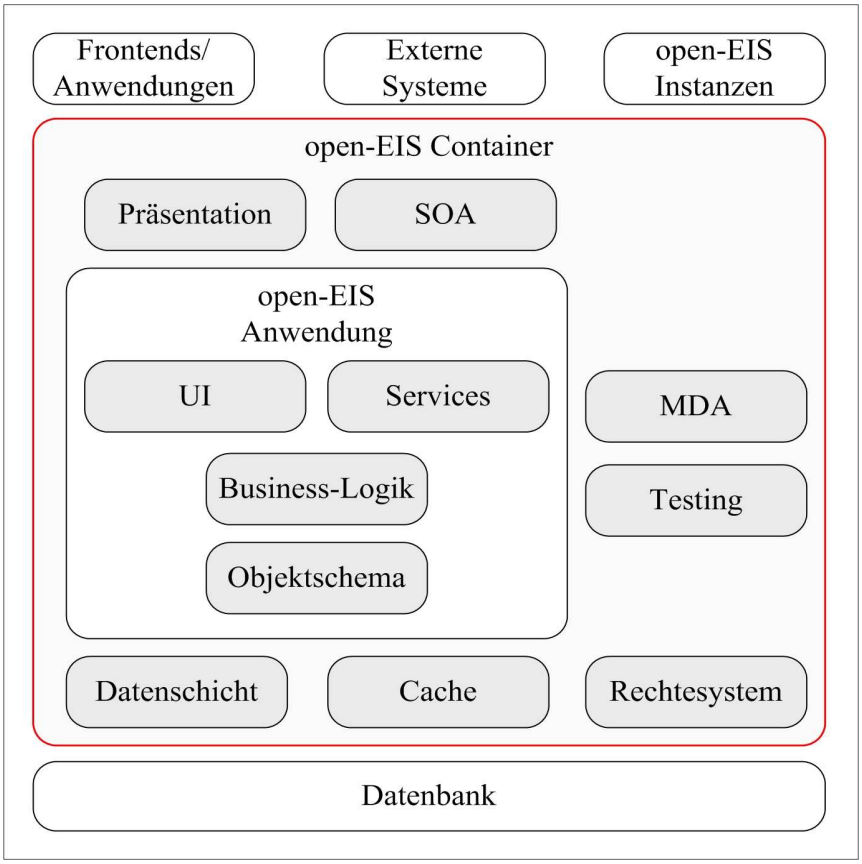

Abb. 10: open-EIS Plattform

2.4.1. Präsentationsschicht

Diese Schicht stellt die Bedienoberfläche für den Nutzer dar. Dazu können JSP und Servlets eingesetzt werden. Selbstverständlich kann auch die Erstellung der Oberfläche mit RCP (Rich Client Platform), Swing oder SWT realisiert werden. Die Verwendung der Webservices dient dem Datenaustausch mit der Logikschicht.

2.4.2. Logikschicht

Die Anwendungslogik wird auf dieser Ebene realisiert. Sie besteht im open-EIS aus mehreren Bausteinen: Rechte- und Cachemanagement, Frameworks für Datenbank (Onuris), SOA (Hermes) und Test (Sabina) sowie O/R-Mapping und DDM. Die Webservices werden auch dieser logischen Schicht zugeteilt. Im weiteren Verlauf werden diese Bausteine einzeln näher beschrieben.

Das Rechtemanagement ist ein zentrales Framework, welches für die Verwaltung der vordefinierten Rechte in den bestimmten Bereichen eines Systems verwendet wird. Dabei kann solch einem Bereich eine oder mehrere Nut-

zergruppe zugeordnet werden. Daher kann der Anwender dieser Verwaltung bei der Realisierung einer neuer Applikation schematisch die Vergabe der einzelnen Rechte für die bestimmten Bereiche des Systems durchführen und somit den Zugriff auf die einzelnen Bereiche steuern.

Um einen effektiven und schnellen Zugriff auf die Daten zu erhalten, verwendet open-EIS ein weiteres zentrales Framework, das Cachemanagement. Dieses Framework kann u.a. automatisch die zuletzt verwendeten Daten im Cache behalten. Bei der Änderung von Daten, die sich auch im Cache befinden, wird ein Validierungsprozess ausgeführt und die Daten im Cache erneuert. Beim Löschen der Daten werden die entsprechenden Daten, falls vorhanden, im Cache als zum Löschen freigegeben markiert. Falls der Cache voll wird und neue Daten in den Cache gespeichert werden sollen, so werden die die ältesten Daten oder auch die zum Löschen markierten Daten aus dem Cache entfernt. Es gibt auch andere Cachetypen, welche nicht automatisch, sondern nach anderen Kriterien ausgeführt werden können. Dies kann der Anwender bei der Realisierung seines Vorhabens festlegen.

Die Objektrelationale Abbildung *O/R- Mapping* ist eine Technik zur Speicherung von Objekten in eine relationale Datenbank. *O/R- Mapping* von open-EIS ist für die Abbildung von Objekten entwickelt worden, um dem Benutzer die Realisierung der Applikation zu erleichtern. Diese Technologie implementiert auch die Schnittstellen von dem Datenbankframework, um die unterschiedlichsten Abfragen einfach gestalten zu können.

Das Datenbankframework *Onuris* ist für das Abbilden der Daten für die bestimmten Datenbanksysteme verantwortlich. Der Benutzer von diesem Framework konzentriert sich bei seiner Arbeit auf die logisch richtigen Abfragen und braucht nicht die Funktionalitäten des jeweils darunter liegendes DB-Systems zu beachten. *Onuris* verwendet JDBC zur Verbindung mit einer Datenbank durch den Einsatz des jeweiligen JDBC- Treibers. JDBC ist eine Java-Schnittstelle zu den verschiedenen Datenbankenherstellern mit der Ausrichtung auf die relationale Datenbanken. [vgl. Ha07] S.1-3

Für das Testen der Applikation bzw. der Applikationsteilen ist ein flexibel konfigurierbares *Sabina*- Framework vorhanden, welches in *JUntit* und *TestNG* integriert werden kann. *JUnit* und *TestNG* sind die Testframeworks, welche zum Testen der Java- Applikationen entwickelt wurden. Somit können mit dem *Sabi-*

na- Framework die open-EIS- spezifischen Testfälle durchgeführt werden. Der Benutzer kann sowohl automatisierte als auch interaktive Testszenarien mit diesem Framework realisieren. Dabei kann eine Anwendung mit vielen einzelnen Tests auf die Funktionalität, Logik und die Leistung überprüft werden. [vgl. Ce07b] S. 3 Der Unterschied zwischen einem funktionalen und einem logischen Test liegt in der Komplexität der zu testenden Prozesse. Mit den funktionalen Tests können komplexere Workflow- Mechanismen überprüft werden. Die logischen Tests eignen sich für die Abdeckung eines bestimmten Sachverhaltes. Des Weiteren besitzt das *Sabina*- Framework eine Erweiterung, *Selenium*, welche für die Tests von Weboberflächen eingesetzt wird.

Dynamisches Datenmodell (DDM) ist ein Modell für ein Datenbanksystem, welches eine Abstraktionssprache MetaSQL für DDL und DML besitzt. Das DDL (Data Definition Language) ist eine Sprache für das Erzeugen, Ändern und Vernichten von Datenstrukturen. Dagegen ist DML (Data Manipulation Language) eine Sprache für das Abfragen, Einfügen, Ändern und Vernichten von Daten. Somit definiert MetaSQL die Datenstrukturen, verwaltet die Datenbank und ist für das Abfragen und Manipulieren der Daten verantwortlich. Diese Abstraktionssprache MetaSQL übersetzt die open-EIS- spezifischen Anweisungen in SQL, welche in SQL in dieser Form nicht vorhanden sind. Daher kann eine MetaSQL- Anweisung mehrere SQL- Anweisungen beinhalten. Im DDM können eine oder mehrere Tabellen und sogar ihre Beziehungen untereinander in ein Template zusammengefasst werden. Die Spalten werden dabei als Komponenten bezeichnet, um die notwendigen Spalteninformationen abzuspeichern. Ein Template definiert damit die gesamte Dokumentenstruktur in open-EIS. Das DDM erlaubt eine datenbankunabhängige Beschreibung des Datenmodells von open-EIS. Die Verwendung findet das DDM u.a. bei der Generierung von Oberflächen aus dem Datenmodell.

2.4.2.1. *Hermes*- Framework

Hermes ist ein SOA- Framework mit der Aufgabe, die vorliegenden Services zu verwalten. Die Verbindung zwischen der Server- und der Clientkomponente wird durch die fünf vordefinierten Transportadapter realisiert [vgl. Ce07a] S. 4:

- Local

- Serial
- RMI
- REST
- SOAP

Die Unterschiede zwischen diesen Adapter liegen in der Übertragungsart und –geschwindigkeit der Daten. Der *Local*- Transportadapter erlaubt die Zugriffe auf die Services über die Java- Methodenaufrufe. Somit ist dieser Adapter nur serverseitig verfügbar und bietet die höchste Datenübertragungsgeschwindigkeit. [vgl. Ce07a] S. 38 Bei der Verwendung des *Serial*- Transportadapters werden die zu übertragenden Objekte auf der Serverseite serialisiert, über das HTTP übertragen und auf der Clientseite deserialisiert. Aus diesem Grund wird dieser Adapter nur bei Verbindungen zwischen zwei Java-Komponenten eingesetzt. Die Servicezugriffe über das RMI- Protokoll werden mit Hilfe des *RMI*- Transportadapters realisiert. Der Einsatz dieses Adapters eignet sich gut für die Java-basierten Intranetumgebungen, bei welchen die Methoden des entfernten Java-Objektes aus einer anderen JVM aufgerufen werden können. [vgl. Ce07a] S. 10, [vgl. ja10] Mit dem *REST*- Transportadapter kann auf die Services mittels REST zugegriffen werden. Dabei werden Daten über das HTTP übertragen und lassen sich mit den HTTP- Operationen manipulieren. Der *SOAP*- Transportadapter erlaubt die Servicezugriffe über das SOAP (s. dazu Kapitel 2.3.3.2). Dieser Adapter wird häufig verwendet, um das Hermes- Framework in die vorhandene und komplexe IT- Szenarien zu integrieren, wo ein hoher Maß an standardbasierter Kompatibilität erforderlich ist. [vgl. Ce07a] S. 10

Nachdem die Verbindung zwischen dem Client und dem Server über einen Transportadapter hergestellt wurde, müssen die Zugriffsarten für die Services festgelegt werden. Folgende drei Arten stehen zur Verfügung:

- statische Zugriffsart
- dynamische Zugriffsart und
- dynamische XML- Zugriffsart

Die statische Zugriffsart erfordert die Serviceschnittstellen sowie alle ihre referenzierten Typen zur Kompilierungszeit. Diese werden vom Client benötigt, um die Servicefunktionalitäten nutzen zu können. Diese Art des Zugriffes findet in dieser Arbeit Verwendung, da sie die effizienteste von allen ist. Bei der dynami-

schen Zugriffsart werden keine Schnittstellen und ihre referenzierten Typen zur Kompilierungszeit gebraucht. Diese Zugriffsart bietet einen Kompromiss zwischen Flexibilität und Leistung. Hier teilt der Client dem Server mit, welche Funktionalität mit welchen Parametern ausgeführt werden soll, ohne direkt auf eine bestimmte Servicemethode zuzugreifen [vgl. Ce07a] S. 11 Die dynamische XML- Zugriffsart liefert eine automatische Überbrückung zwischen dem objektorientierten Servicevertrag auf der Basis von Java und dem dokumentenbasierten Servicezugriff über XML. Bei diesem Zugriff findet das XML-OO-Mapping statt. Diese Art des Servicezugriffes bietet dadurch ein hohes Maß an Flexibilität, weil es den Zugriff über eine plattformunabhängige, dokumentenbasierte API aus den unterschiedlichsten Umgebungen wie *Flex* oder *Silverlight* erlaubt. [vgl. Ce07a] S. 12 Der Aufbau eines Webservices wird im Kapitel 4.2 näher beschrieben.

2.4.3. Datenhaltungsschicht

Diese Schicht beinhaltet die Datenbank und in Verbindung mit dem Datenbankframework *Onuris* sind die eingesetzten Datenbanksysteme (unterstützt werden z.B. Oracle, SQL Server, PostgreSQL) leicht zu verwenden. Die Hauptaufgabe liegt hier beim Speichern und Laden der Nutzdaten. [vgl. Co07] 3-5

2.5. Vorgehensmodelle in der Softwareentwicklung

Ein Vorgehensmodell nach HINDEL „… stellt also Methoden und Elemente der Softwareentwicklung inklusive des Projektmanagements zu Prozessen und Projektphasen eines standardisierten Projektablaufs zusammen“. [Hi04] S. 14 Die Vorgehensmodelle bilden somit die Basis für die [vgl. Hi04] S. 14:

- Projektplanung
- Bewertung
- Effizienzanalyse und
- Prozessverbesserungen

Sie liefern wesentliche Erfahrungswerte für die Durchführung von Softwareprojekten und erleichtern die Planung sowie die Verbesserung von Abläufen bei diesen Projekten. [vgl. Hi04] S. 24

Der Einsatz von Vorgehensmodellen ist nicht bei jedem Projekt sinnvoll. Bei kleinen Projekten, welche einen überschaubaren Zeitraum von drei bis vier Wochen umfassen, und mit einem 2-3 Personenteam realisiert werden, besteht keine Notwendigkeit für den Einsatz von Vorgehensmodellen. [vgl. Ve00] S. 26 Bei solchen Projekten ist eine gute Kommunikation zwischen den Mitarbeitern sowie eine schnelle Anpassung des Entwurfes und der Implementierung auf die Veränderungen der Anforderungen zu erwarten. Weiterhin sollte berücksichtigt werden, dass die Einführung eines Vorgehensmodells mit Zeitaufwand und damit Kosten verbunden ist.

Im Folgenden werden in kurzer Form die Vorgehensmodelle der letzten Jahre angerissen, welche maßgebend die Softwareentwicklung geprägt haben. [vgl. Ve00] S. 27-28

2.5.1. Wasserfallmodell

Für die Softwareentewicklung war eines der ersten Vorgehensmodelle ein Wasserfallmodell. Bei diesem Modell wird die gesamte Entwicklung in die einzelnen zeitlich aufeinander folgenden Phasen eingeteilt, welche bei der Abarbeitung sequenziell durchlaufen werden. Zwischen diesen Phasen sind keine Rückkoppelungen erlaubt. [vgl. Hi04] S. 14 In der Praxis kann jedoch nicht sichergestellt werden, dass die festgelegten Anforderungen im Laufe des gesamten Projektes unverändert bleiben oder auch ein Entwurf nicht mit der vorgeschriebenen Effizienz implementiert werden kann. Das Wasserfallmodell kann u.a. aus diesen Gründen erweitert werden, sodass bei eintretenden Änderungen der Anforderungen zurück zur Anforderungsanalyse gesprungen werden kann. Dabei werden die übrigen Phasen angehalten, bis die Analysephase vollständig abgeschlossen wird. Die Rückkoppelungen zwischen jeweils zwei aufeinander folgenden Phasen sorgen für mehr Flexibilität bei der Realisierung eines Projektes. Diese Rückkoppelungen werden in diesem erweitertem Modell nicht als reguläre Iterationen, sondern als Ausnahmen verstanden. [vgl. Me04] S. 93-94

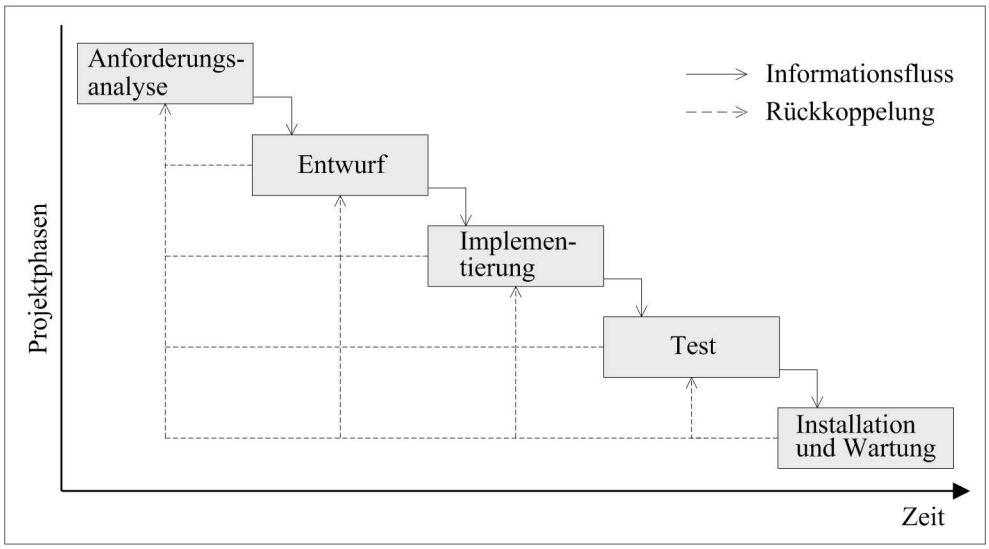

Abb. 11: Wasserfallmodell mit Rückkoppelungen

[vgl. Me04] S. 93

Die Vorteile des Wasserfallmodells ohne die Rückkoppelungen liegen in der einfachen Verständlichkeit, guter organisatorischen Beherrschung sowie verringertem Managementaufwandes. Des Weiteren kann der Prozessablauf durch die Einführung von Meilensteinen und Dokumenten innerhalb und am Ende jeder Phase besser kontrolliert werden. Zu den Nachteilen zählen unter anderem das streng dokumentierte Vorgehen sowie die Erkennung von Risiken erst in der Implementierungs- oder Testphase. Die Veränderungen bzw. die Detaillierungen der Anforderungen zu einem späteren Zeitpunkt können nicht mehr berücksichtigt werden. Mit der Testphase kann nur nach der kompletten Fertigstellung der Implementierung begonnen werden, was auch als großer Nachteil angesehen wird. Durch die überwiegenden Nachteile kann das Wasserfallmodell nur in Erwägung gezogen werden, wenn es sich um stabile Projekte handelt, die nach einem einheitlichen Ablauf durchgeführt werden müssen. Ebenfalls dürften sich die Anforderungen von diesen Projekten in dessen Verlauf nicht ändern. [vgl. Hi04] S. 15

2.5.2. V-Modell

Das V-Modell ist eine Erweiterung des Wasserfallmodells, bei welchem die Qualitätssicherung im Vordergrund steht. Bei diesem Modell werden nach jeder Phase entsprechende Maßnahmen zur Qualitätssicherung durchgeführt. Au-

ßerdem wurden zwei weitere Begriffe in dieses Modell eingeführt [vgl. Hi04] S. 16:

- **Verifikation**: beschreibt einen Prozess, welcher sicherstellt, dass ein Produkt seinen Anforderungen entspricht. Hier findet dementsprechend eine Überprüfung statt (rechter Zweig der Abb. 12), ob das Produkt nach seiner Spezifikation entwickelt wurde (linker Zweig der Abb. 12).
- **Validierung**: beschreibt einen Prozess, welcher dokumentiert, dass ein Produkt der vorgesehenen Verwendung in der geplanten Wirkumgebung entspricht.

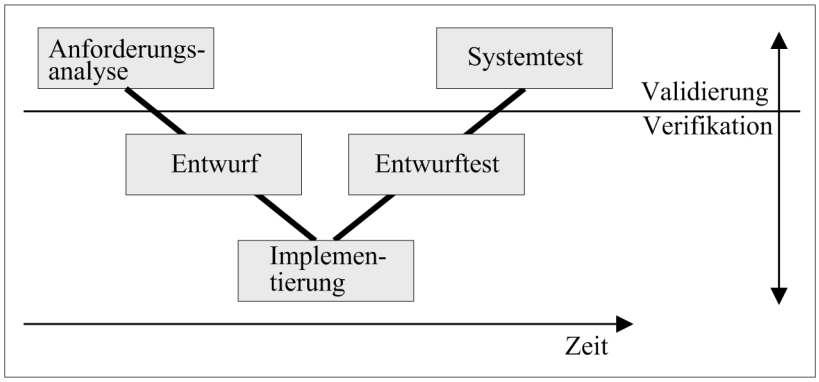

Abb. 12: V-Modell

[vgl. Me04] S. 93

Bei dem V-Modell können im Vergleich zum Wasserfallmodell bereits sehr früh die Testfälle charakterisiert werden. Sobald die Anforderungsanalyse fertig gestellt wird, kann anschließend der Systemtest definiert werden. Nach der vollständigen Konzeption des Entwurfes wird der Entwurftest durchgeführt. Dieses Modell eignet sich somit gut für Projekte, welche einen hohen Grad an Qualitätssicherung verlangen.

2.5.3. Spiralmodell

Das Spiralmodell ist eine weitere Variante des Wasserfallmodells, welches auch mehr Flexibilität anbietet. Dieses Modell kann eingesetzt werden, wenn eine zuverlässige Projektplanung und Kontrolle erforderlich ist, da dies die großen Schwächen des Wasserfallmodells sind. Der Software-Entwicklungsprozess

wird in diesem Modell als ein iterativer Prozess angesehen, welcher zum Teil die Grundlage für das Rational Unified Process lieferte. [vgl. Ve00] S. 30

Die Spirale stellt eine Zeitachse dar und jede Phase (Teilaufgabe) entspricht einer vollständigen Windung dieser Spirale, auch Zyklus genannt, wobei die erste Phase im Inneren der Spirale beginnt. Die nächste Phase startet nach der Fertigstellung der vorhergehenden Phase ohne zeitliche Überlappung usw. Innerhalb eines Zyklus treten folgende Aktivitäten auf [vgl. Me04] S. 90, [vgl. Fe04] S. 18:

- Festlegen von Zielen, Alternativen und Rahmenbedingungen
- Bewertung der Alternativen, Identifizierung der Risiken und Lösung der Probleme
- Weiterentwicklung, Verifikation
- Planung der nächsten Phase und
- Aufgabenverteilung

Am Ende jedes Zyklus findet eine Überprüfung statt, welche die Bewertung des aktuellen Projektfortschrittes vorgesehen ist. Entweder werden die Planung für die nächste Phase sowie die Festlegung der Ressourcen durchgeführt oder die Projektentwicklung abgebrochen. [vgl. Ve00] S. 30

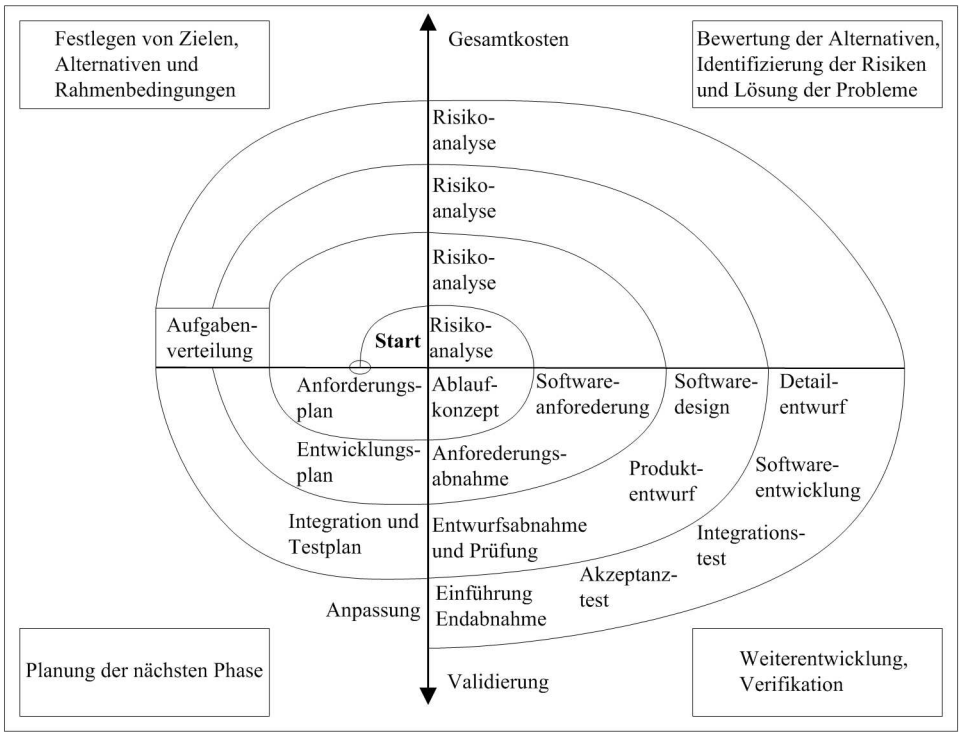

Abb. 13: Spiralmodell

[vgl. Fe04] S. 18

2.5.4. Rational Unified Process - RUP

Mit dem Rational Unified Process wurde ein Modell entwickelt, welches die objektorientierte Softwareentwicklung mit UML (s. Kapitel 3) unterstützt. [vgl. Hi04] S. 19 Dieses Modell ist eigentlich keine Neuentwicklung, sondern eine konsequente Weiterentwicklung des Wasserfallmodells mit integrierten Elementen aus dem Spiralmodell. Der RUP wird in die einzelnen Phasen eingeteilt, welche ihrerseits aus den Iterationen bestehen. Die unten aufgeführten Phasen decken den kompletten Softwarezyklus ab [vgl. Fe04] S. 18:

- Initialisierungsphase (Inception)
- Entwurfsphase (Elaboration)
- Implementierungsphase (Construction)
- Übergangsphase (Transition)
- Echtbetriebsphase (Production)
- Abschlussphase (Retirement)

Die Aktivitäten der s.g. Workflows verteilen sich nicht sequentiell, sondern auf alle Phasen nur mit unterschiedlicher Gewichtung. Auf der linken Seite werden die Prozessaktivitäten und auf der rechten Seiten die Unterstützungsaktivitäten vorgestellt, welche im RUP verwendet werden [vgl. Ve00] S. 39-40:

- Geschäftsprozessmodellierung
- Anforderungsmanagement
- Analyse und Entwurf
- Implementierung
- Test
- Verteilung
- Konfigurationsmanagement
- Projektmanagement
- Einführung einer Werkzeugumgebung

Die Vorteile des RUP liegen in der Flexibilität des Modells, Beseitigung von Fehlern und ungeeigneten Alternativen sowie Unterstützung der parallelen Aktivitäten. Weiterhin eignet sich gut der RUP für die objektorientierte Softwareentwicklung durch den Einsatz von UML. Der hohe Managementaufwand, welches durch das Treffen neuer Entscheidungen über den weiteren Prozessablauf entsteht, zählt zu den Nachteilen dieses Modells. Ebenso ist mit mehr Aufwand bei der Planung der Iterationen einzukalkulieren.

Der Einsatz von RUP ist bei der inkrementeller Softwareentwicklung sinnvoll, bei der die fertig gestellten Teilaufgaben das gesamte System um ein wei-

teres Inkrement erweitern. Der RUP wird auch bei den Projekten eingesetzt, welche auf große Bürokratie verzichten und dadurch schnelle Entscheidungen bei den Lösungsalternativen treffen wollen. [vgl. Hi04] S. 20-21

2.5.5. Zusammenfassung

Alle vorgestellten Modelle wurden für ein gemeinsames Ziel, die Verbesserung der Koordination des Softwareentwicklungsprozesses, entwickelt. Die erzielten Erfolge dieser Modelle sind dagegen recht unterschiedlich. Das Wasserfallmodell hat in seiner Gesamtheit versagt, wobei es für die einzelnen Iterationen im RUP durchaus geeignet wäre. Das V-Modell schaffte es nicht die hohe Papierflut zu dämmen und mehr Pragmatismus mit weniger Bürokratie durchzusetzen. Der hohe Managementaufwand des RUP kann seine Vorteile nicht in den Schatten stellen. So können beim Einsatz dieses Modells gut koordinierbare Projekte mit einem hohen Grad an Flexibilität erreicht werden. [vgl. Ve00] S. 43-44

Ausgehend von den Ergebnissen dieser Modelle wurde für die Realisierung dieser Diplomarbeit, auch mit einem Projekt zu vergleichen, das Wasserfallsmodell mit Rückkoppelungen gewählt. Durch eine klare Vorgabe der Diplomarbeitstruktur, fehlende Kommunikationskonflikte und wegen dem 1-Mann-Projekt konnten die einzelnen Phasen ohne größere Risiken sequentiell abgearbeitet und gleichzeitig dokumentiert werden. Die Rückkoppelungen wurden bei auftretenden Änderungen der Anforderungen für die Sprünge zu den vorhergehenden Phasen eingesetzt. Auch in der Testphase wurden diese Rückkoppelungen verwendet, falls die bereits gelieferten Ergebnisse nicht den gestellten Systemanforderungen entsprachen.

3. Anforderungsanalyse

In diesem Kapitel werden als Basis jeder Analyse die Rahmenbedingungen gesetzt, welche die Umgebung eines Systems durchleuchten sollen. Anschließend erfolgt eine exakte Definition der gestellten Anforderungen an das System. Dabei werden die Anforderungen in zwei Arten, in funktionale und nicht-funktionale Anforderungen, gegliedert. Bei den funktionalen Anforderungen werden solche behandelt, welche die Frage nach dem *was* ein System leisten soll beantwortet. Dieser Bereich wird in diesem Kapitel ausführlich behandelt. Hier findet auch eine Unterteilung der Anforderungen aus der Nutzer- sowie aus der Administratorsicht statt. Im Gegensatz dazu beschäftigen sich die nicht-funktionalen Anforderungen mit der Frage *wie gut* ein System arbeiten soll, welche ebenfalls im Kapitel der Anforderungsanalyse behandelt werden.

Nach VERSTEEGEN wird eine Anforderung von einem Auftraggeber oder einem Endbenutzer gestellt und sie beschreibt die Verhaltensweise des zu implementierenden Systems. [vgl. Ve00] S.25 Zum besseren Verständnis werden in der Anforderungsanalyse diese Systemanforderungen durch geeignete Diagramme visualisiert. Dazu wird die vereinheitliche Modellierungssprache UML (Unified Modeling Language) eingesetzt, welche u.a. für die Dokumentation der Analyse objektorientierter Software entworfen wurde. [vgl. Se06] S. 2

3.1. Rahmenbedingungen

Das in dieser Arbeit zu entwickelte DMS wird nach den Prinzipien einer serviceorientierten Architektur aufgebaut. Es basiert auf der open-EIS Plattform unter Verwendung des Hermes- Frameworkes, welches eine transaktionsbasierte und flexible SOA- Architektur für das open-EIS darstellt. Dadurch besteht keine feste Kopplung zwischen der Anwendungslogik und der UI, was zu einer enormen Flexibilität im Sinne der Erweiterbarkeit führt. Die Bereitstellung der Dienste sowie deren Schnittstellen für die Kommunikation mit dem DMS setzen die Schwerpunkte an das zu entwickelnde DMS- Backend, wobei die grafische Benutzerschnittstelle (Frontend) nicht Bestandteil dieser Arbeit sein wird. Das Rechtemanagement, auch DRM genannt, des Dokumenten-Management-Systems wird nicht neu konzipiert und entworfen, weil open-EIS bereits eine

Rechteverwaltung besitzt und sie in dieser Arbeit eingesetzt wird. Und auch die Nutzer des DMS sind den Nutzern des open-EIS Systems gleichgestellt. Als Entwicklungsumgebung wird bei dieser Arbeit die Eclipse- Plattform mit den notwendigen Erweiterungsmodulen (Tomcat, Quantum DB) eingesetzt.

3.2. Funktionale Anforderungen

Bei der Ermittlung der funktionalen Anforderungen wird das Ausmaß der Funktionalitäten eines Systems untersucht, aufgelistet und detailliert beschrieben. [vgl. fr10a] Das zu entwickelnde Backend-System konzentriert sich auf die Bereiche Eingabe, Verwaltung und Recherche von Dokumenten, welche für die Realisierung des Funktionsumfanges aus der Nutzersicht notwendig sind. Die Manipulation von

- Dokumenten
- Mediadateien
- und deren Metainformationen

beschränken sich auf die primären Funktionalitäten, u.a. das Erstellten, Bearbeiten und Löschen. Unter einem Dokument wird bei dieser Arbeit eine textbasierte *Passage* verstanden. Bei einer Mediadatei kann es sich um eine formatunabhängige Datei (z.B.: JPEG, PDF) handeln, welche genauso wie ein Dokument einen Informationsstand speichern soll. Ein Dokument und eine Mediadatei sollen in Verbindung zu einander gesetzt werden können, um nach außen als eine logisch zusammenhängende Einheit zu wirken. Zusätzlich dazu müssen ein Dokument und eine Mediadatei einen Zustand besitzen, welcher die *Sichtbarkeit* dieser im System steuert. Um die Dokumentenverwaltung zu realisieren, soll der Nutzer beim Erstellen eines Dokumentes die Verfügungsbereiche auswählen können, welche die konkrete Position des Dokumentes in der Ablage definieren. Aus diesem Grund soll es dem Nutzer möglich sein, solche Bereiche im Vorfeld zu erstellen und zu einem späteren Zeitpunkt auch diese zu bearbeiten bzw. zu löschen. Die Recherche nach den Dokumenten soll sowohl über ihren Standort (Verfügungsbereiche), als auch über bestimmten Kriterien von Dokumenten erfolgen. Das bedeutet, dass alle Dokumente eines solchen Systems eigene Attribute und Suchbegriffe erhalten werden. Je mehr Suchbegriffe für ein Dokument vorhanden sind, desto präziser werden die Ergebnisse bei der Suche nach die-

sem ausfallen. Ebenfalls sollen dem Nutzer diese erwähnten Funktionalitäten für die Verwaltung der Mediadateien zur Verfügung gestellt werden.

Aus der Administratorsicht liegt der Fokus in den Aufgaben der Nutzer- und Rechteverwaltung. Der Administrator soll die Nutzer des Systems, ihre Gruppierungen sowie die Rechtevergabe verwalten. Um das bestehende Rechtemanagement von open-EIS zu verwenden, sollen die Kategorien aus open-EIS zur Abgrenzung der Verfügungsbereiche eingesetzt werden. Im Folgenden wird der Begriff *Kategorie* für den Verfügungsbereich benutzt. Die Abb. 14 stellt die allgemeinen Anforderungen aus der Nutzersicht sowie aus der Administratorsicht dar.

Dokument/ Mediadatei und Kategorie (Nutzersicht)	**Nutzer und ihre Rechte (Administratorsicht)**
Erstellen	Nutzer/ Gruppe erstellen
Suchen	Nutzer/ Gruppe suchen
Bearbeiten	Nutzer/ Gruppe bearbeiten
Löschen	Nutzer/ Gruppe löschen
Dokumente und Mediadateien verbinden	Nutzer der Gruppe zuordnen
	Gruppe der Kategorie zuordnen
	Suchbegriffe verwalten

Abb. 14: funktionale Anforderungen an DMS- Backend

Die Abb. 15 und Abb. 16 zeigen zusammengefasst die aufgezählten Funktionen aus den beiden Sichten in den Anwendungsfalldiagrammen, welche in dem Backend-System zu realisieren sind. An dieser Stelle ist anzumerken, dass diese Funktionen nur ausgeführt werden sollen, wenn der jeweilige Benutzer die entsprechende Rechte im Dokumenten-Management-System besitzt.

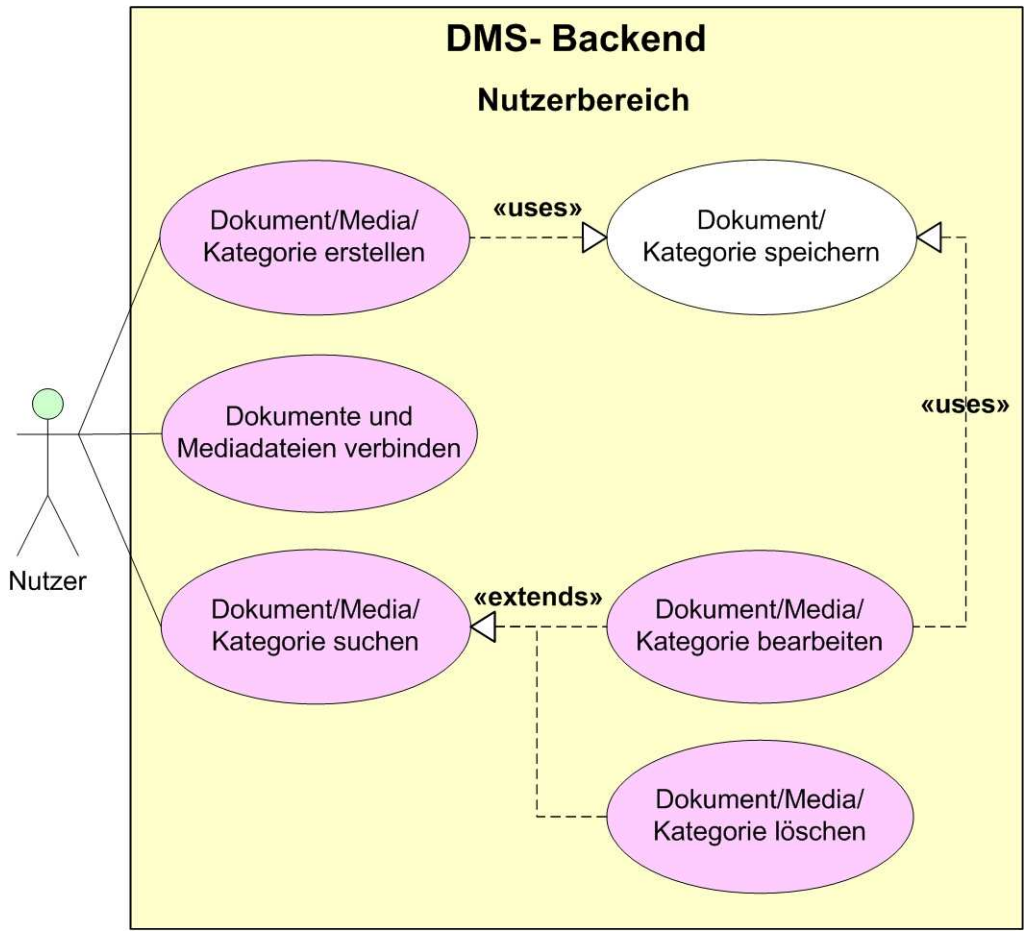

Abb. 15: Grundanforderungen an DMS – Nutzerbereich (Anwendungsfalldiagramm)

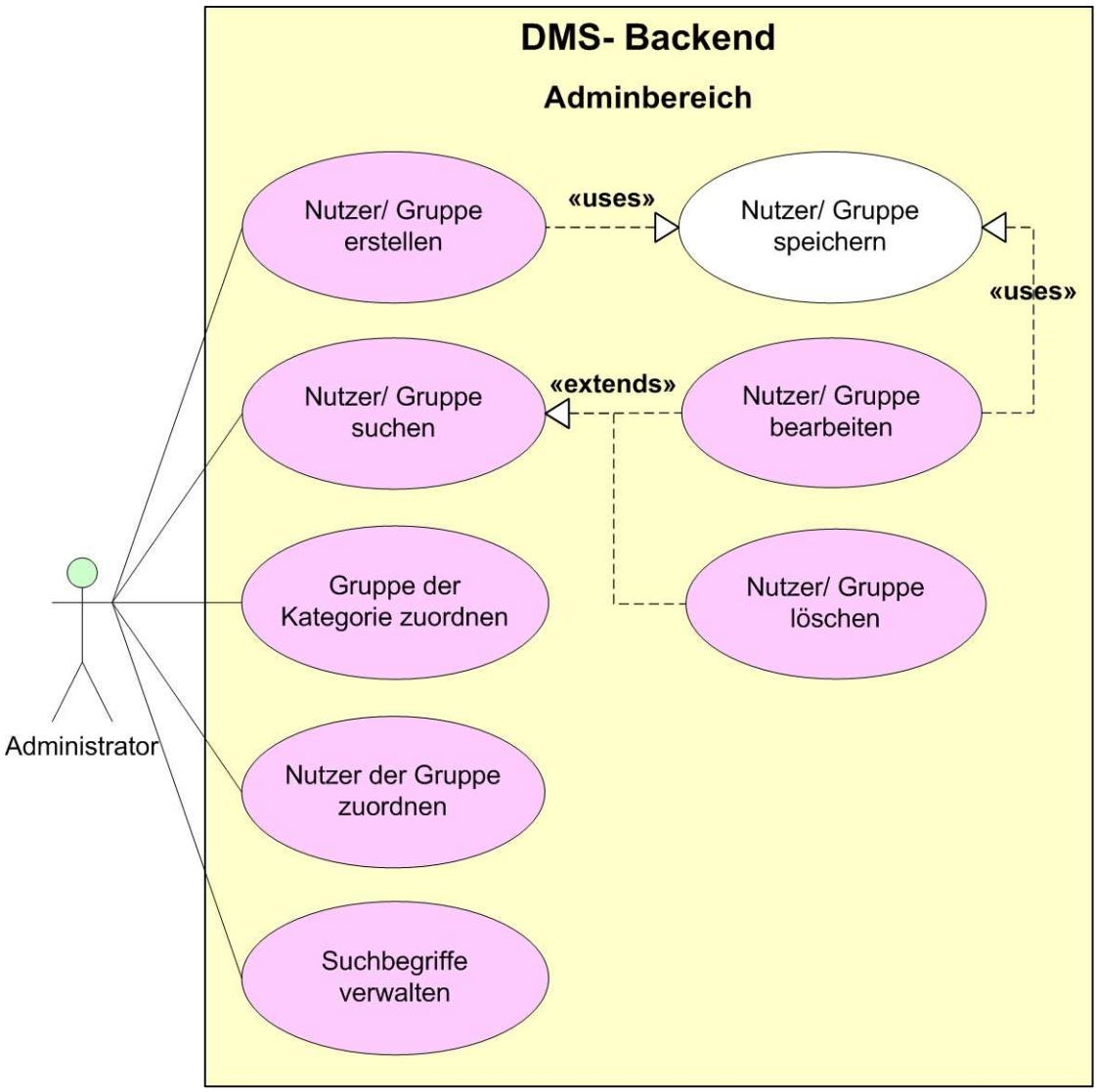

Abb. 16: Grundanforderungen an DMS – Adminbereich (Anwendungsfalldiagramm)

Die Aktivitäten des gesamten Systems werden in den folgenden Aktivitätsdiagrammen (Abb. 17 und Abb. 18) wieder in zwei Sichten zusammengefasst. Daraus können die einfachen Aktionen in Verbindung zueinander gestellt werden.

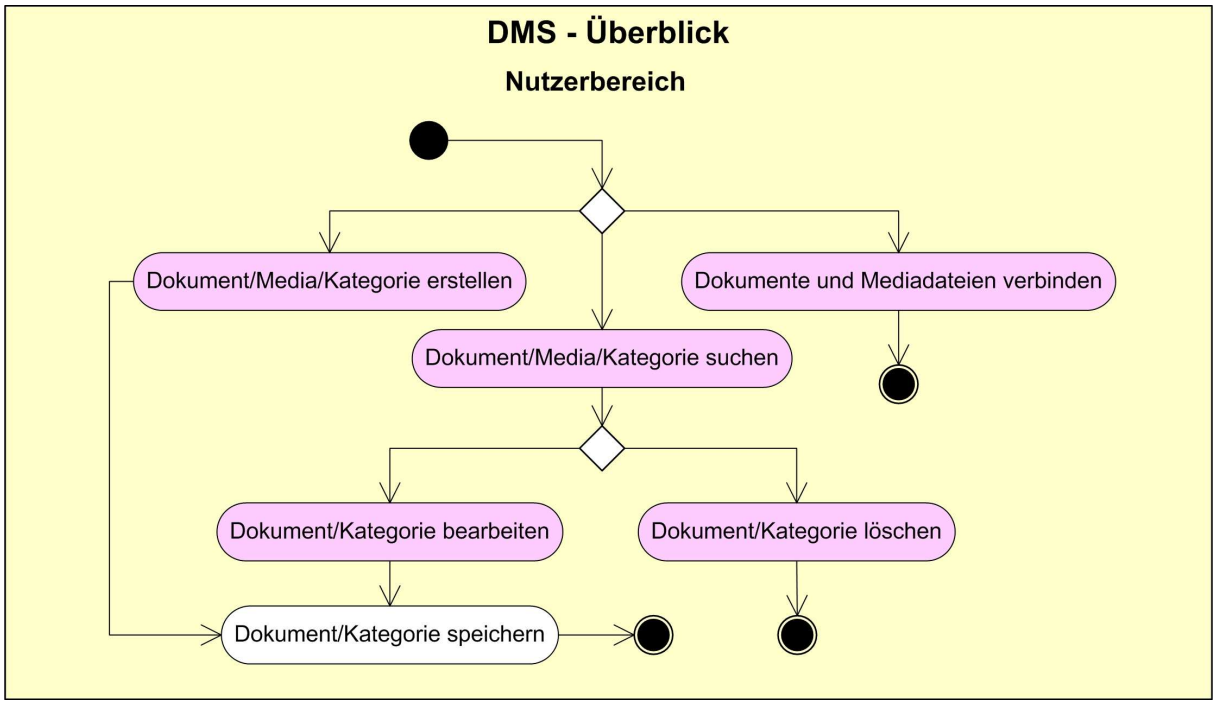

Abb. 17: Grundanforderungen an DMS – Nutzerbereich (Aktivitätsdiagramm)

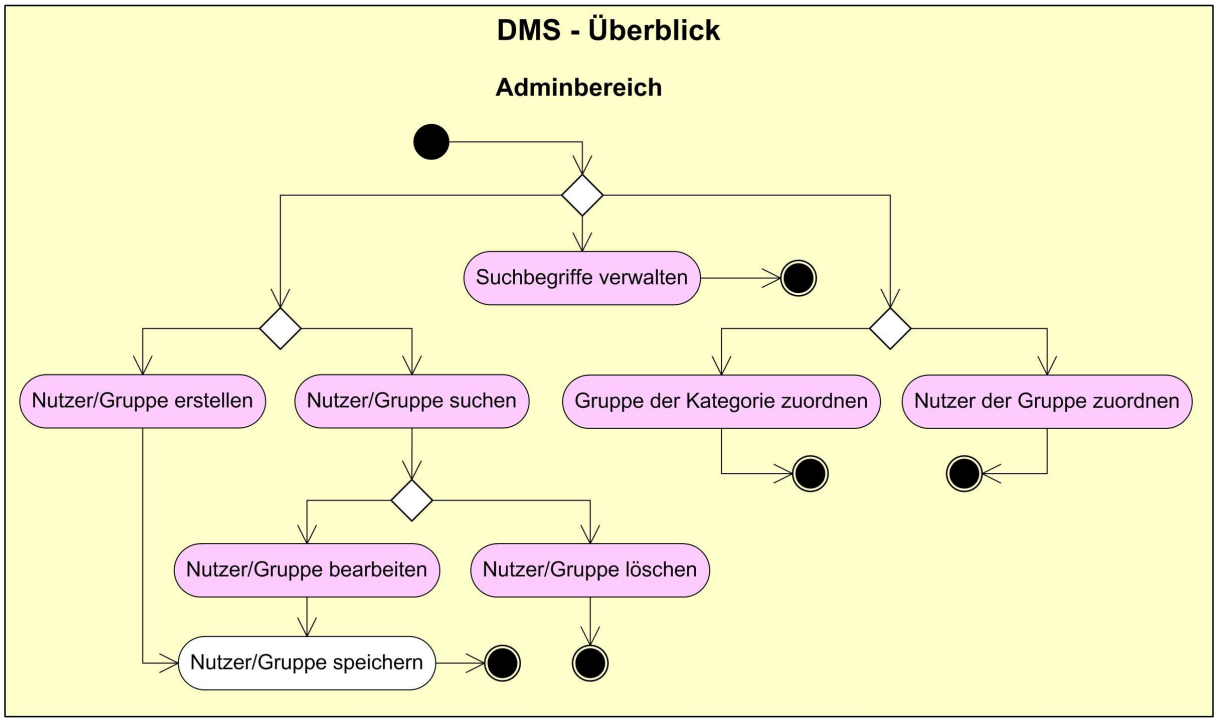

Abb. 18: Grundanforderungen an DMS – Adminbereich (Aktivitätsdiagramm)

Im weiteren Verlauf werden zuerst die einzelnen Anforderungen aus dem Nutzerbereich und anschließend aus dem Administratorenbereich näher erläutert sowie mit den passenden Diagrammen veranschaulicht.

3.2.1. Nutzerbereich

Wie bereits angedeutet, werden die funktionalen Anforderungen aus zwei Sichten betrachtet. In diesem Kapitel werden die Anforderungen aus der Nutzersicht behandelt, welche die Bereiche der Eingabe, Verwaltung und Recherche von Dokumenten umfassen.

3.2.1.1. Dokument erstellen

Der Nutzer des Systems soll in der Lage sein, durch die Eingabe des Inhaltes und des Titels, ein Dokument im System zu erstellen. Falls durch den Nutzer weitere Daten in Verbindung zu diesem Dokument, welche z.B. nicht in der Textform vorliegen, in das System hinzugefügt werden sollen, so können diese Daten als eine externe Datei (Media) mit dem Dokument verbunden werden. Entweder wird eine bereits vorhandene Datei aus dem System verwendet oder eine neue in das System hochgeladen. Die Indexierung, Vergabe der Suchbegriffe, soll beim Erstellen des Dokumentes manuell, automatisch oder auch computergestützt erfolgen. Für die manuelle Indexierung soll der Nutzer selbst verantwortlich sein, wobei zu beachten ist, dass durch diese Zuweisung der passenden Suchbegriffe das Wiederfinden der Dokumente im System stattfindet. Bei der automatischen Indexierung sollen anschließend einige Suchbegriffe dem erstellten Dokument aus seinen Attributen hinzugefügt werden, u.a. der Autor und der Titel des Dokumentes. Der Einsatz der computergestützten Indexierung soll dem Nutzer das Hinzufügen der passenden Suchbegriffe erleichtern. Bei dieser Methode der Verschlagwortung werden die passenden Suchbegriffe automatisch ermittelt, durch den Nutzer manuell ausgewählt und erst dann in das System hinzugefügt. Des Weiteren soll ein Sichtbarkeitszustand dem erstellten Dokument vom Nutzer zugewiesen werden. Als mögliche Zustände sind die Markierungen *in Bearbeitung* (Dokument nicht komplett) oder *Fertig gestellt* (Dokument ist komplett). Der Vorteil dieser Markierungen liegt in der detaillierten Selektion bei der Suche nach den Dokumenten. Zum Schluss ist durch den Nutzer einzugrenzen, für welche Kategorien das Dokument den Nutzern des Systems zur Verfügung gestellt werden soll. Zur Auswahl stehen dem Nutzer jedoch nur die Bereiche, für welche er mindestens das Schreibrecht

besitzt. Somit legt der Nutzer keine Reche auf das Dokument fest, sondern weist nur das Dokument den jeweiligen Bereichen zu.

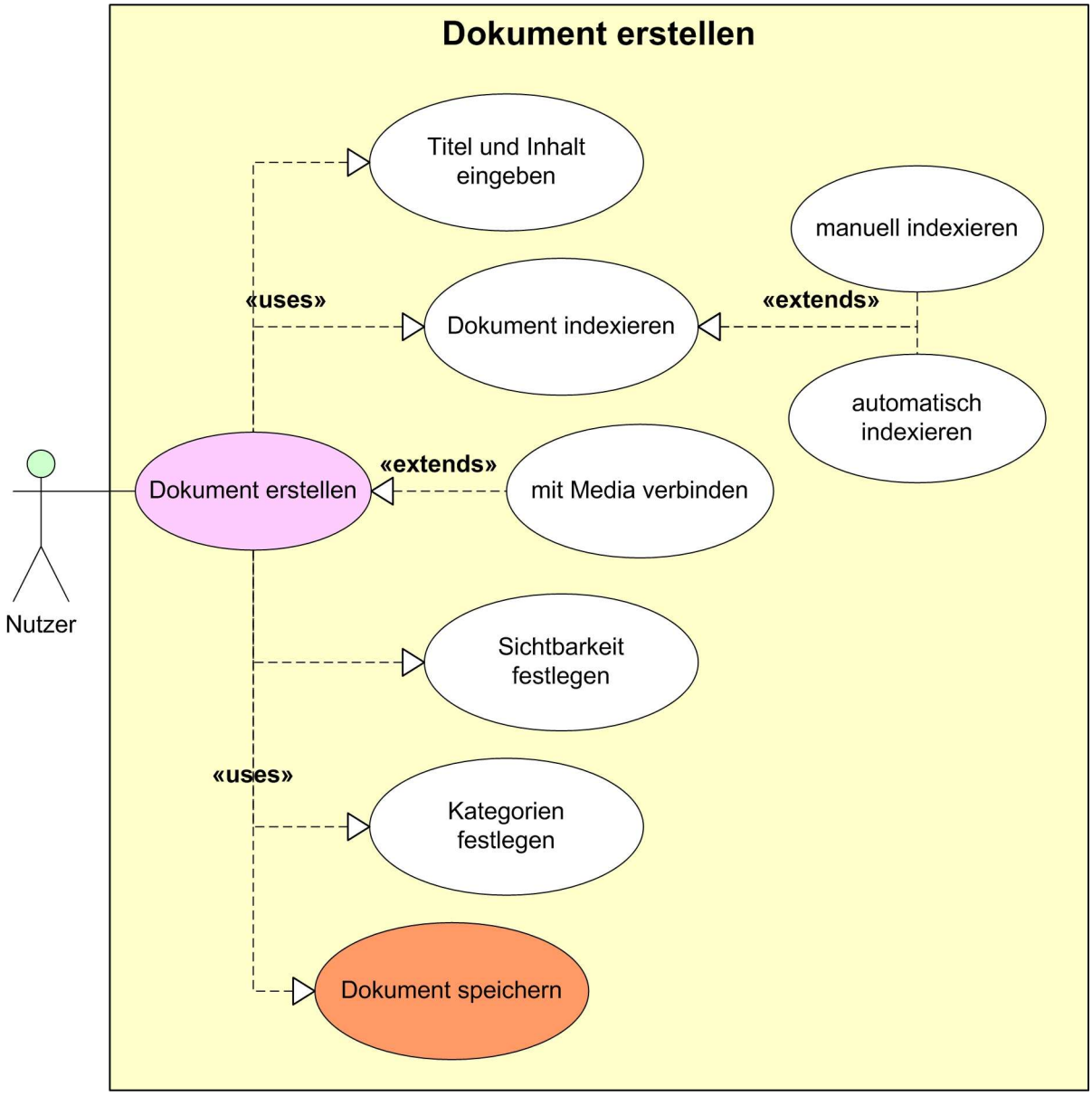

Abb. 19: funktionale Anforderung *Dokument erstellen* im Anwendungsfalldiagramm

Wie schon zu erkennen ist, wird beim Erstellen von Dokumenten in einem System eine Menge an Informationen benötigt. Wie diese Daten samt den dazu notwendigen Aktionen miteinander vernetzt sind, wird im übersichtlichen Aktivitätsdiagramm dargestellt.

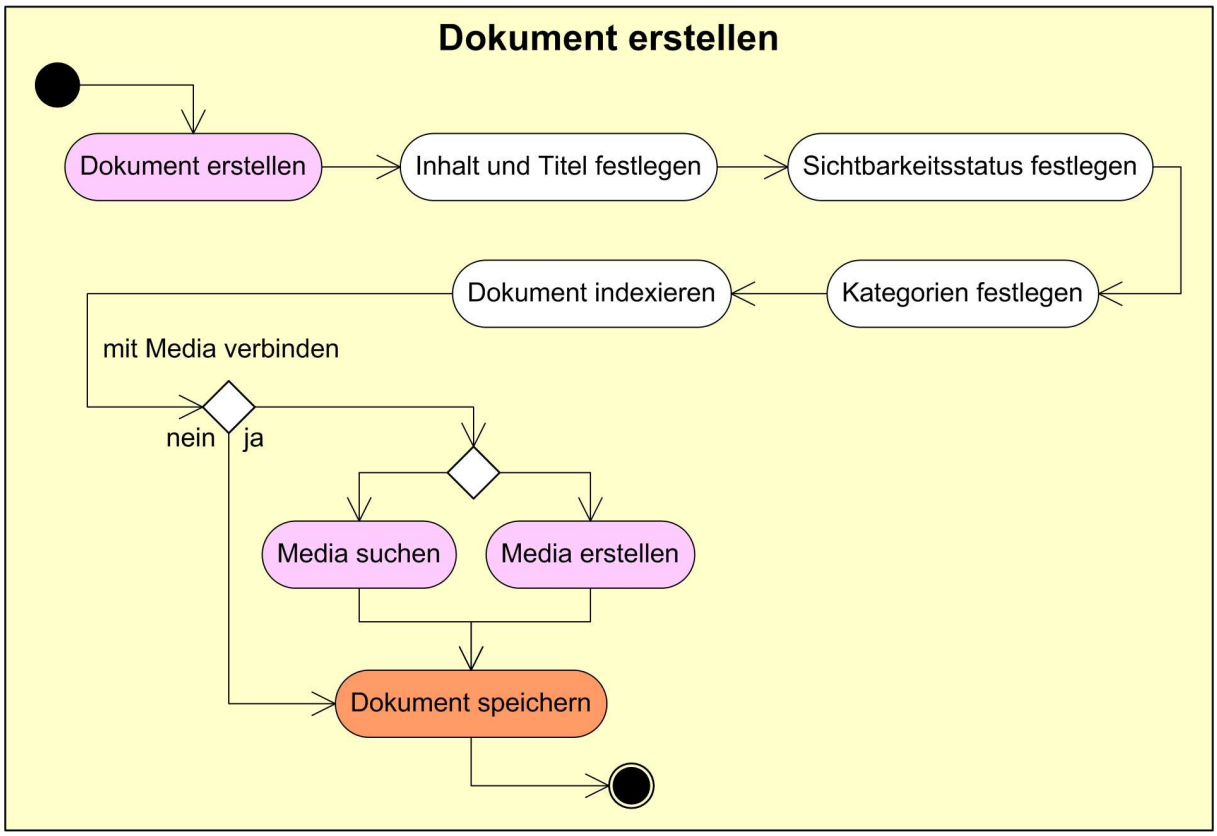

Abb. 20: funktionale Anforderung *Dokument erstellen* im Aktivitätsdiagramm

3.2.1.2. Media erstellen

Die Informationen, welche nicht als Dokument im System abgelegt werden können, werden in einer Mediadatei gespeichert. Die Mediadatei besitzt einen Titel, einen Untertitel sowie den alternativen Text für die Kennzeichnung des Inhaltes. Die Indexierung erfolgt ähnlich wie bei dem Dokument. Der Unterschied liegt darin, dass hier keine computergestützte Indexierung möglich ist. Genauso wie die Dokumente werden die Mediadateien in die Kategorien eingeteilt und erhalten vom Nutzer den Sichtbarkeitsstatus.

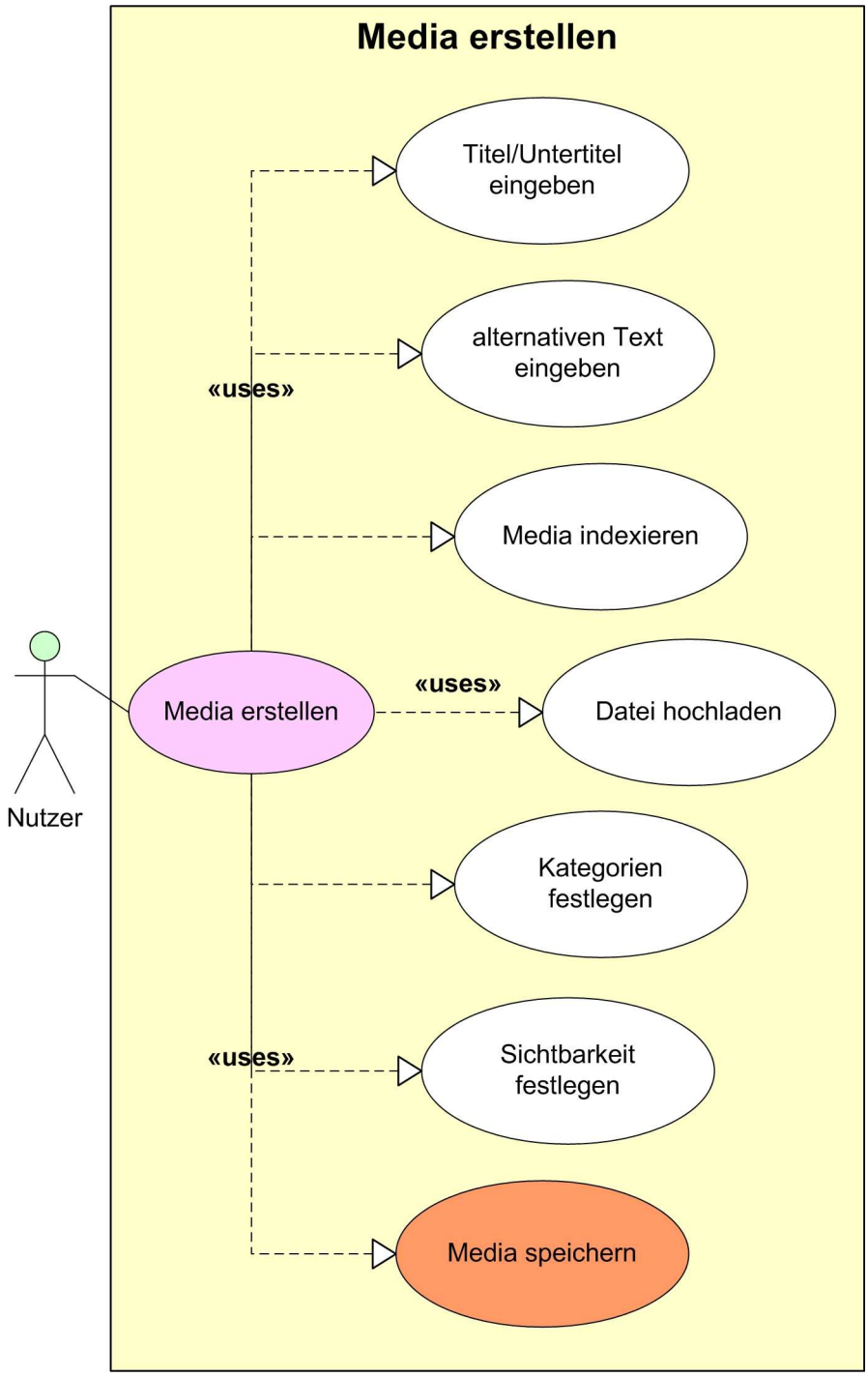

Abb. 21: funktionale Anforderung *Media erstellen* im Anwendungsfalldiagramm

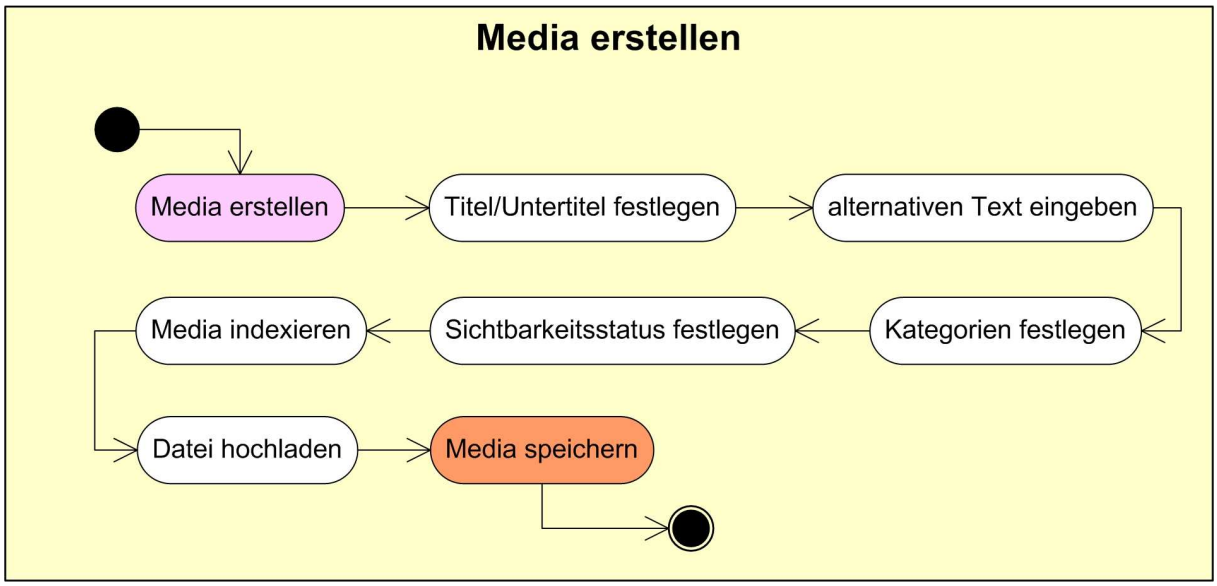

Abb. 22: funktionale Anforderung *Media erstellen* im Aktivitätsdiagramm

3.2.1.3. Kategorie erstellen

Für die Platzierung eines Dokumentes bzw. einer Mediadatei im System soll der Nutzer einen Bereich festlegen. Dazu können z.B. vorhandene Kategorien verwendet werden. Der Nutzer soll aber auch eigene Kategorien anlegen können. Dazu müssen der Titel der Kategorie sowie ihre Oberkategorie angegeben werden. In diesem System darf eine Kategorie nur eine Oberkategorie besitzen. Zudem muss der Nutzer das Recht besitzen, unterhalb der angegebenen Oberkategorie die neue Kategorie zu erstellen. Die Indexierung der Kategorie erfolgt genauso wie bei dem Dokument bzw. der Mediadatei.

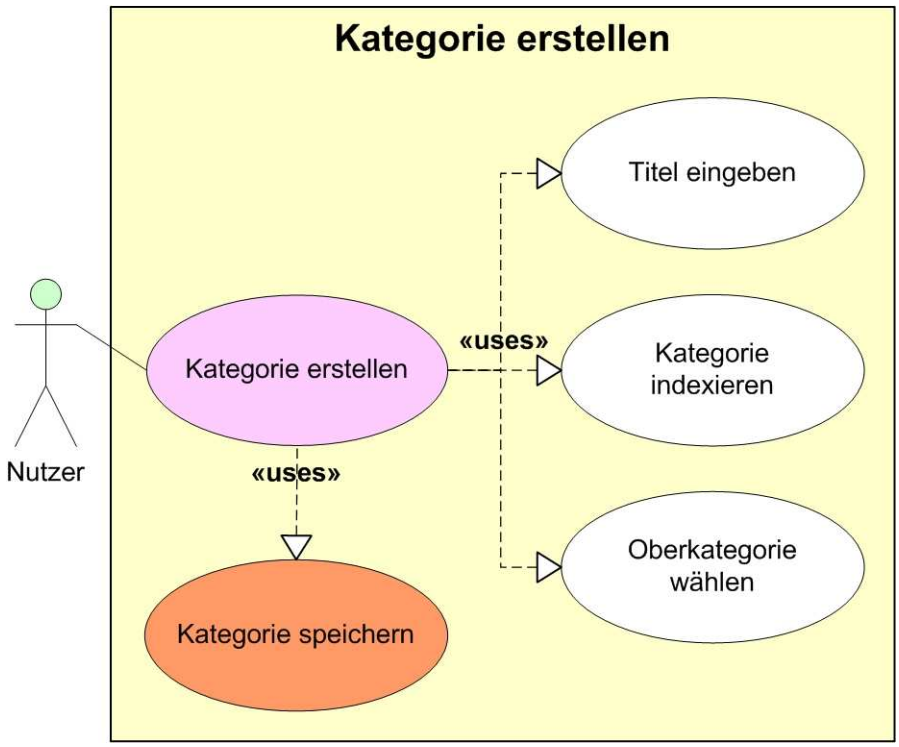

Abb. 23: funktionale Anforderung *Kategorie erstellen* im Anwendungsfalldiagramm

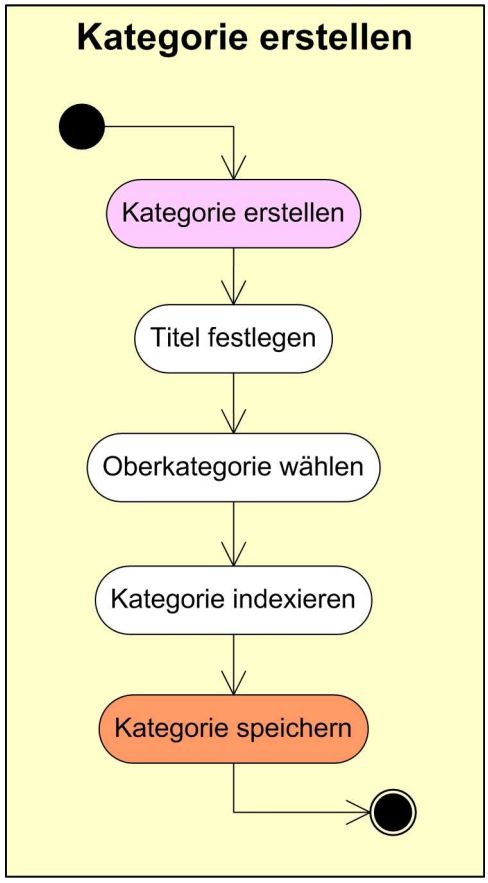

Abb. 24: funktionale Anforderung *Kategorie erstellen* im Aktivitätsdiagramm

3.2.1.4. Dokument und Media verbinden

Die Dokumente und die Mediadateien sollen in Bezug auf den Informationsgehalt den gleichen Stellenwert besitzen. Sie liegen sozusagen auf der gleichen Ebene. Aus diesem Grund können die Dokumente mit den Mediadateien verbunden werden, um eine logische Einheit darzustellen. Somit kann der Nutzer alle mit dem Dokument in Verbindung stehende Mediadateien anzeigen lassen. Analog ist die Verbindung von Mediadatei zu den Dokumenten realisierbar. Voraussetzung ist hierfür das Vorhandensein der erforderlichen Rechte im System. Es ist durchaus möglich, dass ein Nutzer nur einen Teil der verbundenen Elemente und ein anderer Nutzer alle verbundenen Elemente angezeigt bekommt. Dies ist von den Rechten des jeweiligen Nutzers im System abhängig. Diese Verbindungen sollen *leichtgewichtig* und ungerichtet sein. Beim Entfernen dieser Verbindung werden weder die Dokumente noch die Mediadateien, welche miteinander verbunden sind, aus dem System entfernt. Durch die ungerichtete Verbindung ist es irrelevant, von wem zu wem diese Verbindung erzeugt wurde.

3.2.1.5. Dokument bzw. Media suchen

Die Suche bzw. die Recherche ist eine der wesentlichen funktionalen Anforderungen an das DMS. Über die Suche lassen sich die im System befindlichen Dokumente an den Nutzer ausliefern. Dabei soll der Nutzer frei wählen können, ob er die Suche nach Dokumenten oder Mediadateien durchführen möchte. Natürlich soll auch eine Suche für beiden Arten gleichzeitig vorhanden sein. Das System soll die Schlüsselwörter von den Dokumenten/ Mediadateien durchsuchen und die zutreffenden Ergebnisse zurückliefern. Weiterhin soll der Nutzer die eingegebenen Suchbegriffe mit einigen binären Operatoren verbinden können, um die Ergebnisse dieser Recherche exakt wie möglich zu bestimmen. Das Ergebnis der Suche soll neben den gefunden Dokumenten und Mediadateien, auf welche die Suchergebnisse zutreffen, auch noch eine Liste der verwandten Suchbegriffe aufweisen. Diese Liste ist eine Zusammenstellung der einzelnen Listen, welche die Suchbegriffe der jeweils gefundenen Dokumente/ Mediadateien beinhalten sollen. Dabei sollen die gesuchten Begriffe nicht in der Liste mit angegeben werden und die Größe der Liste soll vom Nutzer manuell

festgelegt werden. Diese Liste kann auch als Navigation zu den verwandten Dokumenten/ Mediadateien verwendet werden. Das folgende Beispiel (Abb. 25) veranschaulicht dieses Prinzip der Recherche. Als Suchbegriff wird hier *PKW* angegeben. Dieser Begriff trifft nur auf die zwei blau markierten Dokumente zu. Die blau markierte Liste mit den Suchbegriffen beinhaltet die Suchbegriffe der beiden Dokumente. Über das gemeinsame Suchbegriff *KFZ* soll eine Navigation zum grün markierten Dokument ermöglicht werden.

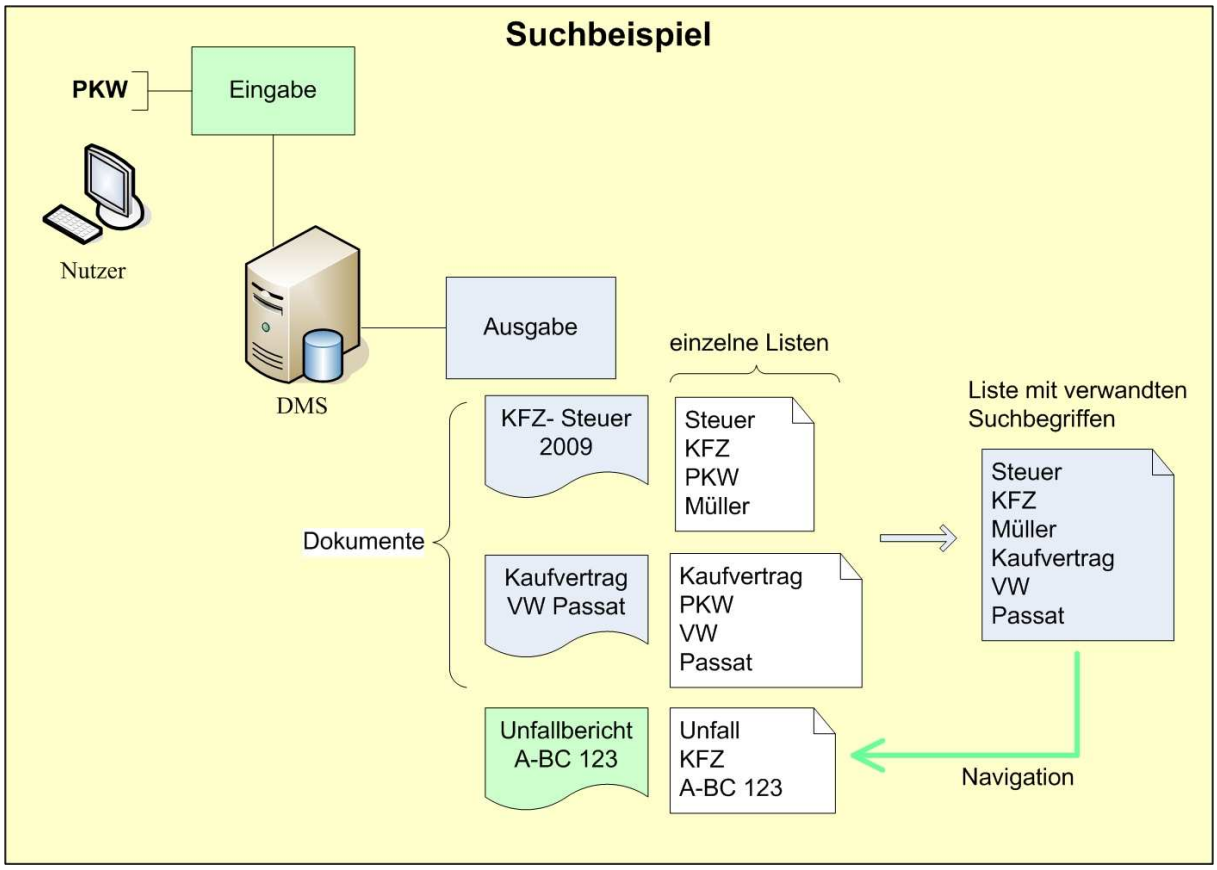

Abb. 25: Beispiel einer Suche

Weiterhin soll der Nutzer eine Aufzählung der von ihm zuletzt verwendeten Dokumente erhalten können. Die Anzahl dieser Dokumente sowie die Anzahl der gelieferten Suchergebnisse legt der Nutzer selbstredend vor der Suche fest. Die Dokumente sollen als zuletzt verwendete Dokumente gekennzeichnet werden, wenn das Dokument vom Nutzer erstellt oder bearbeitet wurde. Zusätzlich sollen die unterschiedlichen Sichtbarkeitszustände bei der Dokumentensuche berücksichtigt werden. Ist die Suche nach den *nicht sichtbaren* Dokumenten erwünscht, so soll der Nutzer durch eine Kennzeichnung nach diesen Dokumenten suchen können. Die Recherche erfolgt unter der Einschränkung, dass der

Nutzer auf diese gesuchten Dokumente die notwendigen Rechte besitzt. Bei der Auslieferung des Dokumentes an den Nutzer sollen auch, falls verfügbar, die damit verbundenen Mediadateien angezeigt werden. Es ist durchaus vorstellbar, dass ein Nutzer die Rechte zur Suche auf das Dokument aber nicht auf die Mediadatei besitzt. So soll in diesem Fall nur das Dokument an den Nutzer ausgeliefert werden. Spiegelbildlich funktioniert die Suche auch für die Mediadateien.

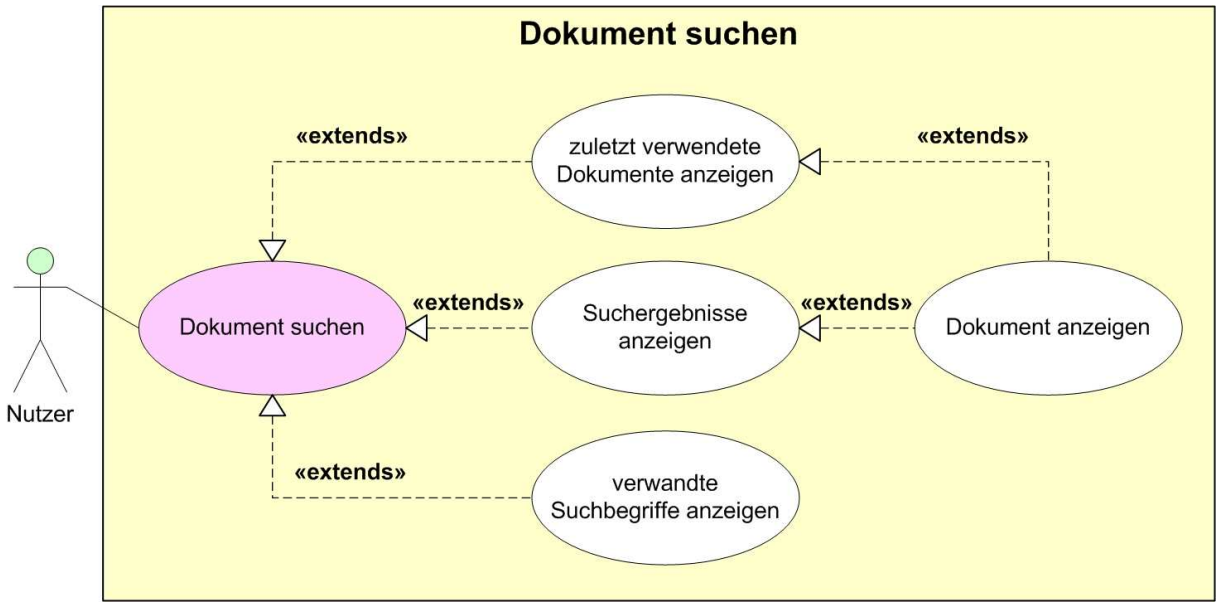

Abb. 26: funktionale Anforderung *Dokument suchen*

3.2.1.6. Dokument bzw. Media bearbeiten

Das System soll das Bearbeiten eines Dokumentes oder einer Mediadatei durch den Nutzer gewährleisten. Damit ist dem Nutzer die Möglichkeit gegeben, den Inhalt eines Dokumentes/ einer Mediadatei mit den zugehörigen Metainformationen zu ändern. Der Nutzer soll alle Eingaben ändern können, welche beim Erstellen angelegt worden sind. Um das Dokument/ Media bearbeiten zu können, muss es zuerst im System unter der Zuhilfenahme der Suche *lokalisiert* werden. Die Änderung des Dokumentes und der Media ist im Prinzip identisch. Die Unterschiede liegen nur in den einzelnen Metainformationen. Die Abb. 27 und Abb. 28 veranschaulichen das Bearbeiten des Dokumentes, gleichermaßen verläuft dieser Prozess bei der Mediabearbeitung.

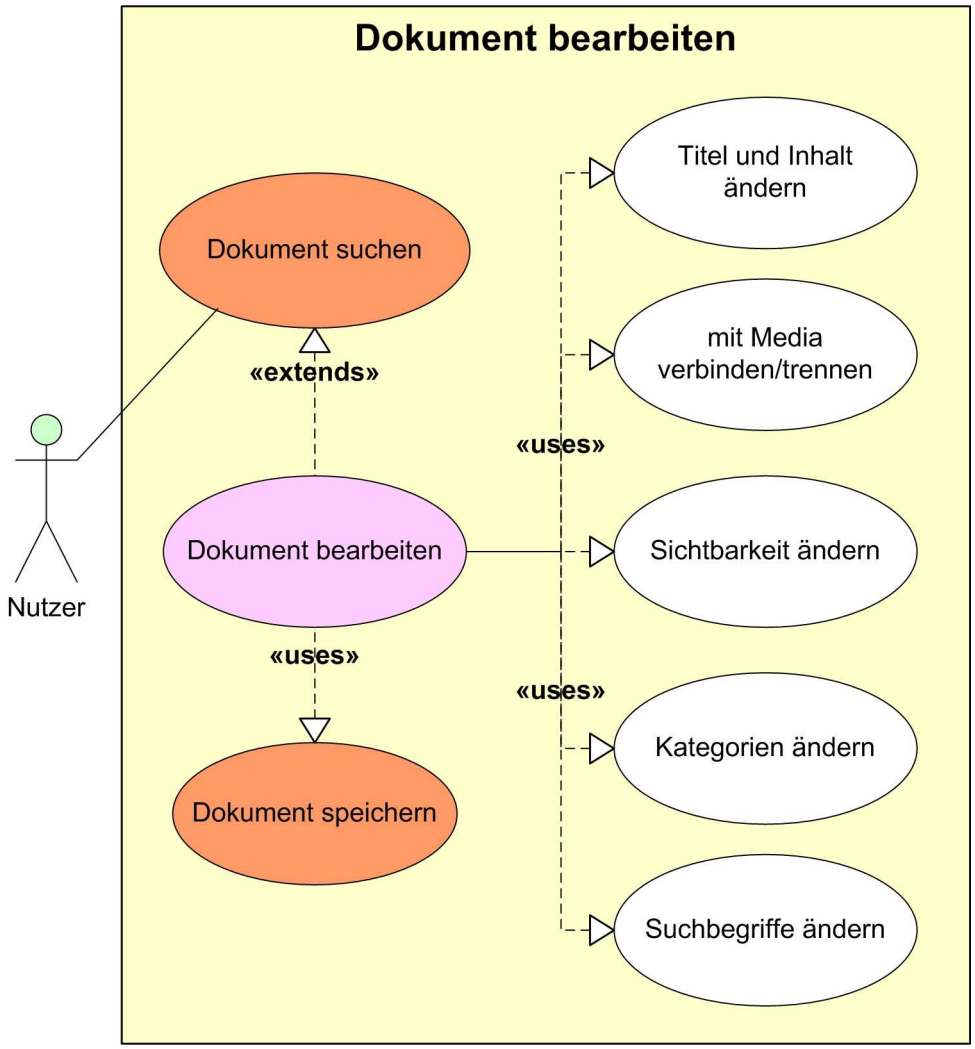

Abb. 27: funktionale Anforderung *Dokument bearbeiten* im Anwendungsfalldiagramm

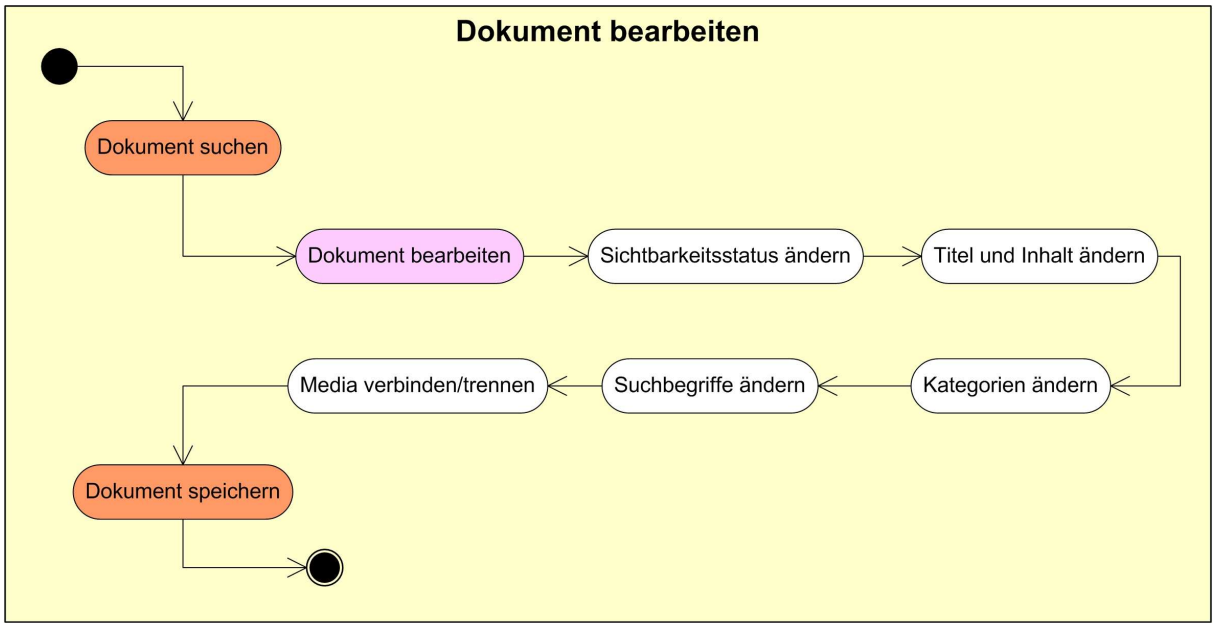

Abb. 28: funktionale Anforderung *Dokument bearbeiten* im Aktivitätsdiagramm

3.2.1.7. Kategorie bearbeiten

Genauso wie ein Dokument soll auch eine Kategorie geändert werden können. Der Nutzer soll den Namen sowie die Oberkategorie bei den vorhandenen Kategorien ändern können. Die Änderung der Suchbegriffe für eine Kategorie verläuft genauso wie bei einem Dokument. Wie überall im System, wird auch an diesem Prozess eine Rechteprüfung stattfinden, um nur den dafür autorisierten Nutzern diese Änderungen zu gestatten.

3.2.1.8. Dokument bzw. Media löschen

Das Vernichten von Dokumenten soll vom DMS unterstützt werden, um dem Nutzer die Möglichkeit zu geben, nicht mehr verwendete bzw. veraltete Dokumente aus dem System herauszunehmen. Für das Entfernen der Dokumente aus dem System ist es wie bei dem Bearbeitungsvorgang notwendig, dass diese zuerst im System ausfindig gemacht werden. Das System soll auch das Löschen von mehreren Dokumenten gleichzeitig anbieten. Bei dem Löschvorgang soll das Dokument unwiderruflich aus dem System mit den bestehenden Zuordnungen zu Kategorien, zur Media, zu dem Nutzer und zu den Suchbegriffen entfernt werden. Wie bereits erwähnt, soll auch der Löschvorgang nur von den dazu berechtigten Nutzern ausgeführt werden. Die aufgezählten Funktionalitäten sollen gleichermaßen auch auf die Mediadatei angewendet werden.

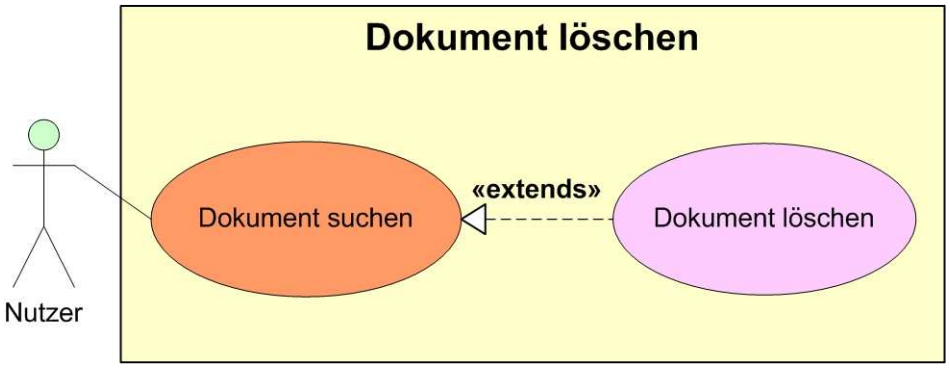

Abb. 29: funktionale Anforderung *Dokument löschen* im Anwendungsfalldiagramm

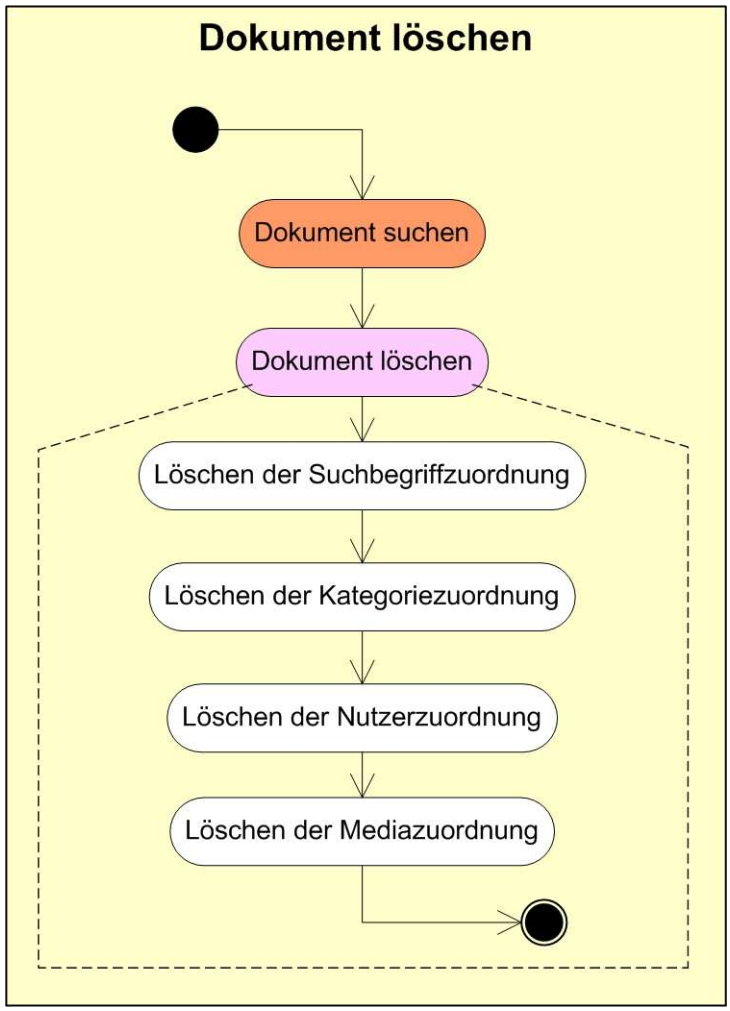

Abb. 30: funktionale Anforderung Dokument löschen im Aktivitätsdiagramm

3.2.1.9. Kategorie löschen

Das Löschen der Kategorie kann dazu verwendet werden, um alle darin befindliche Dokumente sowie Unterkategorien rekursiv aus dem System in einem Schritt zu entfernen. Die Kategorie soll auch bei Nichtverwendung aus dem System gelöscht werden. Über die Suche kann durch den Nutzer die entsprechende Kategorie bestimmt werden.

3.2.2. Administratorbereich

Nachdem die funktionalen Anforderungen aus der Nutzersicht festgelegt wurden, erfolgt die Definition der Anforderungen für die Bereiche Nutzer- und Rechteverwaltung aus der Administratorsicht.

3.2.2.1. Nutzer erstellen

Der Nutzer ist eines der unerlässlichen Elemente des DMS. Ohne die Anmeldung durch einen Nutzer kann das System nicht verwendet werden. Der Administrator soll einen Nutzer mit der Eingabe seines Logins erstellen. Zusätzlich kann der Name, der Vorname sowie die Email eingetragen werden. Jedoch allein der Nutzer hat keine Relevanz im System. Nur bei der Zuweisung des Nutzers in eine Gruppe (oder auch mehrere Gruppen) kann dieser Nutzer das System verwenden, weil die Rechte nicht an die Nutzer, sondern an die Nutzergruppen vergeben werden.

3.2.2.2. Gruppe erstellen

Die Gruppe soll einen Namen sowie eine Beschreibung erhalten. Jede Gruppe soll mit den erforderlichen Rechten den Kategorien des Systems zugewiesen werden. Erst nach dieser Zuweisung sollen die Nutzer dieser Gruppe den Zugriff auf die Dokumente bzw. Mediadateien erhalten.

3.2.2.3. Nutzer bzw. Gruppe bearbeiten

Der Administrator soll bei den erstellten Nutzer und Gruppen ihre Eigenschaften ändern können. Beim Bearbeiten des Nutzers liegt eine der wichtigen Änderungen in der Zuweisung des Nutzers zu den Gruppen. Über diese Änderungen werden die Systemrechte des Nutzers geändert.

3.2.2.4. Rechteverwaltung

Die Rechteverwaltung ist ein zentraler Punkt in diesem DMS. Beim Verwalten der Rechte soll der Administrator die Gruppen zu den Kategorien mit bestimmten Rechten zuweisen können.

3.2.2.5. Nutzer bzw. Gruppe löschen

Wird ein Nutzer nicht mehr gebraucht, so soll dieser aus dem System entfernt werden. Beim Löschen einer Gruppe werden alle zu dieser Gruppe zugewiesene Nutzer nicht aus dem System entfernt. Damit diese Nutzer aber weiter-

hin verwendet werden können, müssen diese in mindestens eine weitere Gruppe eingeteilt werden.

3.2.2.6. Suchbegriffe verwalten

Der Administrator soll die Suchbegriffe für die Dokumente, Media und Kategorien verwalten können. Wird eines dieser Dokumente bzw. Kategorien bearbeitet oder sogar aus dem System entfernt, so kann der Fall eintreten, dass es Suchbegriffe im System existieren, welche nirgendwo eine Verwendung finden. Es ist nicht effizient die Suchbegriffe gleich bei der Nichtverwendung aus dem System zu entfernen. Dies wird dadurch begründet, dass diese eventuell zu einem späteren Zeitpunkt wieder gebraucht werden und nicht ein zweites Mal in das System integriert werden müssen. Aus diesem Grund soll im Rahmen einer Systemwartung die Entfernung von nicht mehr verwendbaren Suchbegriffen erfolgen.

3.3. Nicht-funktionale Anforderungen

Die nicht-funktionalen Anforderungen charakterisieren die Qualität des Systems, also wie gut das System etwas bewerkstelligen kann. Dabei bildet sich der Unterschied zu den funktionalen Anforderungen heraus, welche die funktionelle Sicht beschreiben, also was genau ein System zu leisten hat. Die folgende Auflistung stellt die einzelnen Anforderungen vor [vgl. fr10b]:

- **Leistung und Effizienz**: Das Backend-System muss in der Lage sein, kurze Antwortzeiten zu liefern. Darüber hinaus sollen alle notwendigen Ressourcen zur Verfügung gestellt werden.
- **Korrektheit**: Es ist zu gewährleisten, dass fehlerfreie Ausgabe der Ergebnisse (Dokumente eines DMS) bei der eindeutigen Existierung erfolgt.
- **Integrität**: Die Informationen dürfen bei ihrer Verarbeitung bzw. Übertragung nicht beschädigt werden.
- **Vertraulichkeit**: Der Zugriff auf das System sowie die Teilsysteme ist nur den authentifizierten Benutzern zur Verfügung zu stellen.

- **Flexibilität**: Die Erweiterbarkeit des Systems in Bezug auf die Änderung bzw. Erweiterung der Funktionalitäten soll über das Ändern bzw. Hinzufügen der Dienste ermöglicht werden.
- **Robustheit**: Durch die Fehlbedienung des Systems durch den Nutzer ist ein Mechanismus zu deren Abwehr einzusetzen.

4. Entwurf

Dieses Kapitel stellt den Entwurf des Backend- Systems vor. Der vollständige Systemaufbau wird durch das Datenmodell sowie das Objektmodell näher erläutert. Im weiteren Verlauf wird der allgemeine Aufbau eines Webservices beschrieben und das Gesamtsystem in die einzelne Teilsysteme unterteilt. Ferner erfolgt eine Gegenüberstellung eines Teilsystems zu einem Webservice. Zum Schluss dieses Kapitels werden alle Teilsysteme des kompletten Backend-Systems in einer zusammenfassenden Übersicht vorgestellt.

Der Systementwurf ist nach der Analyse das nächste logische Glied in der Fertigung eines Softwareprojektes. In dieser Phase wird die Systemarchitektur ausgewählt und erarbeitet sowie das entstandene Analysemodell verfeinert. Anschließend erfolgt die Definition der Klassen sowie der Assoziationen untereinander, welche die Grundlage für die Implementierung darstellen. [vgl. lo10] Das Gesamtsystem sollte in Teilsysteme aufgebrochen werden, um die Komplexität des Systems überschaubar zu halten. Bei dieser Zerlegung ist aber zu beachten, dass die entstandenen Teilsysteme untereinander lose gekoppelt sind. Somit lässt sich für jedes Teilsystem eine Schnittstelle erstellen, welche einen bestimmten Aufgabenbereich abdeckt. Im Entwurf wird die Frage nach dem *wie* das System realisiert wird, beantwortet. Aus diesem Grund werden folgende Überlegungen festgehalten. Die Basis für die Architektur liefert die open-EIS Plattform, welche ein SOA- Framework für die Erstellung von Webservice- Anwendungen sowie das O/R- Mapping zur Verfügung bereitstellt. Diese Plattform wurde im Kapitel der Grundlagen vorgestellt. Das Backend-System soll die Anforderungen aus der Analyse übernehmen. Die Funktionalitäten von diesem System werden über die vordefinierten Webservice- Schnittstellen nach außen hin angeboten. Dieses System wird im Grunde eine Sammlung von Webservices aufweisen, welche dem Client zur Verfügung gestellt werden. Durch eine geeignete Kombination von diesen Webservices kann eine fertige Clientanwendung erstellt werden, welche die gewünschten Funktionen in logisch zusammenhängender Abfolge vereinigt.

4.1. Systemaufbau

Das DMS basiert auf einem relationalem Datenmodell, welches später in ein Objektmodell zu überführen ist. Dieser Schritt ist notwendig, weil die folgende Implementierung mit Hilfe einer objekt-orientierten Sprache Java umgesetzt wird. Für die visuelle Darstellung des Datenmodells eignet sich dazu das Entity-Relationship-Diagramm. Dieses Diagramm stellt in einer statische Sicht auf das System die Objekte (entity) sowie deren Beziehungen (relationship) zueinander dar. Im Folgenden werden die dargestellten Objekte aus dem Diagramm (Abb. 31) näher beschrieben:

- **Nutzer** (users): ist ein Objekt, welches für die Autorisierung gegenüber dem System benötigt wird. In der Regel entspricht dieses Objekt einem Systemanwender.
- **Nutzer- Gruppe- Zuordnung** (users2groups): ist ein Verbindungsobjekt, welches einen *Nutzer* einer *Gruppe* zuordnet.
- **Gruppe** (groups): ist eine Menge von *Nutzern*, welche im System als eine Einheit im Sinne des Rechtemanagements behandelt wird.
- **Gruppe- Kategorie- Zuordnung** (groups2cat): ist ein Verbindungsobjekt, welches eine *Gruppe* einer *Kategorie* zuordnet. Dabei erhält jede dieser Zuordnung eine Summe von vordefinierten *Rechten*. Somit lässt sich jede *Kategorie* einzeln jeder *Gruppe* mit bestimmten *Rechten* zuordnen.
- **Recht** (c4u_core_catpermission): ist eine Definition der im System verfügbaren Rechte.
- **Kategorie** (category): ist ein Container. Ein *Dokument*, eine *Mediadatei* oder auch ein *Suchbegriff* können einer oder mehreren *Kategorien* zugeordnet werden. Ebenso wie o.e. können mehrere *Gruppen* derselben *Kategorie* mit unterschiedlichen *Rechten* zugeordnet werden.
- **Kategorie- Dokument- Zuordnung** (cat2doc): ist ein Verbindungsobjekt, welches eine *Kategorie* einem generischen Dokument zuordnet. Das bedeutet, dass dieses Dokument einem *Do-*

kument, einer *Media* oder auch einem *Suchbegriff* entsprechen kann. Die Auflösung findet über das *Template* statt.

- **Template** (dv_template): ist eine Strukturbeschreibung eines Dokumentes in open-EIS.
- **Dokument** (c4u_dms_doc): ist ein zentrales Element des in dieser Arbeit erstellten DMS. Es beinhaltet die Informationen zu einem bestimmten Sachverhalt.
- **Media** (c4u_dms_media): ist wie ein *Dokument* ein zentrales Element des in dieser Arbeit erstellten DMS zur Ablage von unterschiedlichen Mediendateien als zusätzliche Information. Dieses Element kann in Verbindung mit einem oder mehreren *Dokumenten* gestellt werden.
- **Dokument- Media- Zuordnung** (c4u_dms_doc2media): ist ein Verbindungsobjekt, welches ein *Dokument* einer *Media* zuordnet und umgekehrt.
- **Suchbegriff** (c4u_core_keyword): ist ein Schlüsselwort für ein *Dokument*, *Media* oder *Kategorie* im System.
- **Dokument-Suchbegriff-Zuordnung** (c4u_core_document2keyword): ist ein Verbindungsobjekt, welches ein generisches Dokument einem *Suchbegriff* zuordnet. Hier entspricht das generische Dokument einem D*okument*, einer *Media* oder einer *Kategorie*. Die Auflösung findet auch hier über das *Template* statt.
- **Nutzer-Dokument-Zuordnung** (c4u_dms_users2doc): ist ein Verbindungsobjekt, welches einen *Nutzer* einem generischem Dokument zuordnet. An dieser Stelle sind es das *Dokument* oder die *Media*. Die Verwendung dieser Zuordnung findet bei der Bestimmung der zuletzt verwendeten *Dokumente*/*Media* des *Nutzers* statt.

Die Abb. 31 veranschaulicht den o.e Aufbau. Die orange markierten Entitäten werden aus der bestehenden open-EIS Architektur wieder verwendet, die grün markierten Entitäten dagegen werden für dieses DMS neu erstellt.

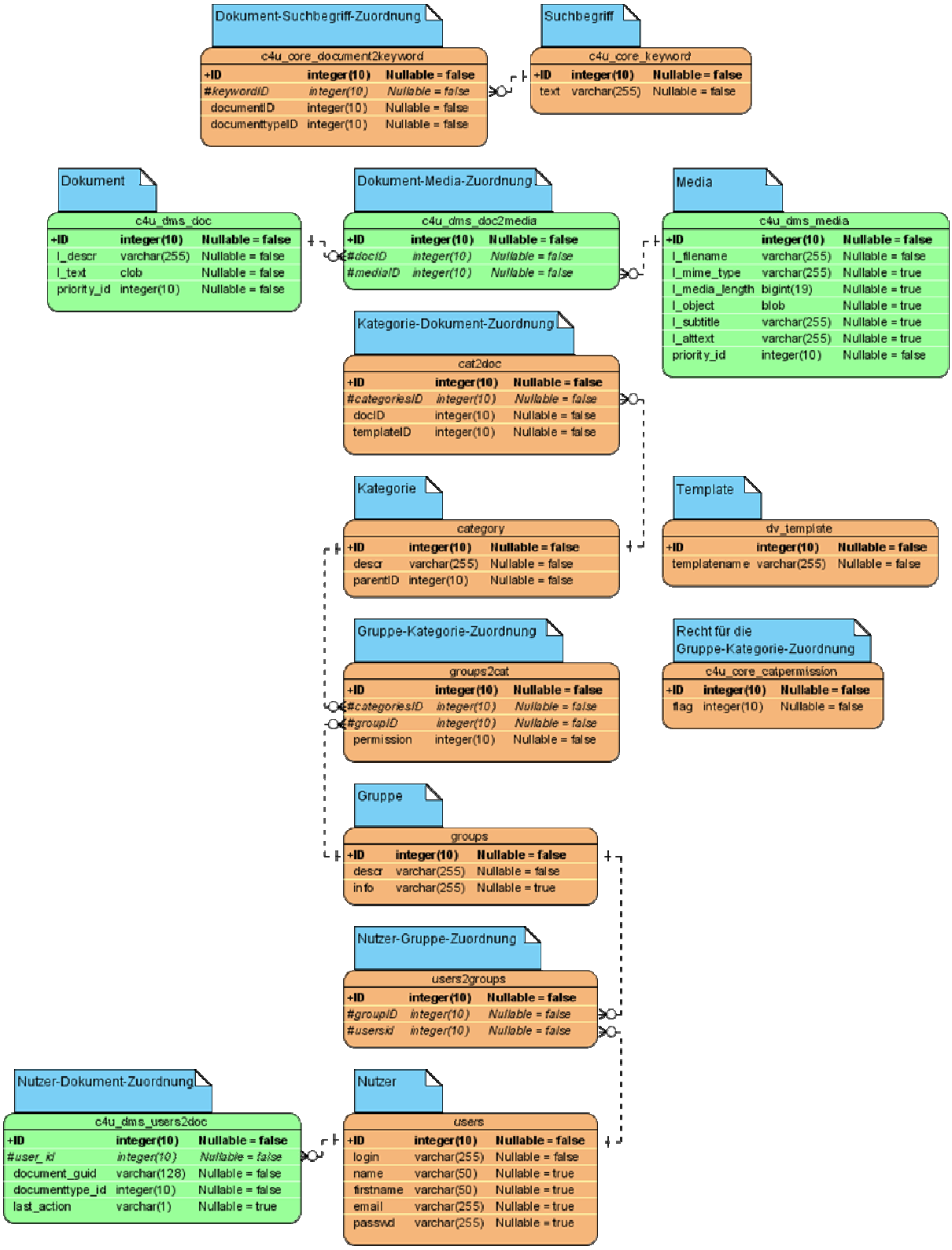

Abb. 31: ER- Diagramm des DMS

Bei der Überführung des relationalen Datenmodells in ein Objektmodell steht meistens eine Klasse für eine Tabelle. Die Tabellen, welche eine n-m- Beziehung auflösen, werden im Allgemeinen nicht als Klassen angegeben, weil sie in Wirklichkeit keine neuen Objekte abbilden. Dennoch werden hier diese Tabel-

len als Klassen verwendet, weil sie für das O/R- Mapping gebraucht werden. Auf Basis des oben vorgestellten Datenmodells wird ein Objektmodell erzeugt, welches im folgenden Diagramm dargestellt wird. An diesem Punkt ist noch zu beachten, dass nur für die neu erstellten Entitäten diese Überführung realisiert wird, weil für die bestehenden Entitäten entsprechende Klassen bereits vorhanden und einsatzbereit sind.

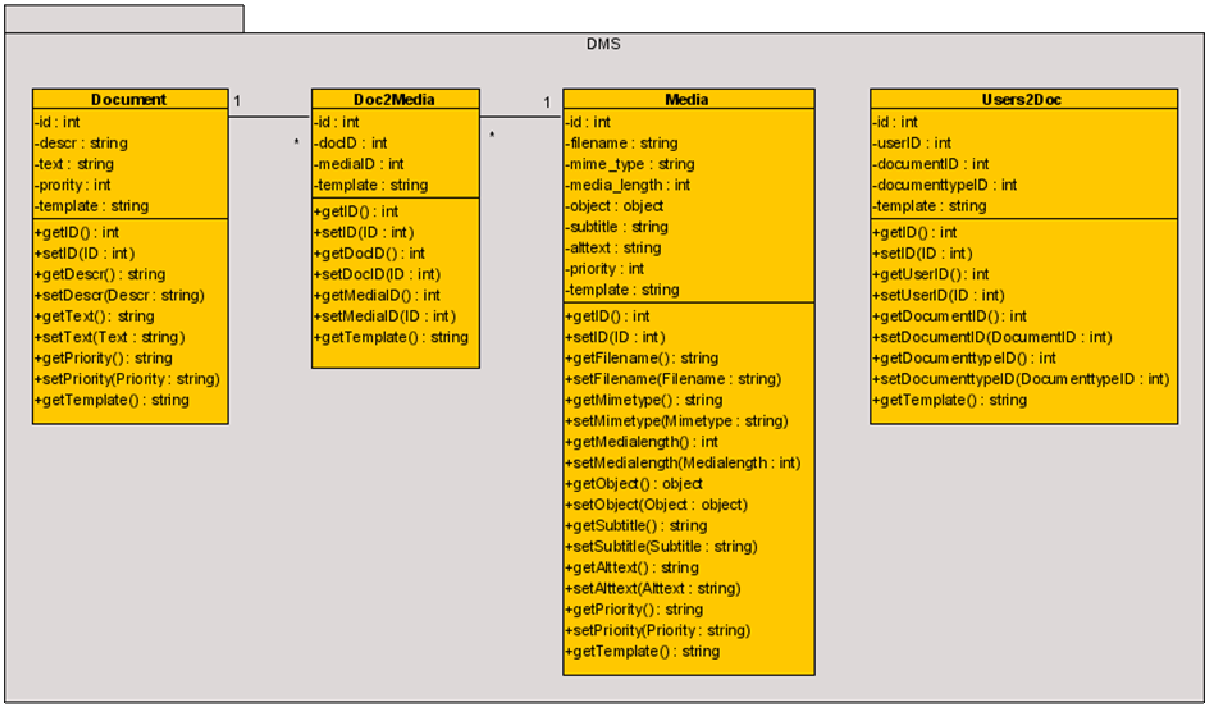

Abb. 32: Objektmodell im Klassendiagramm

4.2. Aufbau eines Webservices

Durch die Verwendung des Hermes- Frameworks unterliegt der Aufbau eines Webservices bestimmten Konventionen. So müssen alle Services in einem Package mit dem Namen *service* im entsprechenden Modul untergebracht werden. Jedes Webservice soll eine bestimmte Funktionalität abbilden und wird in einem eigenen Package verwaltet. Dieses Package enthält in der Regel lediglich zwei Unterverzeichnisse: *client* und *test*. Das Package *client* enthält eine Schnittstelle, welche die gesamten Funktionalitäten des Services definiert und eine Ausnahme- Klasse für die Verarbeitung der auftretenden Fehler in diesem Service. Des Weiteren können die s.g. Datenobjekte (DO) in diesem Package enthalten werden, welche eine konkrete Entität aus der realen Welt für den Client darstellen. Im Package *test* werden die Testklassen abgelegt, um alle

Funktionalitäten des Services auf ihre Richtigkeit hin zu überprüfen. Im Wurzel-Package des jeweiligen Services befindet sich eine Klasse für die Implementierung der Schnittstelle. Im Regelfall beinhaltet diese Klasse nur die Implementierungsmethoden dieser Schnittstelle. [vgl. Ce07a] S.5 Falls weitere Hilfsmethoden gebraucht werden, so können diese in einer Hilfsklasse abgelegt werden. Üblicherweise sind alle Methoden der Hilfsklasse statisch.

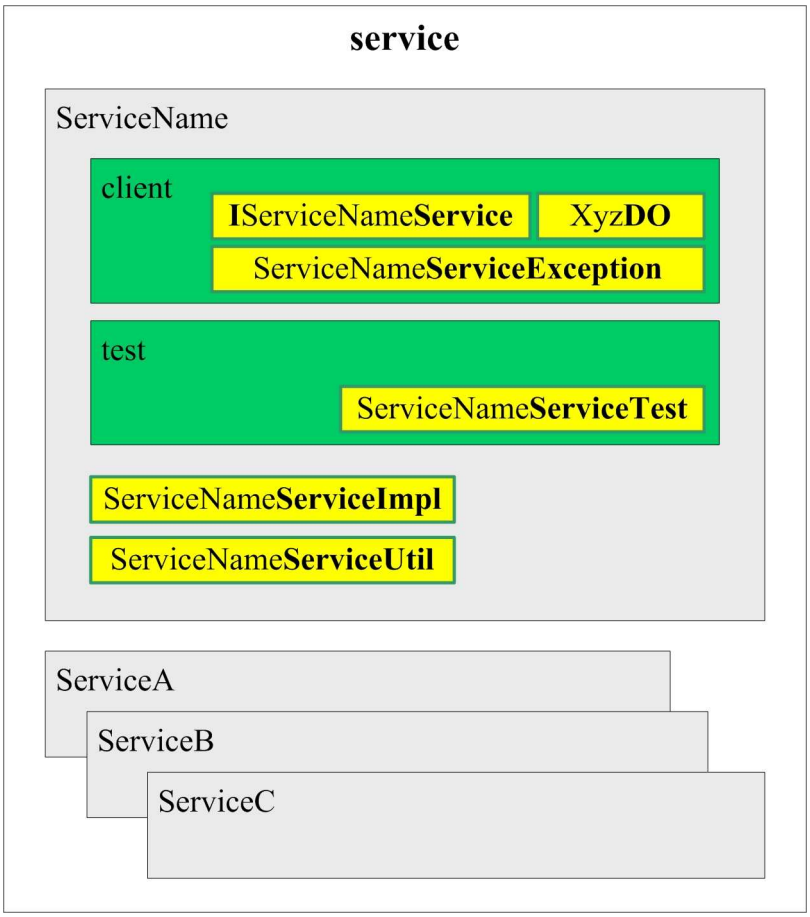

Abb. 33: Aufbau des Webservices

4.3. Teilsysteme

Nachdem das gesamte System entworfen ist und der Aufbau eines Webservices ermittelt wurde, kann das System in die Teilsysteme zerlegt werden. Folgende Teilsysteme lassen sich herausformen: *User*, *Category*, *Document* und *Keyword*. Jedes dieser Teilsysteme entspricht zum großen Teil einem Webservice und stellt seinerseits die erforderlichen Funktionalitäten bereit. Die Abb. 34 zeigt die Übersicht auf das gesamte Backend- System unterteilt in die

einzelnen Teilsysteme. Jedes Teilsystem wird bis auf das blau markierte Verzeichnis in einen für sich abgeschlossenen Webservice überführt.

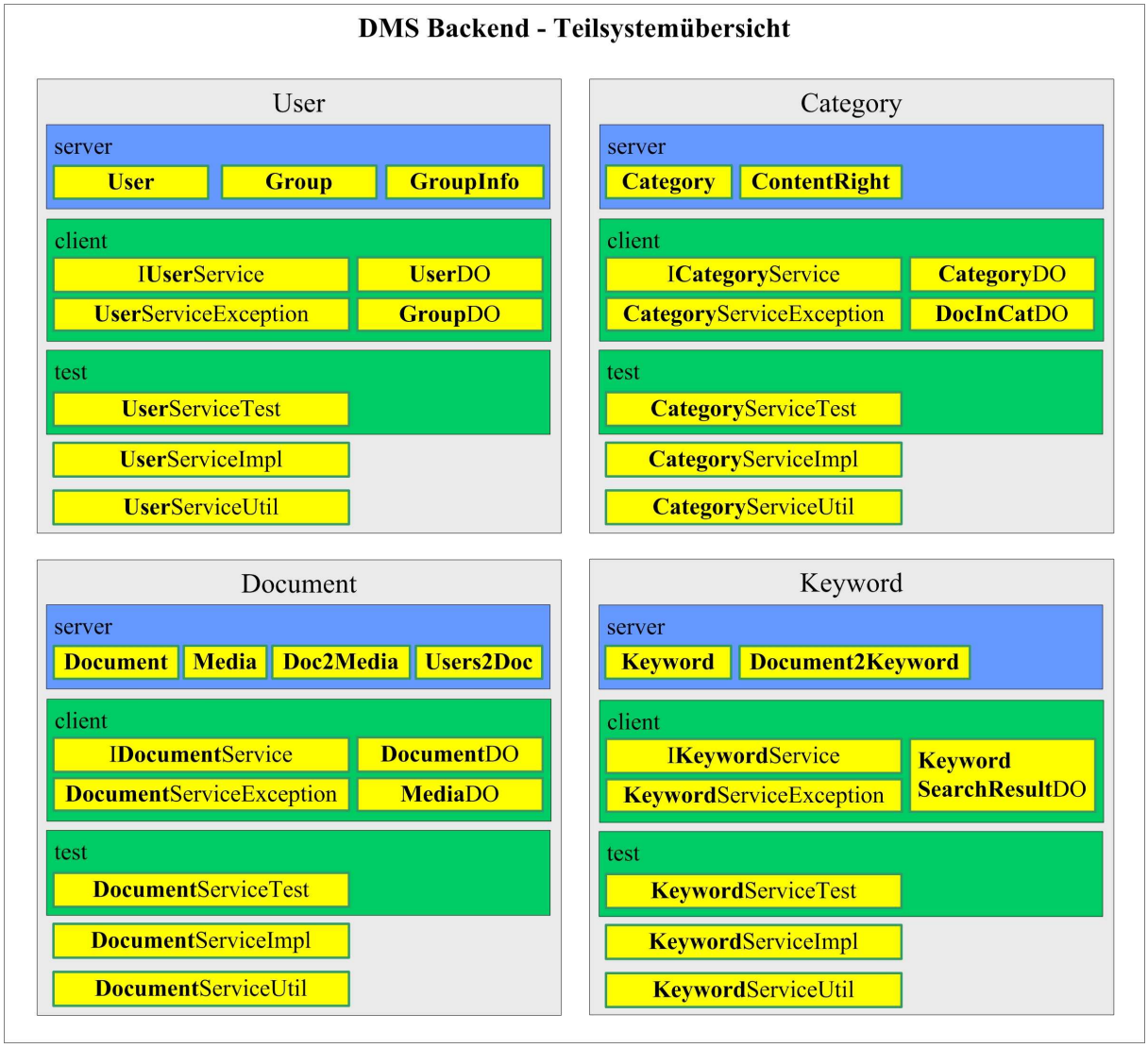

Abb. 34: Teilsystemübersicht

4.3.1. Teilsystem – User

In diesem Bereich werden die Grundfunktionalitäten für die Manipulation von Nutzern sowie ihren Gruppen zur Verfügung gestellt. Da die serverseitigen Klassen für die Nutzer (*User*), Gruppen (*Group*) und Zuweisung der Nutzer zu den Gruppen (*GroupInfo*) wieder verwendet werden, müssen nur noch die fehlenden Data Objects (*UserDO*, *GroupDO*) für den Client angelegt werden. Die Schnittstelle (*IUserService*) beschreibt die o.e. notwendigen Funktionen und die *UserServiceImpl* – Klasse implementiert die Logik der einzelnen Funktionen dieser Schnittstelle. Die Klasse *UserServiceException* verwaltet die auftreten-

den Fehler während der Abarbeitung des jeweiligen Funktionsaufrufes. Durch den Datentransport vom Server zum Client müssen die serverseitigen Klassen in die clientseitigen DO's konvertiert werden. Die umgekehrte Konvertierung ist nicht zwingend notwendig, da die DO's serverseitig sofort ausgewertet werden können. Diese Konvertierung kann durch die *UserServiceUtil* – Klasse vorgenommen werden.

4.3.2. Teilsystem – Category

Ähnlich dem *Teilsystem – User* wird dieses Teilsystem mit den Grundfunktionalitäten für die Kategorie aufgebaut. Auch hier müssen die fehlenden Data-Objects (*CategoryDO* und *DocInCatDO*) erstellt werden. Die serverseitigen Klassen für die Kategorie (Category) und die Zuweisung der Gruppen zu Kategorien (ContentRight) sind bereits vorhanden. Einige wichtige Funktionalitäten sind das Setzen der Rechte für die Gruppe auf eine bestimmte Kategorie sowie das Holen des Kategoriebaumes samt aller darin befindlichen Dokumenten. Diese sowie die anderen Funktionalitäten werden durch die Schnittstelle (*IcategoryService*) beschrieben und mit der *CategoryServiceImpl* – Klasse implementiert.

4.3.3. Teilsystem – Document

Dieser Bereich stellt die Grundfunktionalitäten für die Kernelemente des DMS (Dokumente und Media) zur Verfügung. Des Weiteren werden die Funktonalitäten für das Hinzufügen und Entfernen der Verbindungen zwischen diesen Elementen erstellt. Für die computergestützte Indexierung der Dokumente sowie für die zuletzt verwendeten Dokumente wird jeweils eine zusätzliche Funktion erarbeitet. Alle diese Funktionen werden durch die *IdocumentService* – Schnittstelle beschrieben und mit der *DocumentServiceImpl* – Klasse implementiert. Da dieser Bereich vollständig neu entworfen wird, werden für den Client die Datenobjekte (*DocumentDO*, *MediaDO*) und für den Server die Klassen *Document*, *Media*, *Doc2Media* und *Users2Doc* erstellt. Für die Konvertierung der Klassen sorgt die *DocumentServiceUtil* – Klasse.

4.3.4. Teilsystem – Keyword

Der Bereich *Keyword* existiert bereits in der open-EIS Plattform, sodass es nicht notwendig ist, diesen wiederholt zu entwerfen. Dieser Webservice wird komplett übernommen und es werden lediglich Anpassungen von einigen Funktionalitäten bei der Implementierung vorgenommen. Die Schnittstelle *Ikeyword-Service* beschreibt außer den Grundfunktionen (Hinzufügen und Entfernen) auch noch eine wichtige Suchfunktion für die Dokumente, Media und Kategorien. Für die Administration wird eine weitere Funktionalität zum Löschen von nicht mehr benötigten Suchbegriffen erstellt.

5. Implementierung

Die gesamte Implementierung des Backend- Systems wird in diesem Kapitel beschrieben. Die Basis dafür bildet der im letzten Kapitel vorgelegte Entwurf. Zunächst werden einige Vorbereitungen beschrieben, welche wegen der Umsetzung des Systems auf der open-EIS Plattform für die konkrete Implementierung notwendig sind. Im nächsten Schritt wird die Überführung des Datenmodells in das Objektmodell dargestellt. Anschließend werden die vier erarbeitenden Teilsysteme aus dem Entwurf in ihrer Realisierung einzeln erläutert.

Alle in der Anforderungsanalyse ermittelten Funktionalitäten des gesamten Systems wurden im Entwurf berücksichtigt und werden in der Implementierung umgesetzt. Da die Implementierung auf der open-EIS Plattform stattfindet, sind vorerst einige Vorbereitungen notwendig. Diese Plattform ist modular aufgebaut und besteht im Moment aus drei Hauptmodulen (Core, Classic und Media), welche für jedes neue Projekt notwendig sind. Bei der Umsetzung des Backend-Systems handelt es sich nicht um eine Erweiterung von Hauptmodulen und aus diesem Grund wird ein weiteres Modul (DMS) benötigt. Alle Module von open-EIS werden im CVS verwaltet. Um eine lauffähige Entwicklungsumgebung mit open-EIS zu bekommen, müssen die erforderlichen Module ausgecheckt und installiert werden. Bei diesem Prozess wird gleichzeitig die PostgreSQL- Datenbank angelegt. Aus dem Ergebnis dieser Installation kann ein Projekt im Eclipse erstellt werden.

5.1. Datenmodell und Objektmodell

Bevor die ersten Klassen mit ihrer Logik geschrieben werden können, müssen zuerst die benötigten Tabellen (Templates) aus dem Entwurf erzeugt und angelegt werden. Die Erzeugung erfolgt mittels einer Umgebung für die visuelle Modellierung und das Anlegen durch den Import dieser Modellierung im XMI- Format in das Eclipse- Projekt. Aus diesem Modell werden sowohl die Datenbank erweitert als auch die im Entwurf beschriebenen Klassen für den Server und die DO- Klassen für den Client automatisch über den MDA- Prozess generiert. Diese serverseitigen Klassen werden im weiteren Verlauf auch als Inhaltsklassen (content classes) bezeichnet, weil die Objekte dieser Klassen die

Inhaltsträger während des O/R- Mappings sind. Alle erzeugten Klassen müssen im jeweiligen Modul registriert werden, um sie später in der Implementierung verwenden zu können.

5.2. Webservices

Alle erstellten Webservices müssen in ihren Schnittstellen einen eindeutigen Identifikationsnamen haben. Über diesen Namen kann der jeweilige Service angesprochen und auch registriert werden. Jede Webservice-Methode muss nach den Konventionen von open-EIS als ersten Parameter eine s.g. *Context ID* besitzen und erst danach folgen die Nutzdaten. [vgl. Ce07a] S. 18 Diese *Context ID* ist der Identifikator für das Context- Objekt, welches die Authentifizierungsinformationen enthält. Nach dem Login des Clients in das System wird durch den Authentifizierungsservice eine eindeutige *Context ID* erzeugt. Sie kann mit einer Menge von Berechtigungsnachweisen des Clients assoziiert werden, welche für das Rechtemanagement erforderlich ist. [vgl. Ce07a] S. 16-17

Wenn eine Webservice-Methode eine Reihe an Änderungen in der Datenbank verursacht, so können diese Änderungen durch eine Transaktion abgesichert werden. Sie gewährleistet, dass alle Änderungen in einem unteilbaren Vorgang abgearbeitet werden.

5.2.1. Teilsystem – User

Die notwendigen Funktionen für die *Nutzer* und *Gruppen* werden durch die Methoden der Schnittstelle (*IUserService*, s. dazu Kapitel III, Anlage 1) beschrieben:

- *createUserDO*
- *getUserDO*
- *updateUserDO*
- *removeUserDO*
- *createGroupDO*
- *getGroupDO*
- *updateGroupDO*
- *removeGroupDO*

Bei der Implementierung der Methoden *createUserDO* und *createGroupDO* wird über die *Context ID* auf das *INSERT*- Recht des erforderlichen Elementes überprüft. Beim Besitz dieses Rechtes wird das übergebene DO ausgewer-

tet, daraus das entsprechende Objekt der Inhaltsklasse (*User*/*Group*) erzeugt und in der Datenbank gespeichert. Als Ergebnis dieser Methode wird dem Client das konvertierte DO zurückgegeben. Ähnlich verläuft dieser Prozess bei den Methoden *updateUserDO* und *updateGroupDO*, wobei hier auf das *WRITE*- Recht geprüft werden muss. Beim Holen des Elementes mit den Methoden *getUserDO* und *getGroupDO* findet eine Überprüfung auf das *READ*- Recht und beim Löschen mit den Methoden *removeUserDO* und *removeGroupDO* auf das *DELETE*- Recht statt. Im Erfolgsfall wird das DO an den Client zurückgeliefert.

Die Hilfsklasse (*UserServiceUtil*) implementiert drei statische Methoden. Eine Methode konvertiert das Objekt der Inhaltsklasse in das entsprechende DO. Zwei weitere Methoden werden für die Auswertung des DO beim Erstellen und Bearbeiten benötigt. Als Rückgabe wird das Inhaltsklassenobjekt geliefert, welches anschließend in der Datenbank abgelegt wird.

5.2.2. Teilsystem – Category

Für die Manipulation der Kategorien besitzt die Schnittstelle (*ICategoryService*, s. dazu Kapitel III, Anlage 2) die Grundfunktionalitäten wie bei der *IUserService*- Schnittstelle, sowie drei andere Methoden.

- *createCategoryDO*
- *getCategoryDO*
- *updateCategoryDO*
- *removeCategoryDO*
- *setPermission*
- *getCategoryChildren*
- *getAllDocumentsFromCategory*

Bei der Erstellung einer Kategorie wird über die *Context ID* das *INSERT*- Recht ihrer Oberkategorie überprüft. Ist dieses Recht vorhanden, so erfolgt der Erstellungsprozess. Das *CategoryDO* wird ausgewertet, in das Inhaltsklassenobjekt überführt und in die Datenbank geschrieben. Danach werden der neuen Kategorie die übergebenen Suchbegriffe zugewiesen. Dieser Vorgang findet über den lokalen Aufruf des *Keyword*- Services statt. Anschließend werden alle Rechte der Oberkategorie der neu erstellten Kategorie übertragen. Als Rückgabe wird dem Client die konvertierte *CategoryDO* geliefert. Die Bearbeitung der Kategorie ist im Grunde analog der Erstellung. Die Unterschiede liegen in der Prüfung des *WRITE*- Rechtes für die zu bearbeitende Kategorie. Weiterhin werden alle vorhanden Suchbegriffe für diese Kategorie mit den neuen Suchbegrif-

fen ausgetauscht. Für die Auslieferung der Kategorie an den Client wird die Methode *getCategoryDO* verwendet. Diese prüft das Vorhandensein des *READ*-Rechtes und liefert die konvertierte *CategoryDO* zurück. Beim Entfernen einer Kategorie werden alle Unterkategorien sowie alle ihr zugewiesene Elemente aus dem System entfernt. Für diese Aktion wird das *DELETE*- Recht benötigt. Die Methode *setPermission* weist die übergebenen Rechte einer Gruppe an eine Kategorie zu. Für den Aufbau des Kategoriebaumes auf der Clientseite wird die Methode *getCategoryChildren* verwendet, um dem Nutzer das Einsortieren der Dokumente bzw. Media zu ermöglichen. Diese Methode liefert alle Unterkategorien einer bestimmten Kategorie mit dem *READ*- Recht. Zusätzlich werden alle geholten Unterkategorien mit einer Markierung versehen, welche die Auskunft über das *WRITE*- bzw. *INSERT*- Recht liefert. Somit kann auf der Clientseite für jeden Nutzer ein für ihn sichtbarer Kategoriebaum aufgebaut werden, welcher beim Erstellen oder Bearbeiten von Dokumenten verwendet werden kann. Mit der Methode *getAllDocumentsFromCategory* kann das *DocInCatDO* geholt werden, welches die Informationen über alle *DocumentDO* und *MediaDO* einer bestimmten Kategorie enthält. Folglich können auf der Clientseite nicht nur der Kategoriebaum, sondern auch alle darin befindlichen Dokumenten konstruiert werden. Dies erleichtert das Einordnen der Dokumente bei deren Erstellung.

Die Hilfsklasse (*CategoryServiceUtil*) ist im logischen Aufbau und Funktionsprinzip der *UserServiceUtil*- Klasse identisch, wobei die Auswertung und Konvertierung hier für eine Kategorie durchgeführt wird.

5.2.3. Teilsystem – Document

Die Implementierung dieses Teilsystems umfasst alle Funktionalitäten, welche für die Dokumente und Media erforderlich sind. Da die Implementierung einiger Funktionen von Dokumenten und Media sich deutlich unterscheiden, werden diese beiden Elemente einzeln vorgestellt. Allerdings besitzt dieses Teilsystem eine gemeinsame Schnittstelle (*IDocumentService*, s. Kapitel III, Anlage 3) für die Dokumente und Mediadateien.

5.2.3.1. Document

Für die Manipulationen an den Dokumenten stehen folgende Methoden zur Verfügung:

- *createDocumentDO*
- *getDocumentDO*
- *updateDocumentDO*
- *removeDocumentDO*
- *addDoc2MediaLinks*
- *getAssignedMedias*
- *removeDoc2MediaLinks*
- *getSuitableKeywordsForDocument*

Für die Erstellung eines Dokumentes wird zunächst auf das Vorhandensein des *INSERT*- Rechts geprüft, danach das übergebene *DocumentDO* ausgewertet und als entsprechendes Objekt der *Document*- Klasse in die Datenbank geschrieben. Anschließend wird das neu erstellte Dokument indexiert. Bei der automatischen Indexierung werden der Autor sowie alle Wörter aus der Beschreibung des Dokumentes hinzugefügt, welche eine Länge zwischen 4 und 18 Zeichen aufweisen. Dieses Vorgehen wurde gewählt, um einerseits die Artikel und andererseits die zusammengesetzten Wörter auszuschließen, da diese Begriffe bei der Suche mit hoher Wahrscheinlichkeit nicht verwendet werden. Weiterhin werden alle Suchbegriffe diesem Dokument zugewiesen, welche vom Client übergeben worden sind. Dabei hat der Client die Möglichkeit über die Methode *getSuitableKeywordsForDocument* die passenden Suchbegriffe aus dem Inhalt des Dokumentes ausgeben zu lassen. Als passende Suchbegriffe werden alle Substantive ohne Sonderzeichen mit einer Wortlänge von 4 – 18 Zeichen charakterisiert. Dieser Vorgang wird auch als computergestützte Indexierung bezeichnet. Nach der Indexierung wird das gespeicherte Dokumentobjekt in das *DocumentDO* konvertiert. Danach wird dieses Dokument über die Zuweisung zum Nutzer in die Menge der zuletzt verwendeten Dokumente hinzugefügt. Als letzter Schritt dieser Aktion wird das konvertierte DO an den Client zurückgegeben. Die Dokumentenbearbeitung über die Methode *updateDocumentDO* besitzt einen ähnlichen Verlauf wie seine Erstellung. Die Unterschiede liegen in der Prüfung des *WRITE*- Rechtes sowie der vollständigen Ersetzung der vorhandenen Suchbegriffe des Dokumentes mit den gerade übergebenen Suchbegriffen. Um einen Dokument aus dem System zu holen, wird vom Client das *READ*- Recht benötigt. Bei der Übergabe der Dokumenten- ID liefert die Methode *getDocumentDO* im Erfolgsfall das konvertierte *DocumentDO* zurück. Sonst

erhält der Client ein leeres *DocumentDO* zurück. Das Entfernen der Dokumente aus dem System führt die Methode *removeDocumentDO* aus. Bei diesem Prozess wird vor dem Löschen das *DocumentDO* konvertiert, danach das Dokument mit allen seinen Referenzen aus dem System entfernt und das gelöschte *DocumentDO* an den Client zurückgegeben.

Die Funktionalitäten für die Verbindungen zwischen dem Dokument und den Mediadateien werden durch drei Methoden realisiert. Die Methode *addDoc2MediaLinks* fügt eine neue Verbindung zwischen dem Dokument und einer Menge von Mediadateien hinzu. Dazu muss der Client das *WRITE*- Recht sowohl auf das Dokument als auch auf alle Mediadateien besitzen, welche in dieser Verbindung verwendet werden. Für die Auslieferung aller verbundenen Mediadateien zu einem Dokument wird die Methode *getAssignedMedias* eingesetzt. Diese Methode nimmt als Parameter die Dokumenten- ID entgegen, holt alle mit diesem Dokument verbundenen Mediadateien und liefert diese als Menge von *MediaDO* an den Client zurück. Die Methode *removeDoc2MediaLinks* entfernt die vorhandenen Verbindungen zwischen einem Dokument und einer Menge von Mediadateien. Um diese Aktion erfolgreich ausführen zu können, muss der Client das *DELETE*- Recht auf das Dokument und auf alle damit verbundenen Mediadateien besitzen.

5.2.3.2. Media

Die Aufzählung unten definiert alle Funktionalitäten, welche bei der Manipulation einer Media dem Nutzer zur Verfügung gestellt werden:

- *createMediaDO*
- *getMediaDO*
- *updateMediaDO*
- *removeMediaDO*
- *addMedia2DocLinks*
- *getAssignedDocuments*
- *removeMedia2DocLinks*
- *prepareUploadMedia*
- *uploadMedia*

Eine Mediadatei wird mit der Methode *createMediaDO* erstellt, falls der Client das *INSERT*- Recht besitzt. Wie immer wird das übergebene *MediaDO* ausgewertet und in die Datenbank geschrieben. Ebenso wie bei einem Dokument findet auch bei einer Media die Indexierung statt. Weil der Inhalt der Media nicht in der Textform vorliegt, kann hier keine computergestützte Indexierung er-

folgen. Bei der automatischen Indexierung werden der Autor, der Dateiname und alle Wörter aus dem alternativen Text sowie dem Untertitel mit einer Wortlänge von 4 – 18 Zeichen der Media zugewiesen. Die manuelle Indexierung erfolgt genauso wie bei dem Dokument. Das Mediaobjekt besitzt anders als bei einem Dokumentenobjekt eine Eigenschaft (*object*), welche nicht serialisierbar ist. Hier handelt es sich um die eigentliche Mediadatei, welche den Informationsstand beinhaltet. Dieses Problem der Media- Persistenz wurde wie folgt realisiert. Bevor der Client das *MediaDO* an den Server zum Speichern sendet, erhält er über die Methode *prepareUploadMedia* eine s.g. *Upload ID*. Danach startet er eine HTTP- Verbindung zum Server unter Angabe dieser ID und schreibt in den Ausgabestrom die zu übergebene Mediadatei. Anschließend wird über die Methode *uploadMedia* die hochgeladene Mediadatei als *File* auf dem Server gehalten. Erst dann ruft der Client die Methode *createMediaDO* auf, welche beim Erstellen der Media auf das serverseitig gehaltene Media- *File* zurückgreifen kann. Als letzter Schritt im Erstellungsprozess wird diese Media über die Zuweisung zum Nutzer in die Menge der zuletzt verwendeten Mediadateien hinzugefügt. Über die Methode *updateMediaDO* kann eine Media geändert werden. Dieser Vorgang verläuft mit einer Ausnahme wie bei der Bearbeitung des Dokumentes. Wenn keine HTTP- Verbindung zum Server aufgebaut und somit keine neue Datei gesendet wurde, findet keine Änderung an der nicht serialisierbaren Eigenschaft der Media statt. Andernfalls wird die neue Datei mit der vorhandenen ersetzt. Das Ausliefern der Media wird über die Methode *getMediaDO* realisiert. Die Übertragung dieser nicht serialisierbaren Eigenschaft vom Server zum Client (Download) wurde auf eine andere Weise implementiert, da hier die Funktionalität für das Herunterladen mit Hilfe des Ausgabestromes im open-EIS nicht vorhanden ist. An dieser Stelle wurde die *SerializeableFile*-Klasse des open-EIS Frameworks eingesetzt, welche aus einem *File*- Objekt ein serialisierbares Objekt erzeugt. Auf dem Client findet dann der Prozess der Deserialisierung statt. Diese Lösung könnte auch für die Übertragung der Media vom Client zum Server (Upload) verwendet werden, jedoch ist diese Übertragung weniger effizient. Das Löschen wird über die Methode *removeMediaDO* realisiert. Der Aufbau und der Verlauf dieser Methoden sind analog denen von Dokument.

Für die Verbindungen von Media zu Dokumenten werden ebenfalls drei Methoden verwendet. Sie liefern in ihrer Funktionalität das Spiegelbild der Methoden für die Dokumentenverbindungen.

5.2.4. Teilsystem – Keyword

Die Funktionalitäten für das Teilsystem *Keyword* sind bereits vorhanden und werden ebenfalls durch die Schnittstelle (*IKeywordService*, s. dazu Kapitel III, Anlage 4) beschrieben. Da die Methode *search* nicht den gestellten Anforderungen entsprach, wurde sie komplett überarbeitet. Zusätzlich wurde dieser Webservice um zwei weitere Methoden erweitert. Zu einem ist es die Methode *removeUnusedKeywords*, welche das Löschen der unbenutzten Suchbegriffe ermöglicht. Zu anderem liefert die Methode *getLastModifiedDocuments* die zuletzt verwendeten Dokumente oder Mediadateien. Der Client kann sowohl den Dokumenttyp als auch die Art der letzten Änderung und die Anzahl der Ergebnisse bestimmen.

Die Methode *addKeywords* fügt alle neuen übergebenen Suchbegriffe dem DO hinzu, wenn der Client das *WRITE*- Recht auf dieses Element besitzt. Bei diesem DO kann es sich um ein Dokument, eine Media oder auch eine Kategorie handeln. Über die Methode *getAllKeywords* lassen sich alle Suchbegriffe eines bestimmten Elementes ausliefern. Dabei findet die Prüfung des *READ*-Rechtes auf dieses Element statt. Das Löschen der Suchbegriffe eines Elementes wird mit der Methode *removeKeywords* realisiert. Hier braucht der Client ebenso wie beim Hinzufügen das *WRITE*- Recht auf das Element. Die Methode *getKeywordsByPrefix* liefert alle Suchbegriffe, welche mit einem bestimmten Präfix beginnen. Diese Funktionalität kann verwendet werden, um sich die Übersicht bestimmter Suchbegriffe zu verschaffen.

Die Methode *search* spielt eine Schlüsselfigur und besitzt eine Reihe an Parametern, welche der Client bei der Suche nach den Dokumenten übergeben kann. Mit diesen Parametern steuert er das Ergebnis dieser Suche. Diese Aufzählung stellt die möglichen Parametern vor:

- **documentTypIds**: eine Menge von Dokumententypen, welche mit in die Suche eingeschlossen werden sollen. Die möglichen Typen sind das Dokument, die Media und die Kategorie.

- **visibleStatus**: steuert die Suchergebnisse über den Sichtbarkeitsstatus der Elemente. Der Wert dieses Parameters kann *true* oder *false* sein.
- **ANDList**: eine Menge von Suchbegriffen. Diese müssen bei der Suche nach Dokumenten alle vorhanden sein.
- **ORList**: eine Menge von Suchbegriffen. Mindestens eines der Suchbegriffe muss bei der Suche vorhanden sein.
- **XORList**: eine Menge von Suchbegriffen. Genau eines der Suchbegriffe muss bei der Suche vorhanden sein.
- **NOTList**: eine Menge von Suchbegriffen. Keines der Suchbegriffe darf bei der Suche vorhanden sein.

Diese Methode liefert als Ergebnis dieser Suche die Elemente, welche allen übergebenen Parametern entsprechen und zusätzlich auf das *READ*- Recht überprüft worden sind.

6. Test

Dieses Kapitel beschreibt die Testphase, welche nach der Implementierung des Systems stattfindet. Diese Phase ist das Entscheidungskriterium, welches die Qualität des Systems beschreibt. Im Verlauf dieses Kapitels wird das eingesetzte Test- Framework vorgestellt sowie der allgemeine Aufbau einer Testklasse charakterisiert.

Nach der erfolgreichen Implementierung aller Anforderungen an das System, müssen die gesamten Funktionalitäten auf ihre funktionale und logische Richtigkeit überprüft werden. Wie bereits im Entwurf angedeutet, enthält jeder Webservice eine Testklasse. Diese Klasse wird als ein grundlegendes Qualitätskriterium des Services betrachtet. Weiterhin können auf dieser Weise die Webservice- Funktionalitäten einfacher getestet. Alle Methoden der jeweiligen Serviceschnittstelle werden in dieser Klasse aufgerufen und mit entsprechenden Parametern an den Server gesendet. Die gelieferten Ergebnisse werden überprüft und ausgewertet. Durch die Strukturierung dieser Methodenaufrufe können sehr leicht vollständige komplexe Testszenarien erstellt werden. Als Test- Framework wurde das TestNG mit seiner Eclipse- Erweiterung eingesetzt. Im Grunde besitzen alle Testklassen immer denselben Aufbau. Zuerst werden Servicemethoden aufgerufen, welche für die Erstellung und Speicherung eines Objektes zuständig sind. Im nächsten Schritt werden die Methoden für das Holen des gerade erstellten Objektes durchgeführt. Danach verwendet der Client das geholte Objekt, um die Methoden für die Bearbeitung des Objektes durchzuführen. Als nächster Schritt wird das bearbeitete Objekt aus dem System entfernt. In allen Testklassen wurde die *Assert*- Klasse des Frameworkes TestNG mit ihren statischen Methoden eingesetzt, um die erforderlichen Vergleiche bzw. Behauptungen durchzuführen. Eine Testklasse muss mindestens eine Methode besitzen, welche mit der *Test*- Annotation versehen ist. Diese Annotation kennzeichnet, dass diese Methode eine Testmethode ist. Bei jeder Behauptung können auch Meldungen hinterlegt werden, welche im Falle der unwahren Behauptung beim Ausführen angezeigt werden. Dieses Vorgehen erleichtert die Fehlersuche. Die verwendete Eclipse- Erweiterung zeigt auch beim Ausführen der Tests eine Liste mit erfolgreich bestandenen und gescheiterten Methoden. Es

lassen sich auch die Testmethoden einer Testklasse einzeln auf ihre Richtigkeit überprüfen.

Die Tests liefern die Erkenntnisse, wie vollständig die Webservice- Funktionalitäten hinsichtlich ihrer Geschäftlogik umgesetzt worden sind. Falls Fehler bei den Tests auftauchen, wird die Implementierung der betroffenen Methoden überarbeitet und nochmals getestet. Erst wenn alle Funktionalitäten der Webservices in den Tests ausgiebig getestet wurden und keine Fehler aufgetreten sind, gelten die Webservices als sicher und funktionstüchtig.

7. Fazit

Die Aufgabe dieser Diplomarbeit war die Konzeption und Realisierung eines Backend- Systems für ein DMS, wobei die Implementierung auf der Basis der serviceorientierten Architektur von open-EIS Plattform durchgeführt wurde. Die gestellte Aufgabe wurde komplett erfüllt und hat zu vielen interessanten Erkenntnissen geführt.

Der Einsatz von SOA mit ihren Webservices hat dazu beigetragen, dass das entstandene Backend- System einen modularen Aufbau erhalten hat. Das fertige System besteht aus einzelnen, in sich gekapselten Teilsystemen, welche miteinander agieren und gleichzeitig völlig unabhängig von ihren Implementierungen sind. Die Vorteile dieser Modularität liegen in der einfachen Erweiterbarkeit des Systems um weitere Teilsysteme, in dem unkomplizierten Austausch der Webservices sowie in der bequemen Zusammenstellung einer komplexen Clientanwendung über die vorhandenen Webservice- Funktionalitäten.

Durch die ausführliche Anforderungsanalyse, welche den kompletten Funktionsumfang des Systems dargestellt hat, konnte ein präziser Entwurf erstellt werden. Bereits bei der Modellierung des Datenmodells haben sich die einzelnen Teilsysteme herauskristallisiert. Nach der anschließenden Überführung in das Objektmodell und der Charakterisierung von Aufbau eines Webservices wurden die fertigen Teilsysteme konzipiert.

Dank des umfangreichen Entwurfs wurde die anstehende Realisierung der betroffenen Teilsysteme wesentlich vereinfacht. An dieser Stelle konnte der ganze Fokus auf die Anwendung von open-EIS- Funktionalitäten gerichtet werden. Nach der Implementierung aller Teilsysteme konnte durch die erstellten Tests nachgewiesen werden, dass die umgesetzten Funktionalitäten den Erwartungen an das Backend- System entsprachen.

Als eine sinnvolle Erweiterung kann die Einführung der Papierkorb- Funktionalität erarbeitet werden. Auf diesem Wege können die gelöschten Dokumente bzw. Mediadateien durch den Administrator wiederhergestellt werden. Zusätzlich wird die Überarbeitung der Datenbankabfragen bei der Suche nach Dokumenten zur Optimierung des Systems beitragen. Im Moment werden mehrere Abfragen durchgeführt und die Ergebnismengen außerhalb des Datenbank-Management-Systems manuell ausgewertet.

Die Herangehensweise sowie die durchgeführten Vorbereitungen und Maßnahmen für die Realisierung dieser Aufgabe zeigen, dass auf diese Art im Prinzip jedes System auf der Basis von SOA mittels Hermes- Framework umsetzbar ist.

III. Anlagen


```
 2  * IUserService.java
 5 package com.c4u.eis.dms.service.user.client;
 6
 7 import java.rmi.Remote;
13
15  * The definition for the open-EIS DMS user service.<br />
18 public interface IUserService extends Remote
19 {
21    * The service identifier.
23   String ID = "com.c4u.eis.dms.service.user.UserService";
24
26    * Creates and stores a new user in DB.
38   UserDO createUserDO(String aContextId, String aTransactionId, UserDO
     anUser)
39     throws NoPermissionException, UserServiceException, RemoteException;
40
42    * Update an user in DB.
54   UserDO updateUserDO(String aContextId, String aTransactionId, UserDO
     anUser)
55     throws NoPermissionException, UserServiceException, RemoteException;
56
58    * Remove an user from DB.
70   UserDO removeUserDO(String aContextId, String aTransactionId, String
     aGuid)
71     throws NoPermissionException, UserServiceException, RemoteException;
72
74    * Gets an user from DB.
86   UserDO getUserDO(String aContextId, String aTransactionId, String aGuid)
87     throws NoPermissionException, UserServiceException, RemoteException;
88
90    * Creates and stores a new group in DB.
102  GroupDO createGroupDO(String aContextId, String aTransactionId, GroupDO
     aGroup)
103    throws NoPermissionException, UserServiceException, RemoteException;
104
106   * Update an group in DB.
118  GroupDO updateGroupDO(String aContextId, String aTransactionId, GroupDO
     aGroup)
119    throws NoPermissionException, UserServiceException, RemoteException;
120
122   * Remove a group from DB.
134  GroupDO removeGroupDO(String aContextId, String aTransactionId, String
     aGuid)
135    throws NoPermissionException, UserServiceException, RemoteException;
136
138   * Gets a group from DB.
150  GroupDO getGroupDO(String aContextId, String aTransactionId, String aGuid)
151    throws NoPermissionException, UserServiceException, RemoteException;
152 }
```

Anlage 1: IUserService – Nutzerschnittstelle

```
 2 * ICategoryService.java
 5 package com.c4u.eis.dms.service.category.client;
 6
 7 import java.rmi.Remote;
15
17 * The definition for the open-EIS DMS category service.<br />
20 public interface ICategoryService extends Remote
21 {
23   * The service identifier.
25  String ID = "com.c4u.eis.dms.service.category.CategoryService";
26
28   * Creates and stores a new category in DB.
39  CategoryDO createCategoryDO(String aContextId, String aTransactionId,
    CategoryDO aCategory,
40      Collection<String> theKeywords)
41      throws NoPermissionException, CategoryServiceException,
    RemoteException;
42
44   * Update a category in DB.
55  CategoryDO updateCategoryDO(String aContextId, String aTransactionId,
    CategoryDO aCategory,
56      Collection<String> theKeywords)
57      throws NoPermissionException, CategoryServiceException,
    RemoteException;
58
60   * Remove a category from DB.
70  CategoryDO removeCategoryDO(String aContextId, String aTransactionId,
    String aGuid)
71      throws NoPermissionException, CategoryServiceException,
    RemoteException;
72
74   * Gets a category from DB.
84  CategoryDO getCategoryDO(String aContextId, String aTransactionId, String
    aGuid)
85      throws NoPermissionException, CategoryServiceException,
    RemoteException;
86
88   * Set the required permission for a category of the specified group.
    Note: already existing permissions on this category for
100  void setPermission(String aContextId, String aTransactionId, String
    aCategoryGuid, String aGroupGuid,
101     Collection<String> thePermissionGUIDs)
102     throws NoPermissionException, CategoryServiceException,
    RemoteException;
103
105  * Gets all category children with the "READ" permission for the specified
    user and the flag for specified permission. This
117  Map<CategoryDO, Boolean> getCategoryChildren(String aContextId, String
    aTransactionId, String aGuid,
118     String aPermission)
119     throws NoPermissionException, CategoryServiceException,
    RemoteException;
120
122  * Gets all documents from the category.
132  DocsInCategoryDO getAllDocumentsFromCategory(String aContextId, String
    aTransactionId, String aGuid)
133     throws NoPermissionException, CategoryServiceException,
    RemoteException;
134 }
```

Anlage 2: ICategoryService – Kategorieschnittstelle

```
 2 * IDocumentService.java
 5 package com.c4u.eis.dms.service.document.client;
 6
 7 import java.rmi.Remote;
15
17 * The definition for the open-EIS DMS document service.<br />
20 public interface IDocumentService extends Remote
21 {
23   * The service identifier.
25  String ID = "com.c4u.eis.dms.service.document.DocumentService";
26
28   * Creates and stores a new document in DB.
39  DocumentDO createDocumentDO(String aContextId, String aTransactionId,
    DocumentDO aDocument,
40       Collection<String> theKeywords)
41       throws NoPermissionException, DocumentServiceException,
    RemoteException;
42
44   * Update a document in DB.
55  DocumentDO updateDocumentDO(String aContextId, String aTransactionId,
    DocumentDO aDocument,
56       Collection<String> theKeywords)
57       throws NoPermissionException, DocumentServiceException,
    RemoteException;
58
60   * Remove a document from DB.
70  DocumentDO removeDocumentDO(String aContextId, String aTransactionId,
    String aGuid)
71       throws NoPermissionException, DocumentServiceException,
    RemoteException;
72
74   * Gets a document from DB.
84  DocumentDO getDocumentDO(String aContextId, String aTransactionId, String
    aGuid)
85       throws NoPermissionException, DocumentServiceException,
    RemoteException;
86
88   * Creates and stores a new media in DB.
100  MediaDO createMediaDO(String aContextId, String aTransactionId, String
    anUploadId, MediaDO aMedia,
101      Collection<String> theKeywords)
102      throws NoPermissionException, DocumentServiceException,
    RemoteException;
103
105  * Update a media in DB.
117  MediaDO updateMediaDO(String aContextId, String aTransactionId, String
    anUploadId, MediaDO aMedia,
118      Collection<String> theKeywords)
119      throws NoPermissionException, DocumentServiceException,
    RemoteException;
120
122  * Remove a media from DB.
132  MediaDO removeMediaDO(String aContextId, String aTransactionId, String
    aGuid)
133      throws NoPermissionException, DocumentServiceException,
    RemoteException;
134
136  * Gets a media from DB.
146  MediaDO getMediaDO(String aContextId, String aTransactionId, String aGuid)
```

```
147       throws NoPermissionException, DocumentServiceException,
    RemoteException;
148
150     * Adds the new assignments between a document and the media files.
160   void addDoc2MediaLinks(String aContextId, String aTransactionId, String
    aDocumentGuid,
161       Collection<String> theMediaGuids)
162       throws NoPermissionException, DocumentServiceException,
    RemoteException;
163
165     * Adds a new assignment link between a media and the documents.
175   void addMedia2DocLinks(String aContextId, String aTransactionId, String
    aMediaGuid,
176       Collection<String> theDocumentGuids)
177       throws NoPermissionException, DocumentServiceException,
    RemoteException;
178
180     * Gets all assigned documents to this media considering the user rights.
190   Collection<DocumentDO> getAssignedDocuments(String aContextId, String
    aTransactionId, String aMediaGuid)
191       throws NoPermissionException, DocumentServiceException,
    RemoteException;
192
194     * Gets all assigned media files to this document considering the user
    rights.
204   Collection<MediaDO> getAssignedMedias(String aContextId, String
    aTransactionId, String aDocumentGuid)
205       throws NoPermissionException, DocumentServiceException,
    RemoteException;
206
208     * Removes the assignment links between a document and the media files.
218   void removeDoc2MediaLinks(String aContextId, String aTransactionId, String
    aDocumentGuid,
219       Collection<String> theMediaGuids)
220       throws NoPermissionException, DocumentServiceException,
    RemoteException;
221
223     * Removes the assignment links between a media and the documents.
233   void removeMedia2DocLinks(String aContextId, String aTransactionId, String
    aMediaGuid,
234       Collection<String> theDocumentGuids)
235       throws NoPermissionException, DocumentServiceException,
    RemoteException;
236
238     * Get the list of suitable keywords by the specified content of document.
248   List<String> getSuitableKeywordsForDocument(String aContextId, String
    aTransactionId, String aContent)
249       throws NoPermissionException, DocumentServiceException,
    RemoteException;
250
252     * Get the upload ID.
261   String prepareUploadMedia(String aContextId, String aTransactionId)
262       throws NoPermissionException, DocumentServiceException,
    RemoteException;
263
265     * Upload the media file via POST.
274   void uploadMedia(String aContextId, String aTransactionId, String
    anUploadId)
275       throws NoPermissionException, DocumentServiceException,
    RemoteException;
276 }
277
```

Anlage 3: IDocumentService – Dokumentenschnittstelle

```
 2 * IKeywordService.java
 5 package com.c4u.eis.dms.service.keyword.client;
 6
 7 import java.rmi.Remote;
16
18 * The definition of the keyword service to search open-EIS documents.
22 public interface IKeywordService extends Remote
23 {
25   * Holds the id of the keyword service.
27  String ID = "com.c4u.eis.dms.service.keyword.KeywordService";
28
30   * Adds the given keyword to specified document in specified role. If
   specified role is null the keyword will be assigned to
43  @SuppressWarnings("unchecked")
44  void addKeyword(String aContextId, DocumentHandleDO aDocumentHandleDO,
   String aKeyword, String aRoleId)
45      throws NoPermissionException, KeywordServiceException,
   RemoteException;
46
48   * Adds the given keywords to specified document in specified role. If
   given role is null keywords will be assigned to
61  @SuppressWarnings("unchecked")
62  void addKeywords(String aContextId, DocumentHandleDO aDocumentHandleDO,
   String[] theKeywords, String aRoleId)
63      throws NoPermissionException, KeywordServiceException,
   RemoteException;
64
66   * Gets all assigned keywords for given document (ignores roles).
75  @SuppressWarnings("unchecked")
76  String[] getAllKeywords(String aContextId, DocumentHandleDO
   aDocumentHandleDO)
77      throws NoPermissionException, KeywordServiceException,
   RemoteException;
78
80   * Gets an array of keywords which starts with given prefix.
90  String[] getKeywordsByPrefix(String aContextId, String aKeywordPrefix, int
   aMaxCount)
91      throws NoPermissionException, KeywordServiceException,
   RemoteException;
92
94   * Gets the keyword handle data object for given keyword. If no keyword
   exists it will be created.
103  DocumentHandleDO<? extends AbstractContent> getKeywordHandleDO(String
   aContextId, String aKeyword)
104     throws NoPermissionException, KeywordServiceException,
   RemoteException;
105
107  * Search the documents by keywords. It is possible to determine the AND-,
   OR-, XOR- and NOT-lists with the keywords, which
124  KeywordSearchResultDO search(String aContextId, String[]
   theDocumentTypIds, boolean aVisibleStatus,
125      List<String> theANDList, List<String> theORList, List<String>
   theXORList, List<String> theNOTList, int aLimit)
126     throws NoPermissionException, KeywordServiceException,
   RemoteException;
127
129  * Removes the given keyword to specified document for given role (same
   keyword but different role for given document will not
140  @SuppressWarnings("unchecked")
```

```
141   void removeKeyword(String aContextId, DocumentHandleDO aDocumentHandleDO,
      String aKeyword, String aRoleId)
142       throws NoPermissionException, KeywordServiceException,
      RemoteException;
143
145    * Removes given keywords to specified document for given roles (assigned
      keywords in different roles are not effected).
155   @SuppressWarnings("unchecked")
156   void removeKeywords(String aContextId, DocumentHandleDO aDocumentHandleDO,
      String[] theKeywords, String aRoleId)
157       throws NoPermissionException, KeywordServiceException,
      RemoteException;
158
160    * Removes all unused keywords from the system. That means, that these
      keywords are not contained in the
168   void removeUnusedKeywords(String aContextId)
169       throws NoPermissionException, KeywordServiceException,
      RemoteException;
170
172    * Get the list of document handle data objects.
183   @SuppressWarnings("unchecked")
184   List<DocumentHandleDO> getLastModifiedDocuments(String aContextId,
      String[] theDocumentTypIds,
185       String[] theLastActionFlags, int aLimit)
186       throws NoPermissionException, DocumentServiceException,
      RemoteException;
187 }
188
```

Anlage 4: IKeywordService – Suchbegriffschnittstelle

IV. Literaturverzeichnis

[Ce07a] Cerquetti, Lavinio: The Hermes SOA Framework. Firmeninterne Dokumentation, 2007

[Ce07b] Cerquetti, Lavinio: Sabina Test Framework. Firmeninterne Dokumentation, 2007

[Co07] Community4you GmbH: open-EIS Plattform. Firmeninterne Dokumentation, 2007

[Fe04] Feyhl, Achim W.: Management und Controlling von Softwareprojekten. – 2. überarb. und erw. Auflage. – Wiesbaden: Betriebswirtschaftlicher Verlag Dr. Th. Gabler/GWV Fachverlage GmbH, 2004

[Go04] Götzer, Klaus … : Dokumenten-Management. – 3., vollständig überarbeitete und erweiterte Auflage. – Heidelberg: dpunkt.verlag, 2004

[Gu02] Gulbins, Jürgen; Seyfried, Markus; Strack-Zimmermann, Hans: Dokumenten-Management. – 3., überarb. und erw. Auflage. – Berlin; Heidelberg; New York; Barcelona; Hongkong; London; Mailand; Paris; Tokio: Springer- Verlag, 2002

[Ha07] Härtzer, Danilo: Framework Onuris. Firmeninterne Dokumentation, 2007

[Hi04] Hindel, Bernd,… : Basiswissen Softwaremanagement. – 1. Auflage. – Heidelberg: dpunkt.verlag, 2004

[Jo08] Josuttis, Nicolai: SOA in der Praxis. System-Design für verteilte Geschäftsprozesse. – 1. Auflage. – Heidelberg: dpunkt.verlag, 2008

[Ka99] Kampffmeyer, Ulrich; Merkel, Barbara: Dokumenten-Management. – 2., überarb., aktualisierte und erw. Aufl. – Hamburg: Project Consult GmbH, 1999

[Kr95] Kränzle, Hans-Peter: Dokumentenmanagement: Technik und Konzepte. HMD - Praxis Wirtschaftsinformatik, 181, 1995

[Ma07] Masak, Dieter: SOA? Serviceorientierung in Business und Software. – Heidelberg: Springer-Verlag, 2007

[Me04] Mellis, Werner: Projektmanagement der SW- Entwicklung. – 1. Auflage. – Wiesbaden: Vieweg, 2004

[Me08] Melzer, Ingo: Service-orientierte Architekturen mit Web Services. – 3. Auflage. – Heidelberg: Spektrum Akademischer Verlag, 2008

[Ni96] Niemann, Klaus D.: Client-Server-Architektur. – 2., verb. Auflage. – Braunschweig: Vieweg, 1996

[Se06] Seemann, Jochen; von Gudenberg, JürgenWolff: Software-Entwurf mit UML 2. – 2. Auflage. – Berlin; Heidelberg: Springer-Verlag, 2006

[vB08] vom Brocke, Jan: Serviceorientierte Architekturen - SOA. – München: Verlag Franz Vahlen GmbH, 2008

[Ve00] Versteegen, Gerhard: Projektmanagement mit dem Rational Unified Process. – Berlin; Heidelberg: Springer- Verlag, 2000

[al09] Altenhofen, Christoph: DMS in der Technischen Dokumentation. URL: <http://www.doku.net/artikel/dokumenten.htm#Einführung>, verfügbar am 17.10.2009

[be10] berlecon: SOA in der Praxis - Wie Unternehmen SOA erfolgreich einsetzen. URL: <http://www.berlecon.de/research/reports.php ?we_objectID=271>, verfügbar am 12.01.2010

[fr10a] Fraunhofer IESE: Funktionale Anforderungen erheben. URL: <http://www.re-wissen.de/Wissen/Anforderungserhebung/Praktiken/Funktionale_Anforderungen_erheben.html>, verfügbar am 19.10.2009

[fr10b] Fraunhofer IESE: Funktionale Anforderungen erheben. URL: <http://www.re-wissen.de/Wissen/Anforderungserhebung/Praktiken/Nichtfunktionale_Anforderungen_erheben.html>, verfügbar am 19.10.2009

[it10] IT- Wissen: Drei-Schichten-Architektur. URL: <http://www.itwissen.info/definition/lexikon/Drei-Schichten-Architektur-three-tier-architecture.html>, verfügbar am 21.01.2010

[ja10] Sun Microsystem : Remote Method Invocation (RMI). URL: <http://java.sun.com/javase/technologies/core/basic/rmi/index.jsp>, verfügbar am 24.01.2010

[je10] Jeckle, Mario: Semantik, Odem einer Service-orientierten Architektur. URL: <http://www.jeckle.de/semanticWebServices/ intro.html>, verfügbar am 13.01.2010

[lo10] Logeman, Dennis: Systementwurf. URL: <http://www-is.offis.uni-oldenburg.de/~dibo/teaching/pg-mpig/zwischenbericht-a/node221.html>, verfügbar am 13.01.2010

[ri09] Riggert, W.: Dokumentenmanagement. URL: <http://www.wi.fh-flensburg.de/fileadmin/dozenten/Riggert/bildmaterial/ Dokumenten-management/3-Manage-DMS.pdf>, verfügbar am 08.10.2009

[ww10] W3C: Web Services Architecture. URL: <http://www.w3.org/TR/ws-arch/>, verfügbar am 13.01.2010

Erklärung

Ich erkläre, dass ich die vorliegende Arbeit selbständig und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe.

Chemnitz, 29.01.2010