DAVe

Marvin Luchs

Bachelorthesis

Konzeption und Entwicklung einer
webbasierten Dokumenten- und Aufgabenverwaltung
für das Studierendenparlament der HAW Hamburg

Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

DEPARTMENT INFORMATION

Bachelorarbeit

Konzeption und Entwicklung einer webbasierten Dokumenten- und Aufgabenverwaltung für das Studierendenparlament der HAW Hamburg

vorgelegt von
Marvin Luchs

Studiengang Medien und Information

erste Prüferin: Prof. Dr. Franziskus Geeb
zweiter Prüfer: Prof. Dr. Ulrike Spree

Hamburg, Oktober 2009

Abstract

Gegenstand dieser Arbeit ist die Konzeption und Entwicklung eines Informationssystems für das Studierendenparlament der HAW Hamburg. Die Ergebnisse eines im Vorfeld durchgeführten Information Audits haben gezeigt, dass von den Parlamentsmitgliedern benötigte Informationen und Dokumente oftmals vorhanden sind, jedoch kein schneller und einfach gestalteter Zugriff auf diese möglich ist. Auch das Abhandenkommen von Wissen aufgrund der hohen Personalfluktuation in der Studierendenschaft stellt ein gravierendes Problem für die Arbeit des Parlaments dar. Um dieser Situation zu begegnen, wird ein Anforderungskatalog formuliert, welcher den Ausgangspunkt für die Konzeption einer integrierten Dokumenten- und Aufgabenverwaltung bildet. Das Ziel ist dabei die Entwicklung einer sinnvoll strukturierten Informationsarchitektur mit differenzierter Rechtevergabe und benutzerfreundlicher, optisch ansprechender Oberfläche. Die Umsetzung dieses Konzepts erfolgt in Form einer Webanwendung, die auf dem PHP-Framework Symfony basiert. Für die Abbildung der Informationsarchitektur wird ein Datenbankschema erarbeitet, auf welches über ein mit Propel realisiertes objektrelationales Mapping zugegriffen wird. Die Steuerungsschicht wird weitestgehend mit den von Symfony bereitgestellten Möglichkeiten umgesetzt. Zu den Komponenten, die über diese hinausgehen, gehören die mit Zend Lucene realisierte Suchfunktion, die in der Modellschicht implementierte Rechteverwaltung sowie ein Ereignisprotokoll mit dynamisch generierten Logeinträgen. Die Umsetzung der grafischen Oberfläche erfolgt unter Verwendung gängiger Konzepte und Standards mit einem Schwerpunkt auf klarer Strukturierung der bereitgestellten Informationen.

Schlagworte: Dokumentenmanagement, Informationsarchitektur, Informationssystem, Objektrelationales Mapping, Propel, Studierendenparlament, Symfony, Webanwendung

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listingverzeichnis

1 Einleitung

Die vorliegende Bachelorthesis befasst sich mit der Konzeption und Entwicklung eines Informationssystems für das Studierendenparlament der HAW Hamburg. Die Idee, das Wissensmanagement in Form einer auf die Bedürfnisse des Organs ausgerichteten Anwendung zu implementierten, ist während der Tätigkeit des Autors als studentischer Vertreter innerhalb dieses Parlaments entstanden. Immer wieder kommt es zu Situationen, in denen wichtige Informationen nicht zur Verfügung stehen, Ideen und Aufgaben vergessen werden oder Unklarheit darüber herrscht, welche Version eines Dokuments tatsächlich verabschiedet worden ist.

In vielen Fällen existieren die benötigten Informationen zwar, sind aber nicht griffbereit oder im Bewusstsein der Abgeordneten. Das Problem ist also weniger der Mangel an Informationen als vielmehr der Zugang zu denselben. Es gilt, das Wissen des Studierendenparlaments so aufzubereiten, dass es genutzt werden kann.

Da die im Studierendenparlament erzeugten und verarbeiteten Informationen zum größten Teil in Form von Dokumenten vorliegen, soll ein Informationssystem eingeführt werden, das einen einfachen und unkomplizierten Zugriff auf selbige bereitstellt. Beabsichtigt ist die Verknüpfung dieses Systems mit einer Task-Management-Software, über welche die in den Sitzungen geäußerten Ideen und Aufgaben gesammelt und verwaltet werden können. So ist das Konzept einer integrierten **D**okumenten- und **A**ufgaben**ve**rwaltung, kurz DAVe, entstanden.

Die Umsetzung dieses Konzepts erfolgt in einer eigens für das Studierendenparlament entwickelten Anwendung. Der Vorteil hierbei besteht zum einen darin, dass kein kostspieliges kommerzielles System eingeführt werden muss und zum anderen verringert die exakte Anpassung an die Nutzerbedürfnisse den Einarbeitungsaufwand für die Anwender. Der Benutzer wird nicht durch überflüssige Funktionen irritiert und die in der Anwendung verwendeten Begrifflichkeiten entsprechen dem parlamentsinternen Sprachgebrauch.

Den Ausgangspunkt dieser Arbeit bildet ein Information Audit. Dessen Ergebnisse, welche im Kapitel ***2 Ausgangssituation*** vorgestellt werden, umfassen eine detaillierte

Darstellung der Problemsituation im Studierendenparlament und allgemeine Anforderungen an ein Informationssystem, welches diesen Problemen begegnen soll. Ergänzt um weitere Aspekte, die im Information Audit nicht genannt werden, für die Konzeption aber dennoch relevant sind, formuliert dieser Abschnitt den konkreten Auftrag für die Entwicklung von DAVe. Der Volltext des Information Audits befindet sich in ***Anhang D Browserstatistik***.

Kapitel ***3 Konzept*** beschreibt den Entwurf, der für die Umsetzung dieses Auftrags als Planungsgrundlage dient. Eine Zielgruppendefinition beantwortet noch offene Fragen über den späteren Benutzerkreis der Anwendung. Die nachfolgend beschriebene Informationsarchitektur beschäftigt sich anschließend mit der Struktur der Anwendung: Welche Informationsobjekte werden mit dem System verwaltet, wie stehen sie in Verbindung zueinander und wie sieht das Gerüst aus, das diese Inhalte bereitstellt? Den Abschluss bildet eine detaillierte Darstellung über den Funktionsumfang von DAVe und eine klare Abgrenzung, welche Ziele innerhalb dieser Arbeit nicht verfolgt werden.

Mit der Realisierung befasst sich zunächst der Abschnitt ***4 Technische Umsetzung*** dieser Arbeit. Den Anfang bilden hierbei die Vorstellung verwendeten Frameworks sowie die Gründe für deren Wahl. Es folgt eine Beschreibung der Modellschicht des Systems mit einer Erläuterung des Datenbankschemas und dessen Abbildung auf ein Objektmodell. Den Abschluss bildet eine Einführung in die Steuerungsschicht der Anwendung und die detaillierte Vorstellung einzelner Komponenten.

Das Kapitel ***5 Visuelle Umsetzung*** widmet sich der Schnittstelle zum Benutzer. Entscheidungen während der Entwicklung des grafischen Konzepts werden begründet und das Seitendesign vorgestellt. Anschließend wird näher auf Umsetzung einzelner Navigationskomponenten eingegangen.

In der Schlussbetrachtung resümiert eine kritische Reflexion zunächst die Ergebnisse dieser Arbeit und bewertet diese. Der Ausblick richtet anschließend den Blick nach vorn und gibt Anregungen für die weitere Entwicklung von DAVe.

2 Ausgangssituation

2.1 Struktur der verfassten Studierendenschaft

Das Studierendenparlament (StuPa) nimmt ihm Rahmen dieser Arbeit die Rolle des Auftraggebers ein. Es ist das höchste Organ der Studierendenschaft der HAW Hamburg und stellt die direkte Vertretung der Studierenden auf Hochschulebene dar.

Zu den wichtigsten Aufgaben des StuPas gehören die Wahl und Kontrolle des Allgemeinen Studierendenausschusses (AStA) sowie Entscheidungen über die Finanzen und rechtlichen Grundlagen der Studierendenschaft. Zu diesen gehören die Satzung der Studierendenschaft sowie die in ihr vorgesehenen Ordnungen, wie die Wahlordnung, die Wirtschaftsordnung und die Fachschaftsrahmenordnung. Die hierfür vorgesehenen Sitzungen finden in der Regel alle zwei Wochen statt. Das StuPa nimmt somit in der Studierendenschaft eine legislative und kontrollierende Funktion ein.

Das ausführende Organ der Studierendenschaft bildet der Allgemeine Studierendenausschuss. Er unterhält die Geschäftsräume der Studierendenschaft und ist für das Tagesgeschäft verantwortlich. Vertreter des AStA sind an Werktagen regelmäßig in den Räumen der Studierendenschaft anzutreffen und stellen die direkten Ansprechpartner für die Studenten dar.

Das Verhältnis von StuPa und AStA ist vergleichbar mit dem Verhältnis von Bundestag und Bundesregierung bzw. der öffentlichen Verwaltung.

Zu den weiteren hochschulweiten Gremien gehören beispielsweise der Wirtschaftsrat, die Wahlleitung und die Fachschaftsrätekonferenz. Darüber hinaus existieren auch auf Fakultäts- und Departmentsebene studentische Vertretungen. Diese spielen in diesem Projekt vorerst aber nur eine sekundäre Rolle.

2.2 Problemstellung

Wie in der Einleitung bereits geschildert, bestehen für das Studierendenparlament die größten Schwierigkeiten in der Bereitstellung eines schnellen Zugriffs auf vorhandene Informationen. Bedingt wird dies durch die dezentrale Struktur des Organs und die Art der Bereitstellung der Dokumente. Da die Mitglieder des Studierendenparlaments ausschließlich für die Sitzungen zusammentreten, findet der Austausch von Dokumenten entweder in diesen Sitzungen oder per E-Mail statt. Aufgrund der geringen Sitzungsfrequenz können Dokumente somit nur auf elektronischem Wege zeitnah zugestellt werden. Eine zentrale Ressource, die allen Mitgliedern jederzeit und unabhängig voneinander sämtliche Dokumente bereithält, existiert nicht. Ein einfacher und schneller Zugriff auf Dokumente ist nur dann gewährleistet, wenn die Mitglieder Ordnung in ihren Unterlagen halten und somit eigenverantwortlich ein steter Zugriff darauf garantiert ist. Nicht zuletzt, weil die Tätigkeit im Studierendenparlament eine ehrenamtliche Arbeit ist, sind jedoch nur wenige Mitglieder bereit, den entsprechenden Aufwand zu erbringen.

Aus denselben Gründen besteht regelmäßig Unklarheit darüber, welche Version eines Dokuments die aktuell gültige bzw. welche der letzte Arbeitsentwurf ist. Diese Frage kann in der Regel zwar durch eine Anfrage beim StuPa-Präsidium oder beim AStA geklärt werden, eine zeitnahe Antwort ist jedoch nicht garantiert.

Eine weitere Problemquelle ist die hohe Personalfluktuation im Parlament. Besonders der Wechsel von einer Legislaturperiode in die nächste hat sich hierbei als signifikantes Informationsloch herausgestellt, da die studentischen Vertreter jährlich neu gewählt werden und viele nur für eine Wahlperiode im Amt bleiben. Informationen darüber, welche Änderungen an der Satzung oder einer Ordnung bei der nächsten Bearbeitung noch durchgeführt werden müssen, existieren meist aber nur in den Köpfen der Mitglieder. Scheidet ein Mitglied aus dem Studierendenparlament aus, geht mit Ihm auch das Wissen über jene noch ausstehenden Aufgaben verloren.

Dies betrifft aber nicht nur die noch ausstehenden Bearbeitungen von Dokumenten, sondern auch die Gründe für zurückliegende Änderungen. So ist für Außenstehende oft nicht mehr ersichtlich, warum bestimmte Änderungen an Dokumenten vorgenommen wurden oder worin sich eine Arbeitsversion von der anderen unterscheidet.

Ein weiterer Punkt, den der Information Audit thematisiert, sind die Informationsflüsse innerhalb des Studierendenparlaments und der Austausch mit dem AStA. Hier hat sich

gezeigt, dass vor allem der Austausch zwischen AStA und regulären StuPa-Mitgliedern nicht ausreichend ist. Das StuPa-Präsidium hingegen fühlt sich aufgrund der regelmäßigen Teilnahme an den wöchentlichen Sitzungen des AStA besser informiert. Die Mitglieder sind somit auf die Auskünfte des Präsidiums angewiesen und über Entwicklungen im Tagesgeschäft der Studierendenschaft werden sie deshalb teilweise erst mit zwei Wochen Verzug in der nachfolgenden Sitzung informiert.

Jedoch zeigt der im Rahmen des Information Audits mehrfach geäußerte Wunsch nach Protokollen aus anderen Gremien, dass ein Bedürfnis der StuPa-Mitglieder besteht, zeitnäher und umfassender informiert zu werden. Umso wichtiger ist dieser Aspekt angesichts der Kontrollfunktion, welche das Parlament gegenüber dem AStA innehat.

Darüber hinaus konnte der Audit auch eine mangelnde Transparenz bei der Bearbeitung von Dokumenten aufdecken. Da die Ergebnisse dem Parlament meist erst nach abgeschlossener Bearbeitung vorgestellt werden, haben die Mitglieder, die nicht Teil der Arbeitsgruppe waren, kaum Gelegenheit, sich in den Bearbeitungsprozess einzubringen. Erst bei der ersten Lesung des Dokuments ist es den Mitgliedern somit möglich, ein Feedback zu den Ergebnissen der Arbeitsgruppe zu äußern. Umfangreiche nachträgliche Änderungen sind hierbei nicht selten die Folge. Diese bereits während des Bearbeitungsprozesses einzubringen, würde den Vorgang effizienter und zeitsparender gestalten.

2.3 Anforderungen

2.3.1 Fachliche Anforderungen

Das wichtigste Ziel, das mit der Entwicklung von DAVe verfolgt wird, ist die Umsetzung eines schnellen und einfachen Zugriffs auf Dokumente und Informationen. Als zentrale Wissensressource für das Studierendenparlament, perspektivisch aber auch für andere Gremien, ist DAVe jederzeit und von jedem Ort aus unkompliziert und ohne Umwege erreichbar. Realisiert werden soll dies in Form einer webbasierten Anwendung, welche die Nachteile der lokalen Speicherung umgeht, die Kompatibilität zu möglichst vielen Plattformen gewährleistet und nicht die Installation lokaler Anwendungen voraussetzt.

Als Informationssystem für das Studierendenparlament spielen primär die Bedürfnisse dieses Organs für die Entwicklung eine Rolle. Trotzdem sollen im Sinne einer besseren Kommunikation mit anderen Gremien auch diese die Möglichkeit haben, unabhängig

vom StuPa DAVe zu nutzen, um beispielsweise ihre Sitzungsprotokolle selbstständig in DAVe einpflegen.

Damit das StuPa DAVe dennoch auch für interne Zwecke nutzen kann, ist die Implementierung eines Rechtesystems notwendig, welches nicht freigegebene Dokumente vor dem Zugriff durch Andere schützt.

Um der Problematik verschiedener Dokumentversionen entgegenzuwirken, listet DAVe die einzelnen Versionen eines Dokuments chronologisch auf. Hierbei muss eindeutig erkennbar sein, welche Versionen freigegeben sind und welche lediglich Arbeitsversionen darstellen. Im Sinne einer besseren Dokumentation der Bearbeitungsprozesse und noch ausstehender Aufgaben ist die Implementierung einer Kommentarfunktion für Dokumente und Dokumentversionen empfehlenswert.

Die Akzeptanz des Systems beim Benutzer ist kritisch für den Erfolg von DAVe, da aufgrund der ehrenamtlichen Natur der Studierendenschaft und ihrer hohen Personalfluktuation die Verwendung einer bestimmten Software nicht ohne Weiteres als obligatorisch erklärt werden kann. Deshalb müssen die Bedürfnisse des Benutzers im Mittelpunkt stehen und die Barrieren zur Nutzung des Systems möglichst niedrig sein. So soll eine intuitive und benutzerfreundliche Bedienung des Systems sicherstellen, dass sich neue Anwender auch ohne Einweisung zurechtfinden. Durch eine E-Mail-Benachrichtigung über neue Dokumente bekommen die Benutzer auch weiterhin ohne eigenes Zutun nach dem Push-Prinzip ihre Informationen in den Posteingang geliefert. Der Zugriff auf die Dokumente muss hierbei unkompliziert und ohne Umwege möglich sein. Wenn auch in DAVe erst langwierig nach dem gewünschten Dokument gesucht werden muss, wird das System seiner Aufgabe nicht gerecht.

In der optischen Gestaltung sollte sich dies ebenfalls widerspiegeln. Neben der Funktionalität muss DAVe deshalb genauso durch sein Erscheinungsbild überzeugen und beim Anwender einen professionellen, gleichzeitig aber auch attraktiven Eindruck hinterlassen.

2.3.2 Systemtechnisches Umfeld

Bei der Entwicklung einer Webanwendung gibt es hinsichtlich des technischen Umfelds zwei Entscheidungen zu treffen: Für welche Browser die Software entwickelt wird und auf welcher technischen Basis.

Als wichtigste Orientierung für die Festlegung der zu unterstützenden Webbrowser dient die Browserstatistik der Website der HAW Hamburg. Da von der AStA-Website oder den Websites anderer Gremien keine zuverlässigen Informationen über die Browsernutzung zur Verfügung stehen, kommt jene Statistik der Zielgruppe von DAVe am nächsten. Zwar umfasst sie darüber hinaus auch andere Klientel, es ist jedoch davon auszugehen, dass sich die Merkmale eines Großteils der Besucherschaft der HAW-Website mit denen der zukünftigen Anwender von DAVe decken.

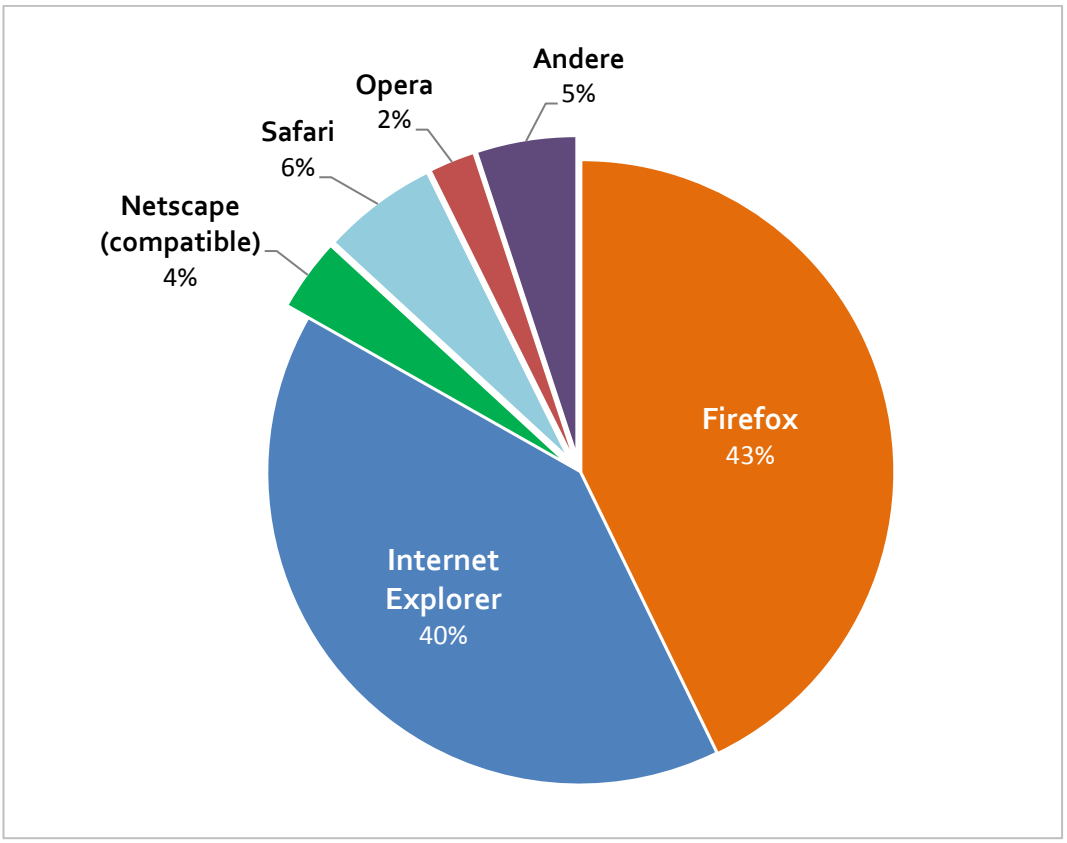


Abbildung 1: Anteile der Browser an der Gesamtzahl der Requests auf www.haw-hamburg.de (vgl. *Anhang D Browserstatistik*)

Abbildung 1 zeigt die Anteile der Browser gemessen an der Gesamtzahl der Anfragen für www.haw-hamburg.de. Demnach erzeugen der Internet Explorer mit 40 Prozent und der Mozilla Firefox mit 43 Prozent mehr als 80 Prozent aller Webanfragen. Weitere 6 Prozent entfallen auf den Mac-Browser Safari sowie 2 Prozent auf Opera. Der relativ neue Browser Chrome von Google wird in dieser Statistik nicht aufgeführt, da die Webtracking-Software der HAW Hamburg diesen noch nicht erkennt. Dessen Anteile werden mit anderen nicht identifizierten Browsern in den 4 Prozent, die auf „Netscape (compatible)“ entfallen, dargestellt.

Für die Entwicklung von DAVe vorranging ist demnach die Kompatibilität zum Microsoft Internet Explorer sowie zum Mozilla Firefox. Eine Unterstützung für Safari, Opera und

Google Chrome sollte zwar auch gegeben sein, da der Firefox aber für alle Plattformen zur Verfügung steht, kann diese als zweitrangig bewertet werden. Zudem existieren zwischen dem Firefox und den genannten, weniger verbreiteten Browsern erfahrungsgemäß nur geringfügige Unterschiede in der Interpretation von Webstandards.

Der Internet Explorer weist hingegen selbst zwischen einzelnen Versionen erhebliche Unterschiede in der Darstellung von Websites auf (vgl. LAZARIS 2009). Deshalb lohnt es sich, hier noch einmal einen genaueren Blick auf die Statistik zu werfen, um Aufschluss über die Verteilung im Segment des Internet Explorers zu erhalten.

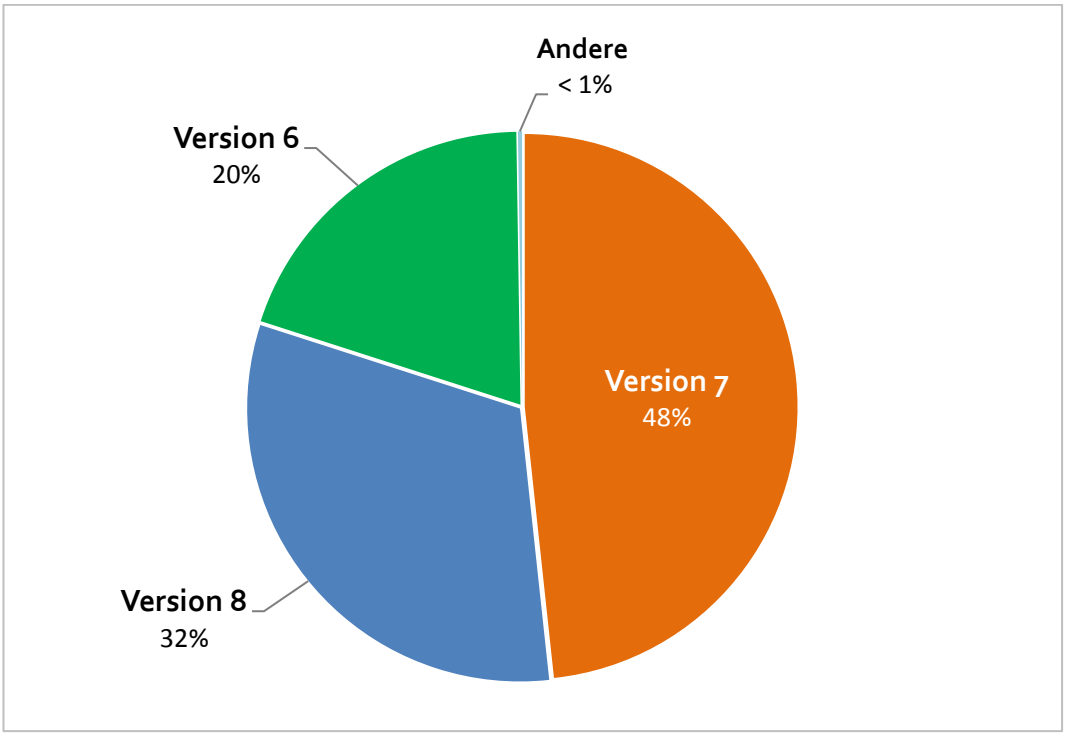


Abbildung 2: Anteile der einzelnen Versionen des Internet Explorers an der Zahl der mit dem Internet Explorer erfolgten Anfragen auf www.haw-hamburg.de (vgl. *Anhang D Browserstatistik*)

Wie in Abbildung 2 zu sehen ist, erfolgen fast die Hälfte aller Anfragen des Internet Explorers mit der Version 7 und 32 Prozent mit der aktuellen Version 8. Aber auch der mittlerweile stark in die Jahre gekommene Internet Explorer 6 weist noch einen Anteil von 20 Prozent auf. Eine Kompatibilität zu den Versionen 7 und 8 ist demnach obligatorisch, zumal die Verbreitung der Version 8 weiter ansteigen wird.

Bei der Version 6 ist das Gegenteil der Fall. Hier ist mit einem stetigen Sinken des Anteils zu rechnen, da die Unterstützung für diesen Browser zunehmend ausläuft und er auf neuen Betriebssystemen nicht mehr zur Verfügung steht. Auch aus Entwicklersicht sprechen einige Gründe dagegen, die Kompatibilität zum Internet Explorer 6 bei der Entwicklung von DAVe zu berücksichtigen.

Das größte Problem des Internet Explorer 6 ist die unvollständige und teilweise fehlerhafte Implementierung von aktuellen Standards wie XHTML, CSS 2.0 oder auch die fehlende Unterstützung für transparente PNG-Grafiken (vgl. LAZARIS 2009). Dies würde entweder aufwändige Browserweichen erforderlich machen oder Einschränkungen im Funktionsumfang zur Folge haben.

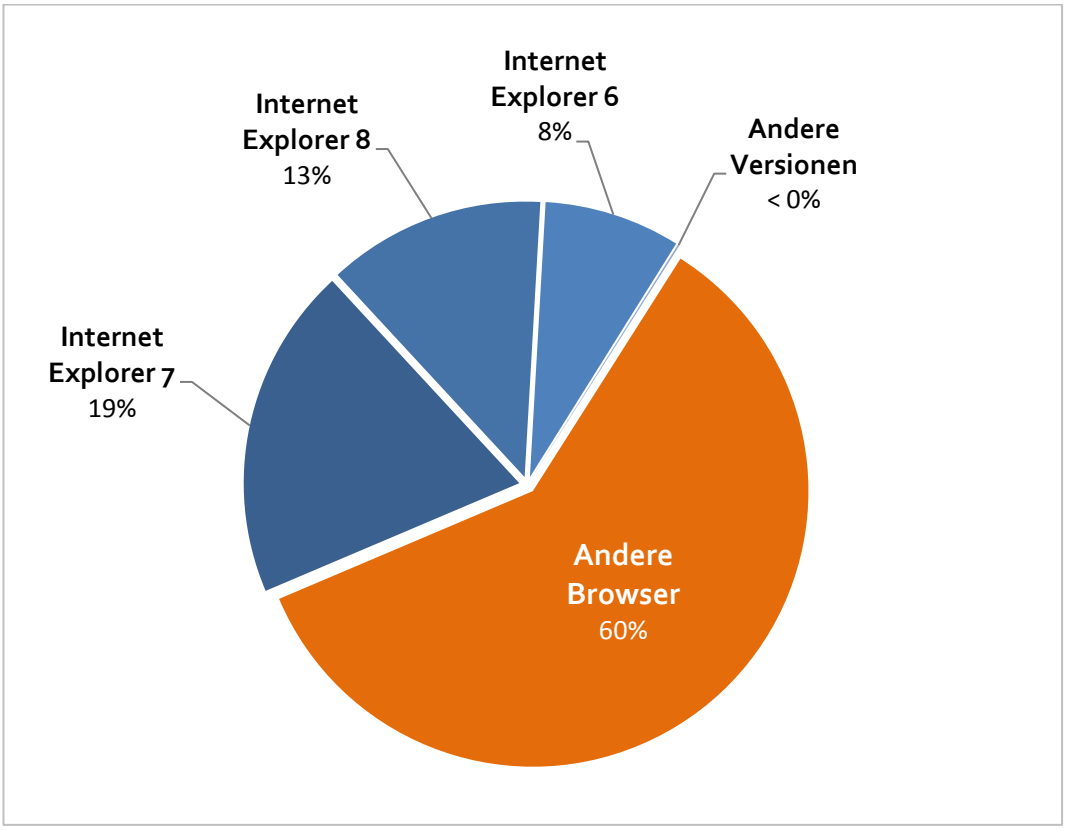


Abbildung 3: Anteile der einzelnen Versionen des Internet Explorers an der Gesamtzahl der Anfragen auf www.haw-hamburg.de (vgl. *Anhang D Browserstatistik*)

Umgerechnet auf alle Anfragen beträgt der Anteil des Internet Explorer 6 nur rund 8 Prozent (vgl. Abbildung 3). Im Hinblick auf den überschaubaren Nutzerkreis ist es somit sinnvoller, Anwender dieser Version bei einem Update oder einem Umstieg auf einen alternativen Browser zu unterstützen.

Auf der Serverseite soll DAVe für die Verwendung auf einem Windows Server System entwickelt werden, damit die Möglichkeit gegeben ist, die Anwendung auf dem Server des AStA zu installieren.

Als Programmiersprache kommt PHP in Verbindung mit einer MySQL-Datenbank zum Einsatz. Hier sprechen die weite Verbreitung und somit auch der Nachhaltigkeitsaspekt für die Verwendung. Darüber hinaus sind beide Produkte frei erhältlich, lassen sich sowohl auf Windows- als auch auf Linux-Systemen einsetzen und sind sehr gut dokumentiert.

2.3.3 Leistungsabgrenzung

An dieser Stelle soll noch einmal deutlich gemacht werden, welche Ziele mit der Entwicklung von DAVe, zumindest im Rahmen dieser Arbeit, nicht verfolgt werden.

DAVe hat nicht zum Ziel, die komplette Funktionalität eines Dokumentenmanagementsystems (DMS) bereitzustellen. Vielmehr sollen nur die Funktionalitäten implementiert werden, die bei der Arbeit des Studierendenparlaments auch Verwendung finden. Beispielsweise besitzt die Umsetzung einer Lebensdauer für Dokumente oder Dokumentversionen, wie sie in den meisten DMS vorhanden ist, für DAVe vorerst keine hohe Priorität. Aufgrund der Tatsache, dass DAVe eine Eigenentwicklung ist, besteht stets die Möglichkeit, eine solche Funktion bei Bedarf nachträglich zu implementieren.

Gleiches gilt für die Anwendergruppe. Der Fokus der Entwicklung liegt auf den Bedürfnissen des Studierendenparlaments, auch wenn andere Teile der Studierendenschaft ebenfalls Zugriff auf das System erhalten sollen. Zwar ist auf längere Sicht eine Ausweitung der Nutzung von DAVe durchaus möglich, jedoch wird diese Eventualität bei der Entwicklung nur zweitrangig berücksichtigt.

Jedes zusätzliche Feature trägt zur Komplexität der Anwendung bei und erhöht somit den Einarbeitungsaufwand und reduziert im schlimmsten Fall die intuitive Bedienung des Systems (vgl. HOEKMAN 2007, S. 54ff.). In diesem Sinne soll sich die Entwicklung von DAVe deshalb auf die Funktionen beschränken, die für die Umsetzung des Hauptziels, namentlich den schnelleren und einfacheren Zugriff auf Informationen, notwendig sind. Wie bereits erwähnt bietet eine Eigenentwicklung den Vorteil, zusätzliche Funktionen zu einem späteren Zeitpunkt zu implementieren, sollte sich dies als empfehlenswert erweisen.

3 Konzept

3.1 Zielgruppe

Als Anwender bilden die Mitglieder des Studierendenparlaments die Kernzielgruppe von DAVe. Eine weitere wichtige Zielgruppe wird der AStA sein, da insbesondere der Austausch zwischen StuPa und AStA über das System verbessert werden soll. Mittelfristig ist nicht ausgeschlossen, dass auch andere studentische Gruppen Zugriff auf DAVe erhalten, um mit dem Parlament Dokumente und Informationen austauschen zu können. Hierzu gehören beispielsweise Gremien wie der Wirtschaftsrat, die Fachschaftsrätekonferenz oder einzelne Fachschaftsräte der HAW Hamburg. Grundsätzlich ist somit festzuhalten, dass DAVe einen vergleichsweise engen Nutzerkreis anspricht.

Gemeinsam haben alle Zielgruppen, dass ihnen ausschließlich Studenten angehören. Der durchschnittliche Anwender ist erfahrungsgemäß Anfang bis Mitte zwanzig, gut gebildet und besitzt die Fähigkeit, sich eigenständig in neue Sachverhalte einzuarbeiten. Für die Konzeption von DAVe bedeutet dies, dass trotz der unterschiedlichen Fachrichtungen, aus denen die Studenten kommen, eine ausreichende Kompetenz im Umgang mit Webanwendungen gegeben ist, sofern gängige Konzepte verwendet und bei komplexeren Vorgängen Hilfestellungen angeboten werden. Um gleichermaßen auch erfahreneren Anwendern entgegenzukommen, bieten sich Schleichwege an, mit denen häufig zu verrichtende Aufgaben effizienter erledigt werden können (vgl. JACOBSEN 2007, S. 115ff.). Diese sind jedoch zweitrangig und können auch in einer nachfolgenden Version umgesetzt werden.

3.2 Informationsarchitektur

3.2.1 Informationsobjekte

Um das Wissen, das in einem Informationssystem abgelegt wird, zu strukturieren, werden Informationsobjekte definiert, welche verschiedenen Arten von Informationen entsprechen. Für DAVe wurde ein Informationsmodell konzipiert, das alle in den Anforderungen genannten Informationsarten beinhaltet und in einen logischen Zusammenhang bringt (s. Abbildung 4). Dabei werden bekannte Metaphern wie Ordner und Dokumente verwendet, die dem Benutzer dabei helfen, ein mentales Modell der Anwendung zu entwickeln (vgl. BATLEY 2007, S. 124ff.).

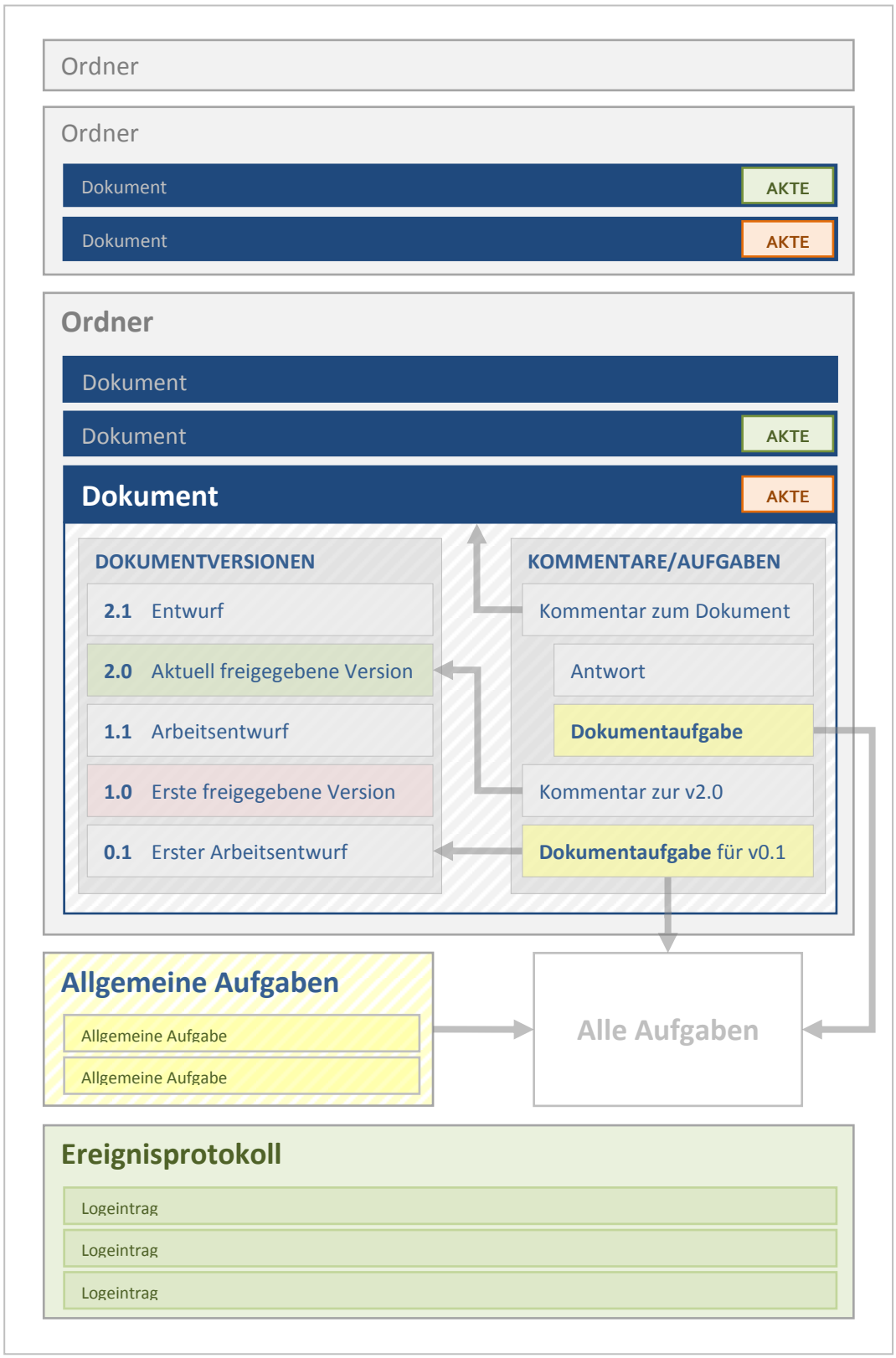


Abbildung 4: Schematische Darstellung des Informationsmodells

Das wichtigste Informationsobjekt in diesem Modell ist das **Dokument**. Es repräsentiert in DAVe jedoch nicht ein Schriftstück, sondern dient als Container, welcher die verschiedenen Versionen eines Schriftstücks gruppiert. Das Dokument selbst umfasst dabei nur die in Tabelle 1 aufgeführten Eigenschaften.

Eigenschaften	Beschreibung
Akte (optional)	Eine Akte, der das Dokument zugeordnet wurde.
Bearbeitet am	Das Datum der letzten Bearbeitung (dies schließt auch neue Dokumentversionen, Kommentare oder Aufgaben ein).
Beschreibung (optional)	Eine kurze Inhaltsangabe, die das Dokument beschreibt.
Dokumenttyp (optional)	Ein Dokumenttyp, dem das Dokument zugewiesen wurde (bspw. Protokoll, Vorlage).
Erstellt am	Das Datum der Erzeugung.
Erstellt von	Der Benutzer, der das Dokument erstellt hat.
Titel	Der Titel des Dokuments.

Tabelle 1: Eigenschaften des Informationsobjekts „Dokument“

Das Schriftstück selbst, das beispielsweise in Form einer Word- oder PDF-Datei vorliegt, wird von der **Dokumentversion** beschrieben und referenziert. Mit jeder Bearbeitung eines Schriftstücks wird innerhalb des Dokument-Containers eine neue Dokumentversion erstellt. Zusätzlich zu der eigentlichen Datei werden in der Dokumentversion noch weitere Informationen zu dem Schriftstück gespeichert, welche in Tabelle 2 dargestellt sind.

Eigenschaften	Beschreibung
Bemerkung	Eine Anmerkung zu den Änderungen oder zum Status der Dokumentversion.
Eingestellt am	Das Datum der Erzeugung.
Eingestellt von	Der Benutzer, der die Dokumentversion eingestellt hat.
Öffentlich	Beschreibt, ob das Dokument öffentlich zugänglich oder nur für den internen Gebrauch bestimmt ist.
Status	Beschreibt, ob es sich um eine freigegebene Dokumentversion oder um einen Arbeitsentwurf handelt.
Versionsnummer	Die eindeutige Versionsnummer der Dokumentversion.

Tabelle 2: Eigenschaften des Informationsobjekts „Dokumentversion“

Jede Dokumentversion wird durch eine Versionsnummer eindeutig innerhalb des Dokuments identifiziert. Die Versionsnummer besteht aus zwei Teilen, die durch einen Punkt voneinander getrennt sind. Der vordere Teil erhöht sich mit jeder freigegebenen

Version, der hintere Teil erhöht sich mit jedem neuen Entwurf und fängt mit Entwürfen, die einer freigegebenen Version folgen, wieder bei eins an. Tabelle 3 verdeutlicht diese Systematik anhand eines Beispiels.

Versionsnummer	Status	Beschreibung
0.1	Entwurf	Die initiale Dokumentversion, als Entwurf gekennzeichnet
0.2	Entwurf	Ein weiterer Entwurf
1.0	Freigegeben	Die erste als freigegeben markierte Version
1.1	Entwurf	Der erste Entwurf nach der freigegeben Version
2.0	Freigegeben	Die nächste freigegebene Version
3.0	Freigegeben	Die aktuell freigegebene Version
3.1	Entwurf	Der aktuellste Arbeitsentwurf

Tabelle 3: Beispiel für die Versionshistorie eines Dokuments

Die Dokumente werden in einer aus **Ordnern** bestehenden Baumstruktur abgelegt, wie es der Benutzer von seinem Betriebssystem kennt. Diese Struktur beginnt bei einer für den Anwender nicht sichtbaren Wurzel, d.h. alle Ordner, die unter der Wurzel angesiedelt sind, bilden die erste Ebene des für den Benutzer sichtbaren Ordnerbaums. Auf jeder Ebene der Ordnerhierarchie können neue Ordner angelegt werden, so dass die Baumstruktur frei den Bedürfnissen angepasst werden kann. Abgesehen von Informationen über ihre Position innerhalb der Ordnerhierarchie umfasst das Informationsobjekt eines Ordners die in Tabelle 4 aufgeführten Eigenschaften.

Eigenschaften	Beschreibung
Beschreibung (optional)	Eine kurze Beschreibung der Funktion des Ordners.
Name	Der Name des Ordners.
Sortierung	Die standardmäßige Sortierung für die in dem Ordner abgelegten Dokumente.

Tabelle 4: Eigenschaften des Informationsobjekts „Ordner"

Neben den Ordnern haben die Anwender zudem die Möglichkeit, Dokumente ordnerübergreifend in Akten zu organisieren. Mit der Angabe einer **Akte** für ein Dokument lassen sich unabhängig von der Zugehörigkeit zu einem Ordner Dokumente zu einer Thematik zusammenfassen. Wird beispielsweise das Protokoll zu einer Sitzung in einem anderen Ordner abgelegt als die Einladung und die zuvor eingereichten Anträge für die Sitzung, so lassen sich diese Dokumente trotzdem über die Angabe einer gemeinsamen Akte „Sitzung vom 11. März 2009" zu einer Gruppen zusammenfassen. Damit können aus einem Dokument heraus über den Aufruf der Akte thematisch verwandte Dokumente ermittelt werden.

Die Erstellung einer neuen Akte erfolgt automatisch, wenn der Benutzer den Namen der Akte beim Erstellen oder Bearbeiten eines Dokuments angibt und bisher noch keine Akte mit dem angegebenen Namen existiert. Abgesehen vom Namen weisen Akten keine weiteren Eigenschaften auf.

Eigenschaften	Beschreibung
Name	Der Name des Ordners.

Tabelle 5: Eigenschaften des Informationsobjekts „Akte"

Für eine bessere Dokumentation und zum Festhalten von Ideen und Anregungen ist es möglich, **Kommentare** zu erstellen. Diese können sich auf das ganze Dokument, einzelne Dokumentversionen oder im Sinne einer Antwort auch auf andere Kommentare beziehen. Die von der Antwortfunktion erzeugten Beziehungen zwischen Kommentaren werden hierarchisch abgebildet. Antworten auf Kommentare erscheinen demnach als Objekt, welches dem ursprünglichen Kommentar untergeordnet ist. Neue Kommentare mit Bezug auf das Dokument oder eine Dokumentversion sowie Kommentare ohne Bezug werden auf der obersten Ebene einsortiert. Die Eigenschaften eines Kommentars listet Tabelle 6 auf.

Eigenschaften	Beschreibung
Text	Der Text des Kommentars.
Benutzer	Der Autor des Kommentars.
Erstellt am	Das Datum, an dem der Kommentar eingestellt wurde.
Antwort auf (optional)	Das übergeordnete Kommentar-Element.
Dokument	Das Dokument, auf welches sich der Kommentar bezieht.
Dokumentversion (optional)	Die Dokumentversion, auf welche sich der Kommentar bezieht.
Bezug (optional)	Ein Vermerk, auf welchen Teil des Schriftstücks sich der Kommentar bezieht.

Tabelle 6: Eigenschaften des Informationsobjekts „Kommentar"

Mit dem Informationsobjekt „Kommentar" eng verwandt ist die **Aufgabe**. Sie unterscheidet sich von Kommentaren durch zusätzliche aufgabenspezifische Eigenschaften und kann zudem auch unabhängig von Dokumenten und Dokumentversionen als „allgemeine Aufgabe" auftreten. Als Dokumentaufgabe fügt sie sich nahtlos in den Kommentarbaum des Dokuments ein, so dass Aufgaben hier in Form von Kommentaren erscheinen, die als Aufgaben gekennzeichnet sind. Die zusätzlichen Eigenschaften des Aufgabenobjekts beschreibt Tabelle 7.

Eigenschaften	Beschreibung
Erledigt	Beschreibt, ob die Aufgabe als erledigt markiert wurde oder nicht.
Erledigt von	Der Benutzer, der die Aufgabe als erledigt markiert hat.
Erledigt am	Das Datum, an dem die Aufgabe als erledigt markiert wurde.
Fällig am (optional)	Das Datum, zu dem die Aufgabe fällig wird.

Tabelle 7: Eigenschaften des Informationsobjekts „Aufgabe"

Alle wichtigen Vorgänge, wie beispielsweise die Erstellung oder Bearbeitung eines Dokuments, werden in einem Ereignisprotokoll aufgezeichnet. Für jedes Ereignis wird hierbei ein **Logeintrag** erstellt, welcher unter Angabe des Benutzers und des Zeitpunkts die Art der Aktion sowie das betroffene Informationsobjekt dokumentiert. Tabelle 8 listet die Eigenschaften der Logeinträge auf.

Eigenschaften	Beschreibung
Zeitpunkt	Der Zeitpunkt, zu dem der Vorgang stattgefunden hat.
Benutzer	Der Benutzer, der den Vorgang durchgeführt hat.
Objekt	Das von dem Vorgang betroffene Objekt.
Aktion	Die Art des Vorgangs (bspw. bearbeitet oder gelöscht).

Tabelle 8: Eigenschaften des Informationsobjekts „Logeintrag"

Mit dem Logsystem verknüpft ist eine Benachrichtigungskomponente, welche die Benutzer über bestimmte Ereignisse per E-Mail informiert. So ist beispielsweise gewährleistet, dass Benutzer weiterhin ohne eigenes Zutun auf neue Dokumente hingewiesen werden.

Das Informationsobjekt **Benutzer** repräsentiert im Gegensatz zu den zuvor beschriebenen Objekten nicht die von DAVe primär behandelten Informationen, sondern hat in diesem Zusammenhang eher einen unterstützenden Charakter. Es steht für einen registrierten Anwender des Systems.

Die Benutzer werden in Benutzergruppen organisiert, welche im Berechtigungskonzept des Systems eine wichtige Rolle spielen (siehe ***3.4 Berechtigungskonzept***). Um Zugriff auf DAVe zu erhalten, muss sich ein Anwender deshalb zunächst gegenüber dem System authentifizieren. Alle Aktionen, die anschließend durchgeführt werden, geschehen unter der Verwendung des angegebenen Benutzerobjekts. Tabelle 9 listet alle Eigenschaften eines Benutzers auf.

Eigenschaften	Beschreibung
Vorname	Der Vorname des Benutzers.
Nachname	Der Nachname des Benutzers.
Benutzername	Der Benutzername, der sich aus Vor- und Nachname des Benutzers zusammensetzt.
Passwort	Das Kennwort, mit dem der Benutzer sich in Kombination mit dem Benutzernamen gegenüber dem System authentifiziert.
Gültig bis	Das Datum, bis zu dem der Benutzer Zugriff auf das System hat.
E-Mail	Die E-Mail-Adresse des Benutzers, an welche die Vorgangsbenachrichtigungen gesendet werden.
Department	Das Department, an dem der Benutzer studiert.
Studiengang	Der Studiengang, in dem der Benutzer studiert.
Administrator	Beschreibt, ob der Benutzer Administratorrechte hat.
Benutzergruppe	Die Benutzergruppe, der der Benutzer zugeordnet ist.

Tabelle 9: Eigenschaften des Informationsobjekts „Benutzer"

Abschließend gilt es zu betonen, dass dieser Abschnitt lediglich die Informationsobjekte beschreibt. Das Informationsmodell darf deshalb nicht mit dem Daten- oder Objektmodell der Anwendung verwechselt werden. Es bildet vielmehr die konzeptionelle Grundlage für die Entwicklung letztgenannter.

3.2.2 Anwendungsstruktur

Nach der Entwicklung des Informationsmodells, über welches das in DAVe vorhandene Wissen abgebildet wird, wird nun eine Struktur für die Anwendung benötigt, die dem Benutzer einen einfachen Zugriff auf diese Informationsobjekte gewährt.

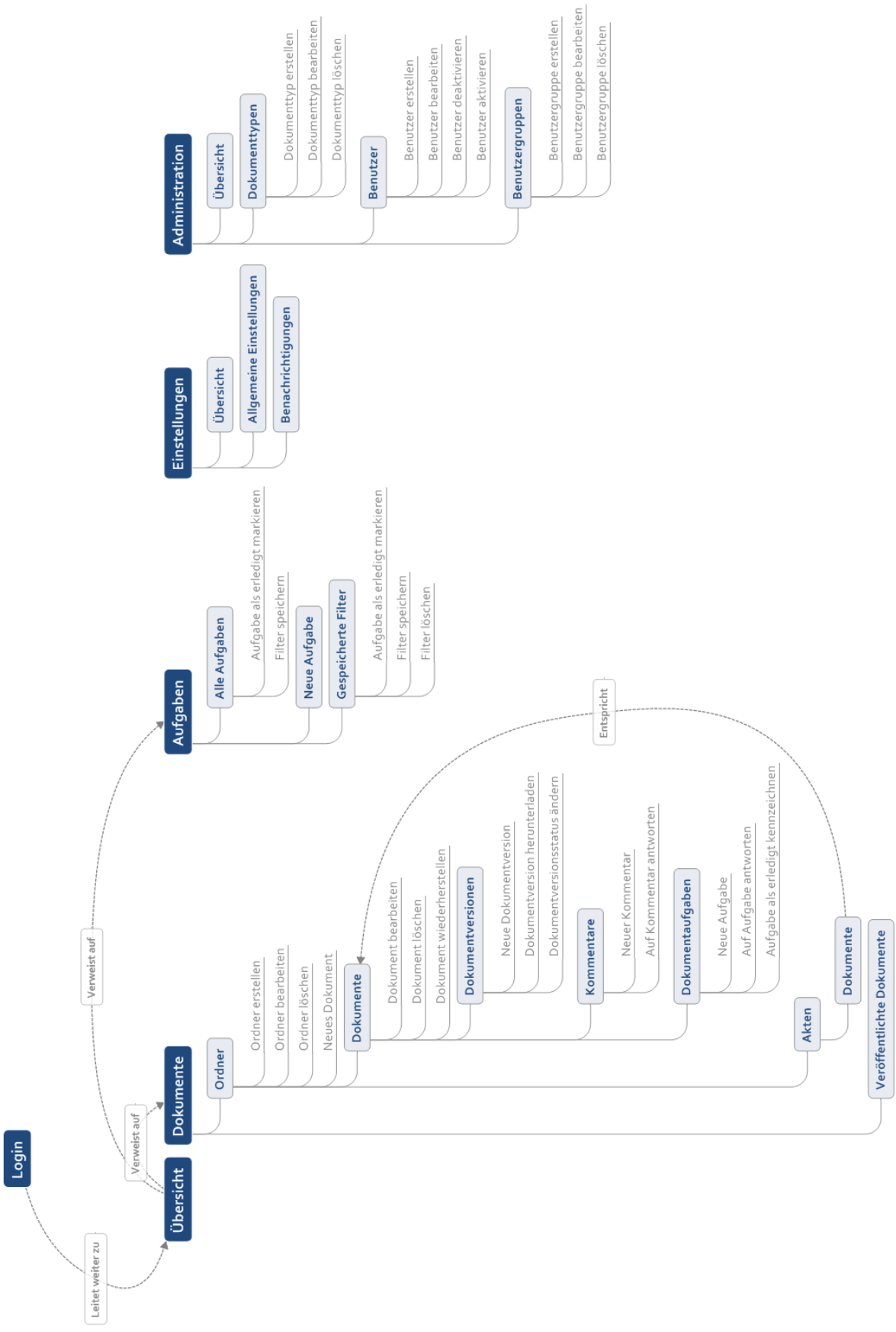


Abbildung 5: Hierarchische Darstellung der Anwendungsstruktur

Wie in Abbildung 5 zu erkennen ist, bildet den Einstiegspunkt nach dem Login eine Übersichtsseite. Sie soll alle aktuellen Vorgänge aufführen, einen direkten Zugriff auf neue Aufgaben und Dokumente bieten und eine Suchoption bereitstellen. Diese Zusammenstellung der Startseite entspricht den Empfehlungen des Usability-Experten Jakob Nielsen, der für die Homepage eine Übersicht über die Hauptbereiche der Seite, eine Zusammenfassung der wichtigsten Neuigkeiten und eine Suchmöglichkeit empfiehlt (vgl. NIELSEN 2001, S. 168).

Von hier aus hat der Benutzer Zugriff auf zwei Ansichten, die die Hauptkomponenten des Systems widerspiegeln: Die Dokumenten- und die Aufgabenansicht.

Eine Informationsarchitektur sollte dem Anwender zwei grundlegende Herangehensweisen für den Information Retrieval gewähren: Eine Suchmaschine, die geschriebene Anfragen verarbeitet, und eine strukturierte Darstellung der Inhalte, die ein Durchsuchen oder Stöbern ermöglicht (vgl. BATLEY 2007, S. 49). Die Dokumenten- und die Aufgabenansicht setzen diese Empfehlung um.

Die Dokumente des Systems erreicht der Benutzer in der Dokumentenansicht über eine hierarchische Navigation, die auf dem in Abschnitt 3.2.1 vorgestellten Ordnerbaum basiert. Darüber hinaus hat er die Möglichkeit, auf die Inhalte von DAVe über eine Volltextsuche zuzugreifen. Neben den Eigenschaften von Ordnern, Dokumenten, Dokumentversionen und Akten werden hierbei auch die Inhalte von Dokumentversionen sowie Kommentare und Aufgaben durchsucht.

Wählt der Benutzer einen Ordner aus, werden die in diesem Ordner befindlichen Dokumente aufgelistet. Bereits auf dieser Ebene wird auch die gegebenenfalls vorhandene Zugehörigkeit eines Dokuments zu einer Akte angezeigt. Ein Verweis führt den Benutzer zu einer ordnerübergreifenden Auflistung aller Dokumente dieser Akte.

Wählt der Anwender ein Dokument aus, erhält er eine Darstellung aller Informationsobjekte und Eigenschaften, die mit diesem Dokument in Verbindung stehen. Dies sind zunächst die für das Dokument vorhandenen Meta-Informationen, wie die Beschreibung und der Dokumenttyp. Anschließend erfolgen eine Auflistung der Dokumentversionen sowie eine Darstellung des Kommentar- und Aufgabenbaums für das Dokument.

Über die hierarchische Navigation in der Dokumentenansicht lässt sich neben den Ordnern noch eine spezielle Ansicht aufrufen, die alle veröffentlichten Dokumentversionen auflistet. So können Benutzer, die für Ordner oder Dokumente

keine Zugriffsrechte besitzen, trotzdem auf die darin enthaltenen öffentlich verfügbaren Dokumentversionen zugreifen.

Ein anderes Bild präsentiert sich dem Benutzer, wenn er in die Aufgabenansicht wechselt. Hier werden zunächst *alle* Aufgaben chronologisch nach Datum der Erzeugung aufgelistet. Eine Eingabemaske ermöglicht es ihm, diese vollständige Liste aller Aufgaben nach verschiedenen Kriterien zu filtern und zu sortieren und sich so benutzerdefinierte Aufgabenlisten ausgeben zu lassen. Die ausgewählten Kriterien können hierbei auch für eine spätere Wiederverwendung gespeichert werden. Der Anwender hat so die Möglichkeit, häufig abgefragte Aufgabenlisten mit einem Klick aufzurufen. Die zur Verfügung stehenden Filterkriterien werden in Kapitel 3.3.3 vorgestellt.

Neben diesen Kernkomponenten von DAVe existiert noch ein weiterer Bereich, in welchem der aktuell angemeldete Benutzer seine persönlichen Einstellungen ändern kann.

Nur für Administratoren sichtbar ist der Administrationsbereich des Systems. Hier lassen sich die Benutzer und Benutzergruppen verwalten sowie neue Dokumenttypen erstellen.

3.2.3 Seitenlayout

Bei der Konzeption des Seitenlayouts wurde darauf geachtet, auf bewährte Gestaltungsformen zurückzugreifen und den Erwartungen der Benutzer zu entsprechen. Das Layout von DAVe basiert deshalb auf dem weit verbreiteten „gekippten L“ (vgl. JACOBSEN 2007, S. 145), wie Abbildung 6 verdeutlicht.

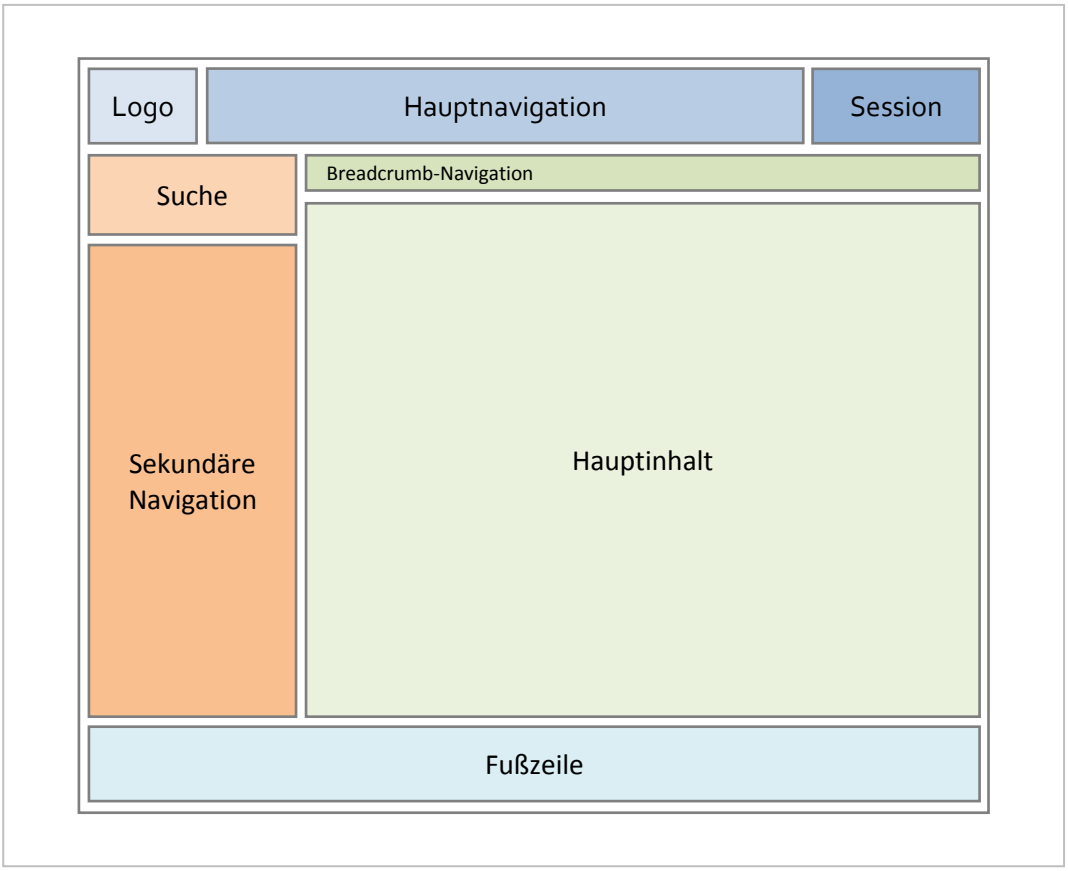


Abbildung 6: Gestaltungsraster für das Seitenlayout

Auch die Anordnung der Seitenelemente entspricht den Erwartungen der Nutzer in Hinblick auf die Positionierung einzelner Komponenten des Layouts (vgl. LYNCH/HORTON 2008, S. 91f.).

Der Header der Seite beginnt links mit dem Logo der Anwendung, welches mit der Übersicht verlinkt ist. So wird im Sinne einer leichten Identifizierung des Systems das Logo prominent platziert und der Benutzer hat bei Orientierungsverlust stets die Möglichkeit, zum Anfang zurückzukehren und von vorne zu beginnen (vgl. JACOBSEN 2007, S. 145).

Rechts neben dem Logo befindet sich die globale Navigation, die Zugriff auf alle Bereiche des Systems bietet sowie daneben Informationen zur aktuellen Sitzung und ein Link zum Abmelden.

Der Bereich unter dem Header gliedert sich in zwei Spalten. Auf der linken Seite, direkt unter dem Logo und somit in einem Bereich, dem der Benutzer viel Aufmerksamkeit schenkt, befindet sich die Suchmöglichkeit (vgl. LYNCH/HORTON 2008, S. 91f.). Darunter folgt die sekundäre Navigation, die je nach ausgewähltem Hauptbereich der Anwendung variiert.

Am oberen Rand der rechten Spalte befindet sich eine Pfadnavigation, die dem Benutzer die Navigation und Orientierung erleichtert. Der restliche und somit größte Teil des Seitenlayouts ist dem eigentlichen Inhalt der Seite gewidmet.

Den Abschluss des Layouts bildet eine Fußzeile, die einen Link zum Impressum und Platz für weitere unterstützende Verweise bietet.

3.3 Funktionsumfang

Nachdem der Aufbau und die Struktur von DAVe bereits ausführlich beschrieben worden sind, gilt es nun, den vollständigen Funktionsumfang der Anwendung zu definieren. Deshalb wird in diesem Kapitel dargestellt, welche Vorgänge mit den bereits vorgestellten Informationsobjekten durchgeführt werden können und welche Funktionen DAVe darüber hinaus bietet.

Die dafür erforderlichen Berechtigungen eines Benutzers für die Durchführung dieser Aktionen werden anschließend in Abschnitt 3.4 geklärt.

Da die Übersicht keine speziellen Funktionalitäten zur Verfügung stellt, sondern lediglich Informationen aus anderen Bereichen der Anwendung aggregiert, beginnt die Beschreibung des Funktionsumfangs in der Dokumentenansicht. Das erste Informationsobjekt, auf das der Anwender hier trifft, ist der Ordner.

3.3.1 Ordner

Ordner bestimmen die Struktur der Dokumentenansicht und organisieren die in DAVe abgelegten Dokumente. Sie lassen sich beliebig verschachteln, so dass die Ordnerhierarchie frei konfigurierbar ist. Für Modifikationen des Ordnerbaums muss der Benutzer Administratorenrechte haben. Reguläre Benutzer können weder Ordner erstellen noch löschen oder modifizieren.

Sowohl beim Erstellen als auch beim Bearbeiten eines Ordners ist lediglich die Angabe eines Namens erforderlich. Zu den weiteren optionalen Feldern gehören Beschreibung, Sortierung und Berechtigungen.

Die Beschreibung ermöglicht es dem erstellenden Benutzer, die anderen Anwender über den Sinn und Zweck eines Ordners genauer zu informieren, damit diese dort möglichst nur Dokumente ablegen, für welche der Ordner auch vorgesehen ist.

Die Sortierung gibt vor, nach welcher Eigenschaft und in welcher Reihenfolge die in dem Ordner vorhandenen Dokumente standardmäßig sortiert sind. Die Eigenschaften, nach denen sortiert werden kann, sind von einem Auswahlfeld vorgegeben und in der folgenden Liste aufgeführt.

- Name
- Erstellt am
- Bearbeitet am
- Akte
- Dokumenttyp

Bei der hier angegebenen Sortierung handelt es sich lediglich um die Standardsortierung, die beim Aufruf des Ordners angewandt wird. So ist es beispielsweise sinnvoll, einen Ordner, der Protokolle enthält, so zu konfigurieren, dass stets die zuletzt erstellten Dokumente an erster Stelle erscheinen. Der Benutzer hat jedoch die Möglichkeit, die Sortierung seinen aktuellen Bedürfnissen anzupassen.

Die Ordnerrechte werden in Abschnitt 3.4 ausführlich behandelt und deshalb hier nicht näher vorgestellt.

Das Löschen eines Ordners ist nur dann möglich, wenn dieser keine Unterordner enthält. Dadurch wird vermieden, dass versehentlich ganze Ordnerbäume gelöscht werden. Enthält ein Ordner noch Dokumente, verhindert dies jedoch nicht die Entfernung des Ordners. In jenem Fall wird der Benutzer im Verlauf des Löschvorgangs aufgefordert, einen neuen Ordner für die vorhandenen Dokumente auszuwählen. Wird ein Ordner gelöscht, gehen demnach in keinem Fall Dokumente verloren.

3.3.2 Dokumente und Dokumentversionen

Die Erstellung eines neuen Dokuments setzt auch die Erzeugung einer initialen Dokumentversion voraus. Dementsprechend werden die Eigenschaften für das Erstellen beider Informationsobjekte gemeinsam abgefragt. Die Einrichtung eines leeren Dokumentcontainers ist somit nicht möglich.

Notwendig für die Erzeugung eines neuen Dokuments ist lediglich die Angabe eines Titels. Darüber hinaus kann der Benutzer noch eine Beschreibung angeben und das Dokument einer Akte und einem Dokumenttyp zuweisen. Der Name der Akte wird als Freitext eingegeben. Sollte bereits eine Akte mit dem eingegebenen Namen existieren, erfolgt eine automatische Zuordnung des Dokuments zu dieser Akte. Andernfalls wird

eine neue Akte mit dem Namen erstellt. Die Dokumenttypen sind vorgegeben und können vom Administrator konfiguriert werden.

Die Pflichtfelder der Dokumentversion umfassen das hochzuladende Schriftstück in Form eines Datei-Uploads sowie Anmerkungen zur Dokumentversion. Letztere sind erforderlich, damit andere Anwender wissen, welchen Inhalt das Schriftstück hat, ohne die Datei herunterladen zu müssen.

Standardmäßig wird die Version als Entwurf gekennzeichnet. Der Benutzer kann die Version alternativ aber auch als freigegeben markieren und in diesem Zuge angeben, ob die Version öffentlich oder intern freigegeben ist. Durch diese Angabe kann später schnell erkannt werden, welche Version dem aktuellen Bearbeitungsstand entspricht und welche der aktuell gültigen Fassung des Schriftstücks.

Ist eine Version als öffentlich gekennzeichnet, kann auf sie auch ohne Zugang zu DAVe zugegriffen werden. Dies ist beispielsweise für Sitzungsprotokolle und Rechtsgrundlagen interessant, die der gesamten Studierendenschaft zugänglich sein sollen.

Wie schon bei den Ordnern lassen sich auch auf Dokumentebene Rechte vergeben. Diese werden ebenfalls in Abschnitt 3.4 vorgestellt.

Bearbeitet ein Benutzer ein schon vorhandenes Dokument, hat er die Möglichkeit, die oben beschrieben Eigenschaften des Dokuments zu bearbeiten: Titel, Beschreibung, Dokumenttyp sowie die Akte und die Berechtigungen. Zusätzlich kann er über die Angabe eines neuen Ordners das Dokument innerhalb des Ordnerbaums verschieben.

Von den Eigenschaften der Dokumentversion lässt sich lediglich die Sichtbarkeit nachträglich bearbeiten. So können freigegebene Versionen auch nach der Erzeugung noch für die Öffentlichkeit zugänglich gemacht oder als intern markiert werden. Der Benutzer hat zwar auch die Möglichkeit, eingestellte Entwürfe als freigegeben zu markieren, dieser Vorgang erzeugt jedoch eine Kopie des Entwurfs und somit ein neues Informationsobjekt – dies ist notwendig, da die bereits vergebene Versionsnummer sonst nicht mit dem neuen Status übereinstimmt.

Einmal erstellte Dokumentversionen lassen sich nicht wieder entfernen, da die Dokumentversionshistorie den Entstehungs- und Bearbeitungsprozess eines Schriftstücks abbilden soll. Lediglich das komplette Dokument kann gelöscht werden, damit einhergehend werden dann auch alle darin enthaltenen Dokumentversionen, Kommentare und Aufgaben aus dem System entfernt.

Das Löschen eines Dokuments ist ein zweistufiger Prozess. Der Löschvorgang bewirkt zunächst lediglich, dass das Dokument systemintern als gelöscht markiert und somit nicht mehr angezeigt wird. Nur für Benutzer mit Administratorrechten ist das Dokument noch sichtbar. Diese haben nun die Möglichkeit, das Dokument wiederherzustellen oder unwiederbringlich mit allen verbundenen Objekten zu löschen. Mit diesem Sicherheitsmechanismus soll das unbeabsichtigte oder böswillige Löschen von Dokumenten verhindert werden.

3.3.3 Kommentare, Aufgaben und Aufgabenfilter

Benutzer können wie in der Vorstellung der Informationsobjekte bereits beschrieben verschiedene Arten von Kommentaren und Aufgaben erzeugen. So ist es möglich, für ein Dokument oder für eine spezielle Dokumentversion einen Kommentar oder eine Aufgabe zu erstellen oder eine allgemeine, dokumentunabhängige Aufgabe anzulegen. Ebenso ist es möglich, auf andere Kommentare zu antworten und auch diese als Aufgabe zu kennzeichnen. Um welche Art von Kommentar oder Aufgabe es sich handelt, hängt vom Kontext ab, aus dem heraus der Kommentar oder die Aufgabe erstellt wurde.

Sowohl Kommentare als auch Aufgaben besitzen beide als einziges Pflichtfeld den Kommentar- bzw. Aufgabentext. Nimmt der Kommentar oder die Aufgabe Bezug auf eine Dokumentversion, kann zusätzlich noch eine Angabe gemacht werden, auf welchen Teil des Schriftstücks genau Bezug genommen wird.

Bei der Erstellung eines Kommentars hat der Benutzer stets die Möglichkeit über das Setzen eines Hakens diesen Kommentar als Aufgabe zu erstellen. Aufgaben haben zusätzlich noch ein optionales Datumsfeld, über welches ein Fälligkeitsdatum für die Aufgabe angegeben werden kann.

Die Darstellung der Kommentare erfolgt in der Dokumentenansicht in Form eines integrierten Kommentar- und Aufgabenbaums. Der Benutzer kann sich aber auch alle Aufgaben in chronologischer Reihenfolge anzeigen lassen.

Die enge Verknüpfung von Kommentaren und Aufgaben in der Dokumentenansicht soll die Benutzer zum Erstellen von Aufgaben motivieren. Kommentiert er ein Dokument oder eine Dokumentversion und bemerkt beim Formulieren seines Kommentars, dass dieser eine Aufgabe darstellt, kann er seinen Kommentar mit einem Klick als Aufgabe kennzeichnen.

Wird eine Aufgabe aus der Aufgabenansicht heraus erzeugt, hat der Anwender die Wahl, ob er die Aufgabe als allgemeine oder dokumentbezogene Aufgabe erstellen will. Ist letzteres der Fall, kann er das entsprechende Dokument oder die entsprechende Dokumentversion über Auswahlfelder definieren.

Weder Kommentare noch Aufgaben können nachträglich bearbeitet oder gelöscht werden. Dies soll verhindern, dass Diskussionen nicht mehr nachvollziehbar sind, wenn Beiträge geändert oder entfernt wurden. Lediglich die Kennzeichnung von erledigten Aufgaben ist möglich. Eine Löschung erfolgt im Falle von Kommentaren und Dokumentaufgaben nur, wenn das zugehörige Dokument gelöscht wird.

Für die Filterung von Aufgaben in der Aufgabenansicht steht eine umfangreiche Auswahl an Kriterien zur Verfügung. Tabelle 10 führt alle zur Verfügung stehenden Filtermöglichkeiten auf.

Eigenschaften	Werte
Aufgabentyp	*Alle Aufgaben*, *Allgemeine Aufgaben*, *Dokumentaufgaben*
Erledigt	*Ja*, *Nein*, *Egal*
Ordner	Alle für den Benutzer sichtbaren Ordner
Unterordner mit einbeziehen[1]	*Ja*, *Nein*
Dokument[2]	Alle Dokumente im ausgewählten Ordner
Sortierung	*Erstellt am*, *Fällig am*, *Erledigt am*, *Dokumenttitel* bzw. *Aufsteigend, Absteigend*
Erstellt am (von/bis)	Datumsangaben
Erstellt von	Datumsangaben
Fällig am (von/bis)	Datumsangaben

Tabelle 10: Filtermöglichkeiten in der Aufgabenansicht

Für das Speichern eines Aufgabenfilters ist die Angabe eines Namens für den Filter notwendig. Administratoren haben zusätzlich die Möglichkeit, den Filter als öffentlich zu kennzeichnen und somit allen Benutzern zur Verfügung zu stellen. So können Filtermöglichkeiten vorgegeben werden, die für alle Benutzer nützlich sind – beispielsweise ein Filter für noch nicht erledigte, überfällige Aufgaben.

Bei einem vorhandenen Aufgabenfilter lassen sich alle Kriterien nachträglich bearbeiten, nur der Name kann nicht geändert werden.

[1] Wird nur angezeigt, wenn ein Ordner und kein Dokument ausgewählt wurden.
[2] Wird nur angezeigt, wenn ein Ordner ausgewählt wurde, der Dokumente enthält.

Jeder Benutzer hat die Möglichkeit, die von ihm angelegten Aufgabenfilter zu löschen. Öffentliche Filter können jedoch nur von Administratoren gelöscht werden, da diese Aktion Auswirkungen auf alle Benutzer hat.

3.3.4 Suche

Eine Suchanfrage kann von jeder Seite aus über das Suchfeld angestoßen werden. Gesucht wird dabei nach Dokumenten, Dokumentversionen, Kommentaren und Aufgaben sowie Akten und Ordnern. Die Umsetzung einer erweiterten Suchfunktion mit der Möglichkeit, nach speziellen Kriterien zu suchen und zu sortieren, ist im Rahmen dieser Arbeit nicht möglich. Dennoch stehen auch für das einfache Suchfeld eine Reihe von Suchoperatoren und -felder zur Verfügung, auf die in Kapitel 4.3.3.3 zusammen mit weiteren Informationen zur Funktionsweise der Suchmaschine näher eingegangen wird.

Die Suchergebnisse werden nach den oben genannten Arten von Informationsobjekten kategorisiert und nach Relevanz absteigend sortiert.

3.3.5 Ereignisprotokoll und Benachrichtigungen

Der Aufbau des Ereignisprotokolls wurde bereits im Rahmen der Informationsarchitektur beleuchtet. In diesem Abschnitt soll nun genauer darauf eingegangen werden, welche Vorgänge protokolliert werden und inwieweit die Anwender die Benachrichtigung über Ereignisse konfigurieren können.

Ein Logeintrag enthält eine Beschreibung und den Zeitpunkt des Vorgangs sowie eine Angabe des Benutzers. Die Beschreibung wird hierbei dynamisch generiert, so dass beispielsweise auch bei einer Änderung des Dokumenttitels alle Logeinträge zu diesem Dokument mit dem aktuellen Dokumenttitel erscheinen und die Benutzer so den Logeintrag besser einem Dokument zuordnen können.

Die Vorgänge, die von DAVe protokolliert werden, und über welche die Benutzer per E-Mail informiert werden, führt Tabelle 11 auf.

Objekt	Aktion	E-Mail-Benachrichtigung
Dokument	Erstellt	Ja
Dokument	Gelöscht	Nein
Dokument	Wiederhergestellt	Nein
Dokument	Bearbeitet (Eigenschaften)	Nein
Dokumentversion	Erstellt	Ja
Dokumentversion	Bearbeitet (Sichtbarkeit)	Nein
Ordner	Erstellt	Nein
Ordner	Gelöscht	Nein
Ordner	Bearbeitet (Eigenschaften)	Nein
Kommentar	Erstellt	Ja
Aufgabe	Erstellt	Ja
Aufgabe	Bearbeitet (Als erledigt markiert)	Ja

Tabelle 11: Vorgänge, die Logeinträge zur Folge haben

Die Angabe zur E-Mail-Benachrichtigung gibt allerdings nur Auskunft darüber, ob die Möglichkeit gegeben ist, sich über diese Ereignisse per E-Mail informieren zu lassen. Ob ein Benutzer über einen Vorgang benachrichtigt wird, hängt von seinen Benutzereinstellungen ab. Dort kann er für die fünf aufgeführten Ereignisse mit Benachrichtigung bestimmen, ob er über diese informiert werden möchte oder nicht.

3.3.6 Benutzereinstellungen

In seinen persönlichen Einstellungen kann der aktuell angemeldete Benutzer seine E-Mail-Adresse sowie sein Kennwort ändern. Die Angabe einer E-Mail-Adresse ist Pflicht.

Zudem hat er die Möglichkeit, seine Benachrichtigungseinstellungen wie in Abschnitt 3.3.5 beschrieben zu konfigurieren.

3.3.7 Administrationsbereich

Der Administrationsbereich ist nur für Benutzer mit der entsprechenden Berechtigung zugänglich. In ihm lassen sich Dokumenttypen, Benutzer und Benutzergruppen verwalten.

Da Dokumenttypen nur dem Zweck dienen, Dokumente nach ihrer Art zu kategorisieren, ist die einzige Eigenschaft eines Dokumenttyps dessen Name. Die Dokumenttypen-

Verwaltung ermöglicht es, neue Dokumenttypen zu erstellen sowie vorhandene umzubenennen und zu löschen.

Gleiches gilt für die Benutzergruppen. Sie stellen Container dar, mit denen sich mehrere Benutzer zu einer Gruppe zusammenfassen lassen. Ihr einziges Merkmal ist ihr Name. Einer Gruppe können anschließend über die Berechtigungseinstellungen von Ordnern und Dokumenten verschiedene Rechte zugewiesen werden.

Die Benutzergruppenverwaltung dient demnach dazu, neue Gruppen zu erstellen sowie vorhandene umzubenennen und zu löschen. Wird eine Benutzergruppe gelöscht, der noch Benutzer zugeordnet sind, so werden diese im Zuge des Löschvorgangs einer anderen Gruppe zugewiesen.

Die umfangreichste Komponente der Administration ist die Benutzerverwaltung. Hier wird der Zugriff auf DAVe verwaltet.

Für die Erstellung eines Benutzers sind eine Reihe von Informationen erforderlich. Um den Benutzer einer Person zuordnen zu können, ist die Angabe von Vor- und Zuname obligatorisch. Daraus wird automatisch der Benutzername generiert. Zusammen mit dem ebenfalls verpflichtenden Kennwort stellten diese die Zugangsdaten dar, die für den Login benötigt werden. Ein Gültigkeitsdatum soll sicherstellen, dass Benutzer keinen zeitlich unbegrenzten Zugriff auf das System haben und einer regelmäßigen Kontrolle ihrer Zugriffslegitimation unterliegen. Die E-Mail-Adresse ist die letzte Angabe, die für die Erzeugung eines neuen Benutzers zwingend benötigt wird.

Zu den weiteren Angaben gehören das Studiendepartment und der Studiengang des Benutzers sowie die für die Rechtevergabe wichtigen Angaben zur Benutzergruppe und Administratorenzugehörigkeit.

All diese Eigenschaften lassen sich nachträglich ändern. Beim Erstellen eines Benutzers ist zusätzlich noch die Option gegeben, diesen über sein neues Benutzerkonto per E-Mail zu benachrichtigen.

Da in jedem Informationsobjekt, das ein Benutzer erzeugt, eine Referenz auf diesen vermerkt wird, ist es nicht möglich, einen Benutzer ohne weiteres zu löschen. Um Personen, die nicht mehr zugriffsberechtigt sind, dennoch die entsprechenden Rechte nehmen zu können, ist es möglich, Benutzer zu deaktivieren.

3.4 Berechtigungskonzept

In DAVe ist ein auf Benutzergruppen basierendes Rechtesystem implementiert, das die Rechtevergabe sowohl auf Ordner- als auch auf Dokumentebene vorsieht. Somit kann bei der Nutzung des Systems durch verschiedene Personengruppen sichergestellt werden, dass die internen Informationen einer Gruppe Anderen nicht zugänglich sind.

Für Ordner können die Rechte „Lesen“ und „Erstellen“ für jede Benutzergruppe vergeben werden. Hat eine Gruppe keine Leserechte, so werden das entsprechende Verzeichnis sowie alle darin enthaltenen Unterverzeichnisse und Dokumente vor den Benutzern dieser Gruppe verborgen. Die Rechte werden also an die untergeordneten Objekte vererbt und können von den diesen nicht überschrieben werden. Das „Erstellen“-Recht bezieht sich auf die Erzeugung neuer Dokumente. Hat ein Benutzer dieses Recht nicht, so besteht nur die Möglichkeit, auf schon vorhandene Dokumente zuzugreifen. Er kann weder neue Dokumente erstellen noch vorhandene löschen.

Werden Ordnerrechte nachträglich geändert, erfolgt eine automatische Aktualisierung der Rechte der untergeordneten Objekte.

Auf Dokumentebene lassen sich die Rechte „Lesen“, „Bearbeiten“ und „Löschen“ vergeben, wobei jedes Recht jeweils die in obiger Reihenfolge genannten vorherigen voraussetzt. Wird das Bearbeitungsrecht vergeben, so muss auch das Leserecht gegeben sein. Das Recht zu Löschen setzt sowohl das Recht zu Lesen als auch das zu Löschen voraus.

Wird einer Benutzergruppe das Recht, ein Dokument zu lesen entzogen, so ist dieses Dokument für die Benutzer der Gruppe nicht mehr sichtbar. Erhält die Gruppe nur das Recht zu lesen, hat sie lesenden Zugriff auf die Dokumenteigenschaften, die Dokumentversionen sowie auf den Kommentar- und Aufgabenbaum. Die Leserechte erlauben auch die Erstellung von neuen Kommentaren und Aufgaben sowie die Kennzeichnung von Aufgaben als erledigt. Neue Dokumentversionen sowie die Bearbeitung der Dokumenteigenschaften sind nicht möglich. Hierfür ist das Bearbeitungsrecht erforderlich. Um das Dokument auch löschen zu dürfen, muss zusätzlich das Löschrecht erteilt worden sein.

Die Rechte des übergeordneten Ordners wirken sich hierbei auf die Rechtevergabe der darin enthaltenen Dokumente aus. Besitzt eine Benutzergruppe für einen Ordner keine Leserechte, so ist es nicht möglich, dieser Benutzergruppe für ein in dem Ordner enthaltenes Dokument Leserechte zu erteilen. Wird ein Dokument in einen anderen

Ordner verschoben, werden deshalb die Rechte des Dokuments automatisch an die Rechte des neuen Ordners angepasst.

Die Erteilung von Administratorrechten an einen Benutzer setzt die durch die Benutzergruppe zugewiesenen Rechte teilweise außer Kraft, da Administratoren, um ihren Aufgaben nachkommen zu können, erweiterte Privilegien benötigen.

Bei Ordnern betrifft dies konkret die Leserechte. Um Modifikationen am Ordnerbaum vornehmen können, ist es erforderlich, dass sie diesen auch komplett einsehen können. Die Administratorrechte überschreiben deshalb die von der Benutzergruppe definierten Leserechte und machen so alle Ordner für den Benutzer sichtbar. Dadurch wird vermieden, dass in jeder Benutzergruppe ein Administrator vorhanden sein muss.

Auf Dokumentebene werden die Administratorrechte differenzierter umgesetzt. So sind für den Administrator alle Dokumente sichtbar, jedoch werden sie in einer Form dargestellt, die keine Rückschlüsse auf deren Inhalt zulässt. Titel, Beschreibung, Anmerkungen zu Dokumentversionen sowie die Inhalte von Kommentaren und Aufgaben sind demnach nicht einsehbar. Die Bearbeitungsrechte beschränken sich auf die Konfiguration der Berechtigungen für das Dokument, auf die übrigen Eigenschaften kann nicht zugegriffen werden. Die Löschrechte stehen administrativen Benutzern ohne Einschränkungen zur Verfügung.

Das Berechtigungssystem wirkt sich auch auf Logeinträge aus. Mit jedem Logeintrag werden Informationen über die Sichtbarkeit des Eintrags für die einzelnen Benutzergruppen gespeichert. Ändern sich die Berechtigungen für ein Dokument werden rückwirkend auch die Sichtbarkeitseinstellungen aller Logeinträge für dieses Dokument entsprechend angepasst.

4 Technische Umsetzung

4.1 Verwendete Frameworks

4.1.1 Symfony

Bevor auf die Frage eingegangen wird, warum für die Entwicklung von DAVe Symfony als zugrundeliegendes Framework ausgewählt wurde, ist es sinnvoll, zunächst auf die generellen Vorteile der Verwendung eines Web-Frameworks einzugehen.

Ein Framework dient als Rahmen für ein Softwareprojekt und nimmt dem Entwickler sich stets wiederholende Tätigkeiten durch die Bereitstellung von vorgefertigten Bausteinen ab, unterstützt ihn dabei, sauberen, gut strukturierten Quellcode zu produzieren und fördert zugleich die Wiederverwendbarkeit von Code. Die offizielle Dokumentation des Symfony-Frameworks schreibt hierzu treffend: „[…] there is no need to reinvent the wheel every time a new web application is built!“, (ZANINOTTO/POTENCIER 2009A, S. 10).

Symfony ist ein auf PHP 5 basierendes Web-Framework, welches von der französischen Firma Sensio Labs entwickelt und als Open-Source bereitgestellt wird. Es vereint in sich die oben genannten Vorteile eines Frameworks und bietet darüber hinaus Unterstützung für viele bewährte Vorgehensweisen und Entwurfsmuster. Hierzu gehören die strikte Trennung von Präsentations-, Steuerungs- und Modellschicht nach dem MVC-Prinzip, die Abstraktion des Datenbankzugriffs und ein objektrelationales Mapping (ORM), welches die in der Datenbank abgelegten Daten in Form von Objekten bereitstellt (ZANINOTTO/POTENCIER 2009A, S. 10ff.).

Darüber hinaus sprechen noch einige weitere Gründe für den Einsatz von Symfony bei der Umsetzung dieser Arbeit.

Die Erzeugung und Bearbeitung fast aller Informationsobjekte des Systems erfolgt mithilfe von Formularen. Sie stellen somit einen zentralen Bestandteil der Benutzerschnittstelle dar. Symfony bietet hier auf Basis des vorgegebenen Datenmodells

vorgefertigte Formularklassen, die sich mit wenig Quellcode an die Bedürfnisse der Anwendung anpassen lassen und somit viel Programmieraufwand vermeiden. Eingebaute Validatoren und automatisches Output-Escaping sorgen für die Fehlerbehandlung und -vermeidung und schützen vor eingeschleustem Schadcode (vgl. ZANINOTTO/POTENCIER 2009C).

Dank des objektrelationalen Mappings kann die Umsetzung des Berechtigungskonzepts direkt an der Schnittstelle zur Datenbank erfolgen. Dieses Konzept wird in Kapitel 4.3.2 näher erläutert. Unterstützt wird das Berechtigungskonzept zudem durch die in Symfony integrierten Authentifizierungs- und Sicherheitsmechanismen, die eine einfache Umsetzung der Zugriffskontrolle ermöglichen.

Nicht zuletzt sprechen die gute Dokumentation des Frameworks sowie eine sehr aktive Community für den Einsatz von Symfony in diesem Projekt (vgl. FRANZ 2008).

4.1.2 jQuery

Auf der Clientseite verwendet DAVe die JavaScript-Bibliothek jQuery für alle dynamischen Funktionalitäten der Anwendung. Die Wahl von jQuery geht darauf zurück, dass Symfony die Bibliothek bereits für einige interaktive Formularelemente verwendet und das Framework somit ohnehin Bestandteil der Anwendung ist. Eine zusätzliche JavaScript-Bibliothek zu laden würde somit vor allem unnötigen Ballast für das System darstellen.

Darüber hinaus sprechen aber noch weitere Gründe für die Verwendung von jQuery. Die Kompatibilität mit allen aktuellen Browsern vermeidet weitestgehend den Einsatz von selbstentwickelten Browserweichen und somit zusätzlichen Programmieraufwand. Zudem stellt jQuery umfangreiche Methoden zur DOM-Modifikation und -Traversierung bereit, die dem Entwickler dabei helfen, lesbaren und schlanken Quellcode zu entwickeln (vgl. HEILMANN 2006, S. 419ff.).

Ein besonderes Merkmal von jQuery ist die Unterstützung von „Unobtrusive DOM Scripting“, einer Technik, bei der die durch JavaScript bereitgestellte Funktionalität von der eigentlichen Seitenstruktur und somit von der Präsentationsschicht getrennt wird. Dabei soll gewährleistet bleiben, dass eine Seite auch dann noch verwendet werden kann, wenn JavaScript-Funktionalitäten nicht zur Verfügung stehen (vgl. HOLT 2008).

jQuery stellt darüber hinaus Methoden zur Verfügung, die JavaScript um fehlende oder bisher umständlich implementierte Funktionalitäten erweitert. Hierzu gehören

Funktionen für die Umsetzung von Effekten, Animationen und CSS-Manipulationen sowie eine einfach zu verwendende AJAX-Umsetzung.

4.2 Datenmodell

Mit Ausnahme der hochgeladenen Dokument-Dateien werden alle in DAVe hinterlegten Informationen in einer MySQL-Datenbank gespeichert. Der Zugriff findet über eine Reihe von Klassen statt, die Symfony mithilfe des ORM-Frameworks Propel automatisch erzeugt.

Dieses Kapitel beschreibt zunächst das für DAVe erstellte Datenbankschema mit den darin enthaltenen Tabellen. Anschließend wird das von Propel verwendete Konzept zur Abbildung der relationalen Datenbankdaten auf PHP-Objekte erläutert.

4.2.1 Datenbankschema

Die Abbildung 7 stellt das von DAVe verwendete Datenbankschema dar.

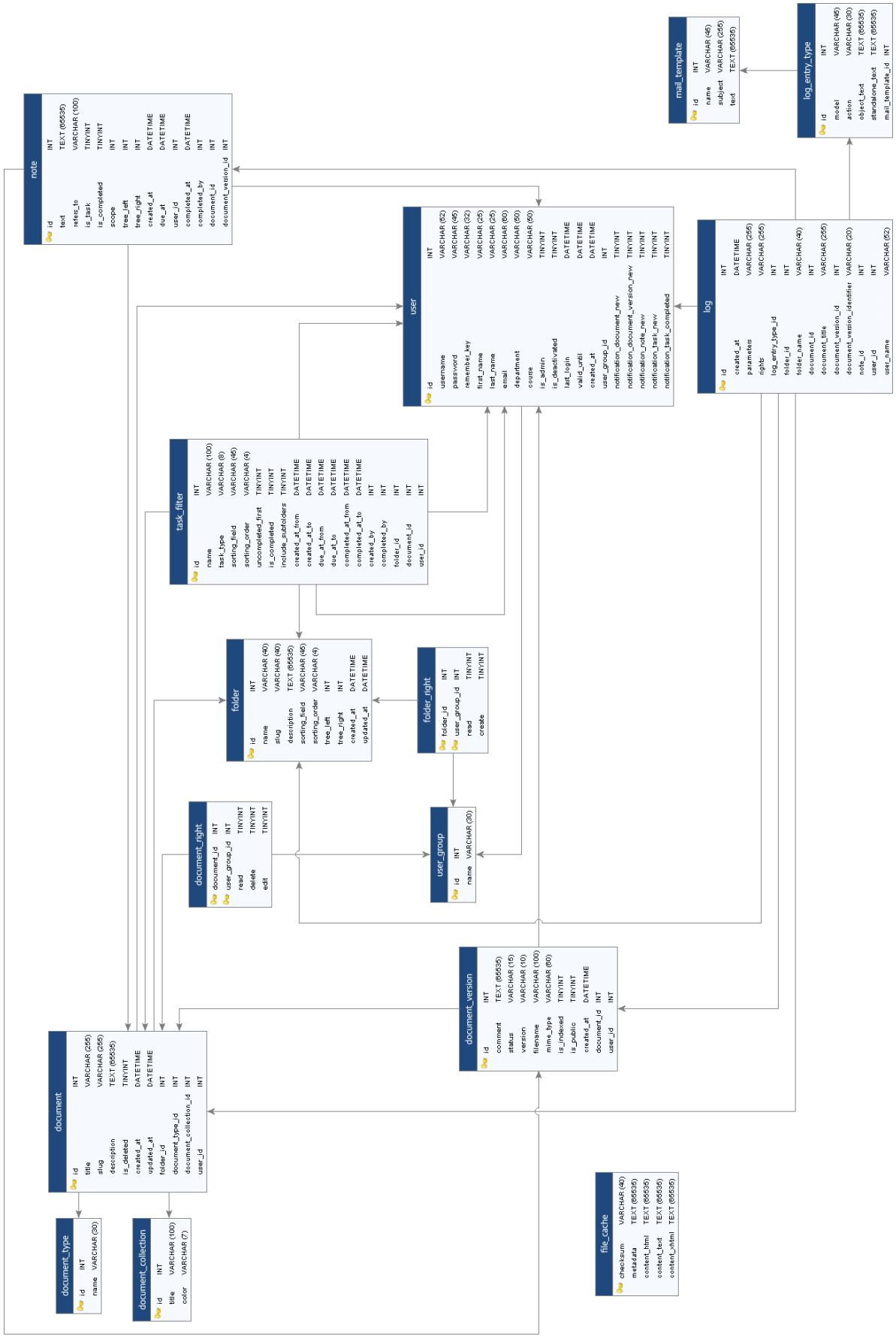


Abbildung 7: Übersicht über das Datenbankschema

Das Diagramm zeigt die einzelnen Tabellen des Schemas und deren Felder. Die Primärschlüssel sind jeweils mit einem Schlüsselsymbol gekennzeichnet. Die dargestellten Beziehungen sind ausschließlich 1-zu-n-Beziehungen.

Im Folgenden werden diese Tabellen und ihre Funktion innerhalb der Anwendung kurz vorgestellt. Da die meisten Feldnamen selbsterklärend sind, wird in diesem Kapitel auf eine vollständige Beschreibung aller Tabellenfelder verzichtet. Stattdessen soll auf ***Anhang B Datenbankschema*** verwiesen werden, welcher eine vollständige, mit Anmerkungen versehene Liste aller Tabellen und Felder enthält.

Für die bereits vorgestellten Informationsobjekte Dokument, Dokumentversion, Ordner, Akte, Logeintrag, Benutzer und Benutzergruppe existiert jeweils eine Tabelle, die die Meta-Informationen des Objekts beherbergt. Kommentare und Aufgaben werden aufgrund ihrer ähnlichen Datenstruktur und der einfacheren Darstellung in einem gemeinsamen Kommentar- und Aufgabenbaum in *einer* Tabelle abgelegt. Die Unterscheidung erfolgt hier über das Feld `is_task`.

Tabelle 12 listet die Informationsobjekte und die zugehörigen Datenbanktabellen auf. Die hier verwendeten englischen Bezeichnungen werden auch im Programmcode durchgängig verwendet. Eine vollständige Liste der internen Bezeichnungen befindet sich in Anhang A.

Informationsobjekt	Datenbanktabelle
Akte	`document_collection`
Benutzer	`user`
Benutzergruppe	`user_group`
Dokument	`document`
Dokumentversion	`document_version`
Kommentar, Aufgabe	`note`
Logeintrag	`log`
Ordner	`folder`

Tabelle 12: Informationsobjekte und die zugehörige Datenbanktabelle

Für die Abbildung der hierarchischen Anordnung von Ordnern und Kommentaren bzw. Aufgaben in der relationalen Datenbank wird das Nested-Set-Modell verwendet, das die Abfrage der kompletten Hierarchie mit einem SQL-Statement ohne Rekursion erlaubt (s. KLEMPERT 2003). Eine Umsetzung dieses Modells ist in Propel implementiert. Methoden

zur Ermittlung von Teilbäumen und zur Durchführung von Modifikationen an der Baumstruktur werden somit in den erzeugten Klassen bereitgestellt (s. PROPEL).

Neben der `log`-Tabelle sind für das Ereignisprotokoll und die damit zusammenhängende E-Mail-Benachrichtigung noch zwei weitere Tabellen relevant. In der Tabelle `log_entry_type` befinden sich die verfügbaren Arten von Logeinträgen und die Tabelle `mail_templates` enthält E-Mail-Vorlagen, die für die Benachrichtigung der Benutzer verwendet werden. Das genaue Zusammenspiel dieser Tabellen und die allgemeine Funktionsweise des Ereignisprotokolls wird in Kapitel 4.3.4 näher erläutert.

Die Zugriffsrechte für Ordner und Dokumente werden in den Tabellen `folder_rights` und `document_rights` abgelegt. In welcher Form dies geschieht und wie die Kontrolle der Berechtigungen stattfindet, beschreibt Kapitel 4.3.2.

Die im Aufgabenbereich erstellten Aufgabenfilter sind in der Tabelle `task_filter` gespeichert. Dabei werden lediglich die Suchkritierien in der Datenbank abgelegt. Die Ergebnisliste wird bei jedem Aufruf neu generiert und ist somit stets auf dem aktuellen Stand.

Die einzige Tabelle in diesem Schema, die keine Referenz auf andere Tabellen aufweist, ist `file_cache`. In ihr werden die Ergebnisse der Inhaltsextraktion abgelegt, die im Rahmen der Suchindizierung gewonnen werden. Dieser Vorgang wird in Kapitel 4.3.3.2 dargestellt.

4.2.2 Objektrelationales Mapping

Dieser Abschnitt beschreibt das objektrelationale Mapping, das für die Interaktion mit der Datenbank verwendet wird.

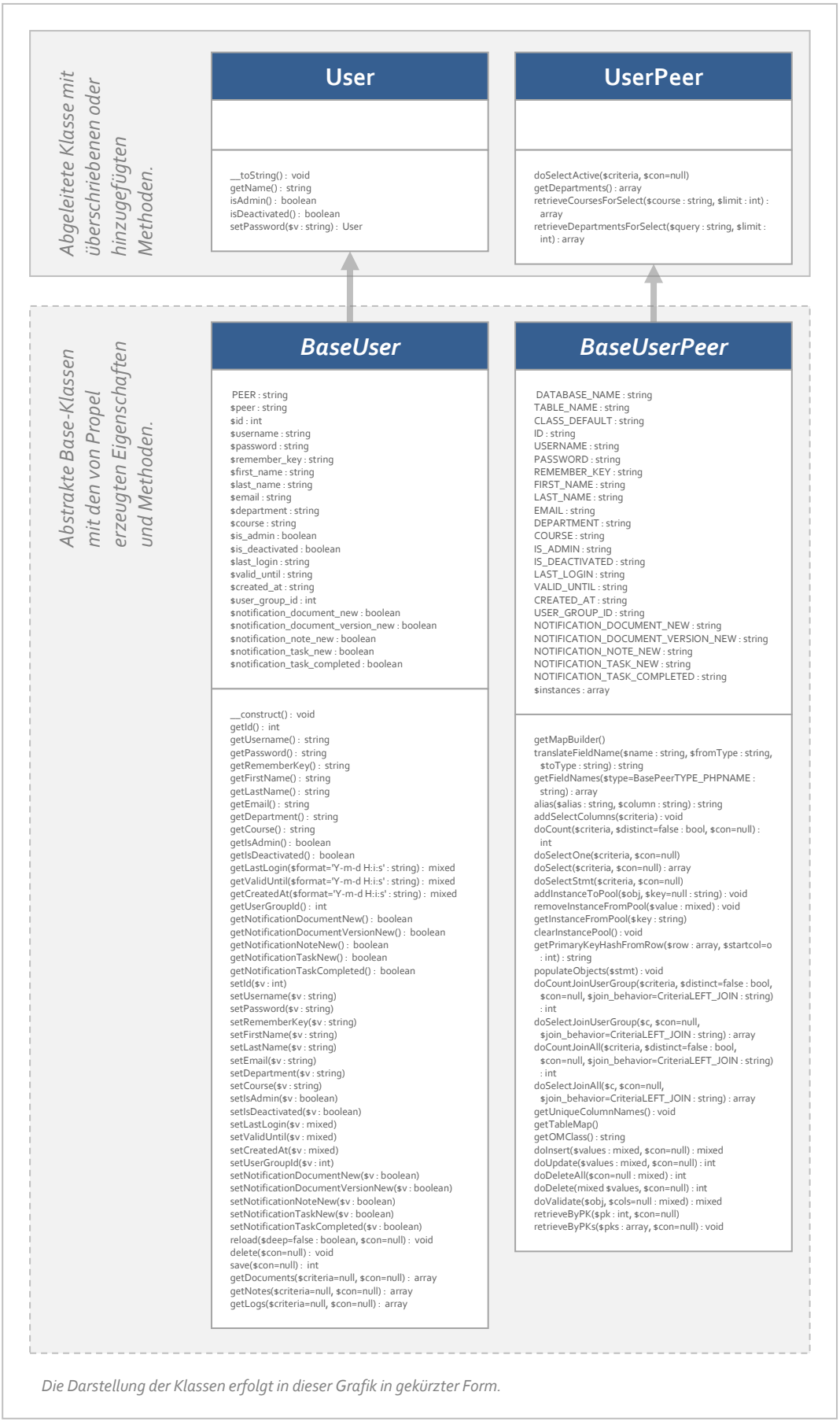
Symfony bietet für das objektrelationale Mapping zwei verschiedene Implementierungen: Propel und Doctrine (vgl. ZANINOTTO/POTENCIER 2009A, S. 15). Im Rahmen von DAVe ist die Wahl auf Propel gefallen, da dies für die verwendete Symfony-Version 1.2 noch die Standardeinstellung darstellt und diese Konfiguration somit weiter verbreitet und besser dokumentiert ist.[3]

[3] Ab Version 1.3 wird Symfony mit Doctrine als Standard-ORM ausgeliefert, da Doctrine komplexere Abfragen erlaubt und die Weiterentwicklung von Propel zunehmend ins Stocken gerät.

Propel repräsentiert Datenbankinhalte in Form von PHP-Objekten und ermöglicht es, diese wiederum in der Datenbank abzulegen. Hierfür wird das Datenbankschema in einer speziellen Konfigurationsdatei angegeben. Aus dieser generiert Propel die entsprechenden Datenbanktabellen sowie die zugehörigen PHP-Klassen (vgl. ebd., S. 128).

Datenbankabfragen werden dabei komplett von Propel abstrahiert und ermöglichen so theoretisch eine Unabhängigkeit vom darunterliegenden Datenbanksystem. Der Programmierer muss zur Abfrage von Datensätzen keine SQL-Statements formulieren, sondern nutzt die von Propel bereitgestellten Methoden (s. ebd., S. 129ff.).

Die Umsetzung des Mappings soll im Folgenden anhand der Tabelle `user` erläutert werden. Für jede Tabelle generiert Propel insgesamt vier Klassen: Zwei abstrakte Base-Klassen und zwei Klassen, die von diesen Klassen ableiten. Im Falle der `user`-Tabelle sind dies `BaseUser`, `BaseUserPeer`, `User` und `UserPeer`. Abbildung 8 veranschaulicht die Beziehung dieser Klassen noch einmal.



Die Darstellung der Klassen erfolgt in dieser Grafik in gekürzter Form.

Abbildung 8: Die von Propel erzeugten Klassen für die user-Tabelle

Die von Propel erzeugten Methoden werden in den abstrakten Base-Klassen definiert und stehen somit auch in den davon abgeleiteten Klassen, die später für den Datenbankzugriff verwendet werden, zur Verfügung. Abgesehen von der Klassendefinition sind diese abgeleiteten Klassen ansonsten leer und stellen zunächst eine reine Kopie der Elternklasse dar. In ihnen können aber die von Propel generierten Methoden überschrieben und erweitert werden. Da Propel im Falle einer Änderung des Datenmodells nur die Base-Klassen neu generiert, bleiben die eigenen Modifikationen unberührt.

Diese zweiteilige Klassenstruktur ermöglicht somit ein Objektmodell, das an die eigenen Bedürfnisse angepasst werden kann, ohne dass der Vorteil der automatisierten Erzeugung und Modifikation verloren geht (vgl. ebd., S. 130).

Die `User`-Klasse ist eine Objekt-Klasse, deren Instanzen einen Datensatz der `user`-Tabelle repräsentiert. Für jedes Datenbankfeld der Tabelle generiert Propel eine Getter- und eine Setter-Methode, über die sich die Eigenschaften des Datensatzes bzw. des Objekts lesen und bearbeiten lassen.

Die `UserPeer`-Klasse hingegen ist statisch und stellt Methoden zur Verfügung, mit denen sich Abfragen auf die `user`-Tabelle durchführen lassen – beispielsweise die Abfrage von Datensätzen. Die ermittelten Tabelleneinträge werden dann in Form von Objekten der `User`-Klasse zurückgegeben.

Listing 1 verdeutlicht das Zusammenspiel der beiden Klassen.

```
198 $user = UserPeer::retrieveByPK(1);
199
200 // Gibt den in der Datenbank gespeicherten Namen aus
201 echo $user->getFirstName();
202
203 $user->setFirstName('David');
204
205 // Gibt 'David' aus
206 echo $user->getFirstName();
207
208 $user->save();
```

Listing 1: Beispiel für das Abfragen und Bearbeiten eines Benutzers

In dem Beispiel wird zunächst über die von der `UserPeer`-Klasse bereitgestellte Methode `retrieveByPK()` der Datensatz mit der ID 1 ermittelt und in Variablen `$user` gespeichert (Zeile 1). Anschließend wird der in der Datenbank gespeicherte Name mithilfe der von Propel generierten Getter-Methode für das Datenbankfeld `first_name` ausgegeben (Zeile 4). Die zugehörige Setter-Methode wird in Zeile 6 verwendet, um den

Namen in „David“ zu ändern, das Resultat wird in Zeile 9 geprüft. Diese Änderung existiert bis zu diesem Zeitpunkt nur in der in `$user` gespeicherten Instanz, nicht jedoch in der Datenbank. Erst die in Zeile 11 ausgeführte `save()`-Methode des `User`-Objekts aktualisiert den entsprechenden Datensatz in der Datenbank.

Ähnlich einfach ist die Erzeugung eines neuen Datensatzes in der `user`-Tabelle. Statt der Abfrage eines bestehenden Benutzers mithilfe der `UserPeer`-Klasse, wird einfach ein neues `User`-Objekt erzeugt.

```
1  $user = new User();
2
3  $user->setUsername('david.bowman');
4  $user->setFirstName('David');
5  $user->setLastName('Bowman');
6  $user->setEmail('david.bowman@haw-hamburg.de');
7  $user->setIsAdmin(false);
8  $user->validUntil('2010-12-31 00:00:00');
9  $user->setUserGroupId(1);
10
11 $user->save();
```

Listing 2: Beispiel für das Anlegen eines neuen Benutzers

In Listing 2 wird mit `new User()` zunächst ein leeres `User`-Objekt erstellt (Zeile 1). Die Eigenschaften des Benutzers werden über die entsprechenden Setter-Methoden definiert (Zeile 3-9). Anschließend wird der neue Benutzer mit der `save()`-Methode in der Datenbank gespeichert (Zeile 11).

Da in der Datenbank der Vor- und Zuname der Benutzer in zwei separaten Feldern gespeichert wird, erzeugt Propel folgerichtig für jedes Feld jeweils eine Getter-Methode. In den meisten Fällen soll jedoch der Vor- und Zuname gemeinsam ausgegeben werden. Hier kommt nun die oben bereits erläuterte leere `User`-Klasse ins Spiel. In ihr kann eine neue Methode `getName()` definiert werden, die die Aufrufe von `getFirstName()` und `getLastName()` verkettet und deren Ergebnisse gemeinsam zurückliefert. Listing 3 zeigt das entsprechende Beispiel aus der Objektschicht von DAVe.

```
/lib/model/User.php
10 class User extends BaseUser
11 {
23     public function getName()
24     {
25           return $this->getFirstName() . ' ' . $this->getLastName();
26     }
68 }
```

Listing 3: Die Objektklasse der user-Tabelle wird um die Methode getName() erweitert

Auch Referenzen auf Einträge in anderen Datenbanktabellen können über die von Propel definierten Getter- und Setter-Methoden behandelt werden. Ein Aufruf von `$user->getUserGroup()` liefert ein entsprechendes Objekt der Klasse `UserGroup` zurück. Genauso lässt sich die Benutzergruppe eines Benutzers über `$user->setUserGroup($userGroup)` ändern, wobei `$userGroup` in diesem Beispiel ein Objekt der Klasse `UserGroup` darstellt und somit einen Eintrag der Tabelle `user_group`.

Auf der beiliegenden CD befindet sich eine umfassende Dokumentation, in der alle für DAVe entwickelten Klassen beschrieben werden[4]. Die von Propel generierten Base-Klassen sind dort im Subpackage *model-om* zu finden, die abgeleiteten Klassen mit ihren modifizierten und ergänzten Methoden im Subpackage *model*.

Ebenfalls von Symfony generiert sind die Klassen in den Subpackages *model-map* und *form*. Letztere repräsentieren die in DAVe verwendeten Formulare, welche analog zum Propel-ORM anhand des Datenschemas in Base-Klassen vordefiniert werden und sich in davon abgeleiteten Klassen modifizieren lassen (s. ZANINOTTO/POTENCIER 2009C, S. 55ff.).

Eine detaillierte Besprechung der Modell- und Formularklassen ist im Rahmen dieser Arbeit nicht möglich. Stattdessen sei an dieser Stelle auf die Symfony-Dokumentation verwiesen, die sowohl die Modell-Schicht (s. ZANINOTTO/POTENCIER 2009A, S. 126ff.) als auch das Formularsystem (s. ZANINOTTO/POTENCIER 2009C) ausführlich vorstellt.

4.3 Programmsteuerung

4.3.1 Programmsteuerung in Symfony

Der Aufbau der Steuerungsschicht wird von Symfony weitestgehend vordefiniert. Eine Symfony-Anwendung gliedert sich dabei in einzelne Module, die wiederum aus einer oder mehrerer Actions bestehen (vgl. ZANINOTTO/POTENCIER 2009A, S. 29). In DAVe existieren unter anderem die Module `overview`, `document`, `task`, `user` und `administration`, welche für die einzelnen Ansichten, die in der Anwendungsstruktur vorgestellt wurden, stehen. Die darin enthaltenen Actions sind für die möglichen Vorgänge innerhalb eines Moduls verantwortlich.

[4] Die Dokumentation wurde automatisiert mithilfe von PHPDOC-Kommentaren im Quellcode erstellt. Da die von Propel bzw. Symfony generierten Klassen englischsprachige Kommentare aufweisen, wurden auch alle speziell für DAVe entwickelten Klassen im Sinne einer einheitlichen Dokumentation in englischer Sprache dokumentiert.

Alle Webanfragen werden in Symfony dabei von dem so genannten Front Controller verarbeitet. Dieser ermittelt anhand des URLs, an welche Action in welchem Modul er die Anfrage weiterreichen muss (s. ebd., S. 74f.). Ruft der Anwender also beispielsweise die Seite zum Erstellen eines Benutzers auf, erkennt Symfony, dass es diese Anfrage an die Action `userNew` in dem Modul `administration` weiterleiten muss.[5]

Die Verknüpfung eines URLs mit einer Action nimmt der Entwickler in der Routing-Konfiguration von Symfony vor (s. ebd., S. 152ff.). Er kann somit selbst über den Aufbau und die Formulierung der URLs bestimmen.

Dieser hohe Standardisierungsgrad innerhalb der Controller-Schicht ermöglicht systemfremden Entwicklern eine leichte Einarbeitung in unbekannte Anwendungen, sofern sie mit der Funktionsweise von Symfony vertraut sind.

Da es der Umfang dieser Arbeit nicht erlaubt, die komplette Funktionsweise von Symfony oder DAVe vorzustellen, werden in den folgenden Abschnitten die Komponenten des Systems näher betrachtet, die in ihrer Umsetzung besonders interessant sind und über reine CRUD-Vorgänge[6] hinausgehen. Einen Überblick über die Module und Actions der Steuerungsschicht sowie eine vollständige Beschreibung der in den folgenden Kapiteln erwähnten Klassen bietet die Dokumentation auf der beiliegenden CD. Für die Module existiert hier jeweils ein Subpackage, welches die Klasse mit den entsprechenden Actions enthält. In Anhang C befindet sich zudem eine mit Anmerkungen versehene Darstellung der Verzeichnisstruktur dieser Anwendung (s.a. ebd., S. 29ff.).

4.3.2 Rechteverwaltung

4.3.2.1 Speicherung der Berechtigungen in der Datenbank

Wie bereits in der Vorstellung des Datenbankschemas erwähnt, existieren für die Order- und Dokumentrechte zwei separate Tabellen: `folder_rights` und `document_rights`. Für jede Benutzergruppe wird hier pro Ordner beziehungsweise Dokument ein Eintrag erstellt, der die Rechte für das jeweilige Objekt festlegt. Fehlt für einen Ordner oder ein Dokument der Eintrag für eine Benutzergruppe, so hat diese keine Zugriffsrechte auf das entsprechende Objekt. Dies ist beispielsweise bei der Erzeugung einer neuen

[5] Die entsprechenden Klassen befinden sich in /apps/frontend/modules/.
[6] CRUD steht für Create, Read, Update und Delete und umschreibt damit die grundlegenden Datenbankoperationen für einen Datensatz.

Benutzergruppe der Fall, da für die bereits vorhandenen Ordner und Dokumente zunächst keine Einträge für die neue Gruppe in den Rechtetabellen erstellt werden. Erst wenn ein Ordner oder ein Dokument bearbeitet wird, erhält die Benutzergruppe einen Datensatz in der entsprechenden Rechtetabelle.

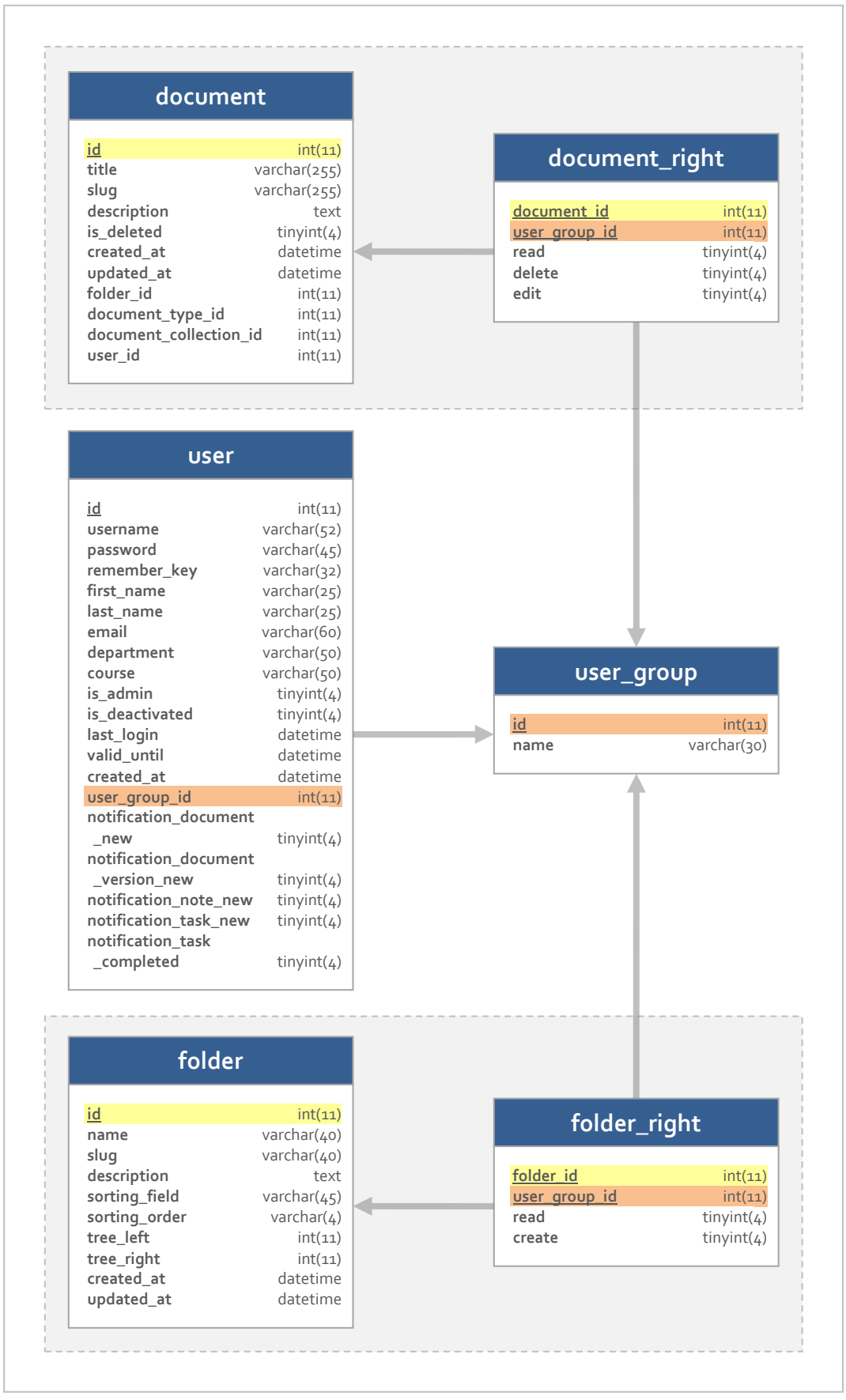


Abbildung 9: Schematische Darstellung der Rechtetabellen und ihrer Beziehungen

Die Rechtetabellen bestehen, wie in Abbildung 9 zu sehen, aus vier beziehungsweise fünf Feldern. Das erste Feld verweist auf den Ordner beziehungsweise das Dokument, für das die Rechte vergeben werden sollen. Der zweite Eintrag enthält die Referenz auf die Benutzergruppe, die die entsprechenden Rechte erhält. Die übrigen zwei beziehungsweise drei Felder stehen für die einzelnen Rechte, die erteilt werden können. Bei den Dokumentrechten sind dies demnach `read`, `delete` und `edit`, bei den Ordnerrechten `read` und `create`. Sie enthalten boolesche Werte in Form von `0` und `1`. Weist beispielsweise das Feld `read` den Wert `1` auf, also `true`, so besitzt die angegebene Benutzergruppe für das entsprechende Objekt Leserechte.

4.3.2.2 Überprüfung der Leseberechtigung in der Modellschicht

Die Überprüfung von Zugriffsrechten erfolgt in vielen Fällen bereits in der Modellschicht. Dadurch bleibt der Quellcode der Steuerungsschicht schlank und es stehen nur die Objekte zur Verfügung, auf die der Benutzer tatsächlich lesenden Zugriff hat. Hierfür wurden die Methoden zur Abfrage von Datensätzen in den Peer-Klassen überschrieben und angepasst.

```
/lib/model/DocumentPeer.php
10  class DocumentPeer extends BaseDocumentPeer
11  {
65     public static function doSelect(Criteria $criteria, PropelPDO $con =
    null)
66     {
67        if(!sfContext::getInstance()->getUser()->isAdmin()) {
68            $criteria->addJoin(DocumentPeer::ID,
    DocumentRightPeer::DOCUMENT_ID);
69            $criteria->add(DocumentRightPeer::USER_GROUP_ID,
    sfContext::getInstance()->getUser()->getUserGroupId(), Criteria::EQUAL);
70            $criteria->add(DocumentRightPeer::READ, 1, Criteria::EQUAL);
71            $criteria->add(DocumentPeer::IS_DELETED, 0, Criteria::EQUAL);
72        }
73
74        return parent::doSelect($criteria, $con);
75     }
188 }
```

Listing 4: Die von Propel in der Base-Klasse bereitgestellte doSelect()-Methode wird überschrieben

Listing 4 zeigt ein entsprechendes Beispiel aus der `DocumentPeer`-Klasse. Die Methode `doSelect()` dient zur Abfrage von Datensätzen unter Berücksichtigung der über den ersten Parameter übergebenen Propel-Kriterien (s. ZANINOTTO/POTENCIER 2009A, S. 134ff.). Sofern der aktuelle Benutzer kein Administrator ist (Zeile 67), ergänzt diese Methode das `Criteria`-Objekt automatisch um Kriterien, die die Abfrage auf Datensätze einschränken, für die der aktuelle Benutzer eine Leseberechtigung besitzt (Zeile 68-71). Hierfür wird zunächst die `document_right`-Tabelle der Abfrage

hinzugefügt (Zeile 68) und anschließend die Abfrage auf Dokumente eingeschränkt, bei denen für die Benutzergruppe ein Eintrag in der `document_right`-Tabelle existiert (Zeile 69), deren `read`-Feld einen positiven Wert enthält (Zeile 70). Anschließend werden aus der Ergebnismenge noch alle Dokumente herausgefiltert, die über das Feld `is_deleted` als gelöscht gekennzeichnet wurden und somit für reguläre Benutzer nicht sichtbar sein dürfen (Zeile 71). Die eigentliche Abfrage erledigt letztendlich die ursprünglich dafür vorgesehene Methode in der Elternklasse (Zeile 74).

Ähnliche Kontrollstrukturen sind in allen Methoden vorhanden, die Objekte abfragen, welche einer möglichen Zugriffsbeschränkung unterliegen. Neben der `FolderPeer`- und `DocumentPeer`-Klasse sind dies auch die Klassen `NotePeer` und `DocumentVersionPeer`, die ihre Zugriffsrechte von dem übergeordneten Dokument erben. Hat der aktuelle Benutzer also für ein bestimmtes Dokument keine Leserechte, so liefern die Abfrage-Methoden von `NotePeer` und `DocumentVersionPeer` für dieses Dokument keine Ergebnisse.

Dadurch wird gewährleistet, dass in der Steuerungsschicht nur die Objekte zur Verfügung stehen, auf die der aktuelle Benutzer auch tatsächlich Zugriffsrechte hat.

4.3.2.3 Rechteüberprüfung mit Methoden der Objektklasse

Da diese Berechtigungsüberprüfung lediglich die Leserechte berücksichtigt, enthalten die Objektklassen für Ordner und Dokumente zusätzlich Methoden, die die Abfrage einzelner Berechtigungen zulassen. Für die `Document`-Klasse sind dies `isReadable()`, `isEditable()` und `isDeletable()` entsprechend der drei Rechte, die für diesen Objekttyp vergeben werden können. Analog hierzu besitzt die `Folder`-Klasse die Methoden `isReadable()` und `isCreatable()`[7] sowie zusätzlich `isEditable()` und `isDeletable()`, welche jedoch nicht auf die in der `folder_right`-Tabelle hinterlegten Berechtigungen zugreifen, sondern lediglich überprüfen, ob der aktuelle Benutzer Administratorrechte besitzt.

Die meisten `is`-Methoden können mit zwei optionalen Parametern aufgerufen werden. Der erste Parameter legt fest, ob bei der Berechtigungsüberprüfung die Administratorenrechte berücksichtigt werden sollen. Wird hier `true` übergeben, was dem Standardwert entspricht, liefert die Methode in jedem Fall `true` zurück, falls der aktuelle Benutzer ein Administrator ist, auch wenn dieser aufgrund seiner Benutzergruppe eigentlich keine Berechtigung besäße. Über den zweiten Parameter

[7] Ein Neologismus der dem Uniformitätsdrang des Autors geschuldet ist.

können die Rechte für eine andere Benutzergruppe als die des aktuellen Benutzers ermittelt werden. Alternativ kann hier auch ein konkreter Benutzer übergeben werden, dessen Benutzergruppe wird dann automatisch erkannt.

Die `is`-Methoden werden in allen drei Schichten zur Kontrolle und Umsetzung von Berechtigungen genutzt. Die folgenden drei Beispiele veranschaulichen die Verwendung in der Modell-, der Steuerungs- und der Präsentationsschicht.

```
/lib/model/Document.php
10  class Document extends BaseDocument
11  {
126    public function getDescription()
127    {
128       if(!$this->isNew()) {
129          if($this->isReadable(false)) {
130             return parent::getDescription();
131          } else {
132             return null;
133          }
134       } else {
135          return parent::getDescription();
136       }
137    }
826 }
```

Listing 5: Beispiel für die Verwendung von isReadable() in der Modellschicht

Listing 5 zeigt die überschriebene `getDescription()`-Methode der `Document`-Klasse. Diese gibt die Dokumentbeschreibung nur dann aus, wenn der Benutzer die Leserechte für das Dokument besitzt (Zeile 129-133).

```
/apps/frontend/modules/document/actions/actions.class.php
10  class documentActions extends sfActions
11  {
198    public function executeDocumentEdit(sfWebRequest $request)
199    {
200       $this->document = $this->getRoute()->getObject();
201
202       $this->forward404Unless($this->document->isEditable());
203    }
1541 }
```

Listing 6: Beispiel für die Verwendung von isEditable() in der Steuerungsschicht

In Listing 6 ist ein Auszug aus der `DocumentEdit`-Action zu sehen, also der Action, die aufgerufen wird, wenn ein Benutzer ein Dokument bearbeiten will. Die `isEditable()`-Methode überprüft in diesem Fall, ob der aktuelle Benutzer das Dokument bearbeiten darf. Ist dies nicht der Fall, wird eine 404-Fehlermeldung angezeigt (Zeile 202).

```
/apps/frontend/templates/_listPublicDocumentVersionElement.php
22 <?php if($document->isReadable()): ?>
23    <ul>
24       <li><?php echo link_to(image_tag('icons/document.gif', array('title'
   => 'Zu Dokument wechseln', 'size' => '16x16')), '@document?id=' . $document-
   >getId()) ?></li>
25    </ul>
26 <?php endif ?>
```

Listing 7: Beispiel für die Verwendung von isEditable() in der Präsentationsschicht

Das letzte Beispiel in Listing 7 ist ein Ausschnitt aus einem Designtemplate. Hier wird die Anzeige eines Icon-Links zur Bearbeitung eines Dokuments nur dann angezeigt, wenn der Benutzer die erforderlichen Rechte aufweist (Zeile 22-26).

Dem Dokument-Objekt untergeordnete Objekte, wie die `Note`- oder `DocumentVersion`-Klasse können über die in ihnen implementierte Propel-Methode `getDocument()` ebenfalls auf die Berechtigungsinformationen zugreifen. Listing 8 zeigt ein entsprechendes Beispiel (wobei `$note` eine Instanz der `Note`-Klasse darstellt).

```
1  if($note->getDocument()->isReadable()) {
2     echo $note->getText();
3  }
```

Listing 8: Beispiel für den Zugriff auf Dokumentrechte aus einem DocumentVersion-Objekt

Eine andere Vorgehensweise wird bei der Überprüfung der Zugriffsrechte von Logeinträgen angewandt. Hierauf geht Abschnitt 4.3.4.2 näher ein.

4.3.2.4 Vererbung von Berechtigungen

Die Vererbung von Rechten an untergeordnete Objekte erfolgt in der Steuerungsschicht im `document`-Modul der Anwendung. Bei der Bearbeitung eines Ordners wird beispielsweise nach der Speicherung der neuen Berechtigungen die Methode `updateChildrenRights()` in der `documentActions`-Klasse aufgerufen, welche rekursiv für alle untergeordneten Ordner und Dokumente die Berechtigungen aktualisiert und den übergeordneten Rechten anpasst. Gleiches geschieht, wenn ein Ordner gelöscht und die darin noch vorhandenen Dokumente in einen anderen Ordner verschoben werden.

Dabei wird jedoch nur die Verweigerung einer bestimmten Berechtigung auf die Kindobjekte übertragen. Entzieht ein Benutzer einer Gruppe also das Leserecht, so werden dieser auch alle anderen Rechte für den Ordner selbst sowie für die darin enthaltenen Dokumente und Ordner entzogen. Wird der Gruppe anschließend wieder das Leserecht gegeben, wirkt sich das nicht auf die Kindobjekte aus, da nicht davon

ausgegangen werden kann, dass die Gruppe zugriffsberechtigt für *alle* in dem Ordner enthaltenen Dokumente und Ordner ist.

Die Berechtigungen des übergeordneten Ordners werden schon bei der Erzeugung neuer Ordner oder Dokument berücksichtigt. So hat der Anwender nur Einfluss auf die Rechte, die ihm unter Anwendung der übergeordneten Rechte zur Verfügung stehen. Besitzt eine Benutzergruppe also für den übergeordneten Ordner keine Leserechte, ist es auch nicht möglich, in einem Unterordner oder einem darin enthaltenen Dokument ein Leserecht für diese Gruppe zu vergeben. Umgesetzt wird dies durch eine Deaktivierung der entsprechenden Kontrollfelder in der Methode `prepareFolderRightForms()` beziehungsweise `prepareDocumentRightForms()` der `documentActions`-Klasse, welche die Berechtigungsformulare anhand der existierenden Benutzergruppen dynamisch erstellt.

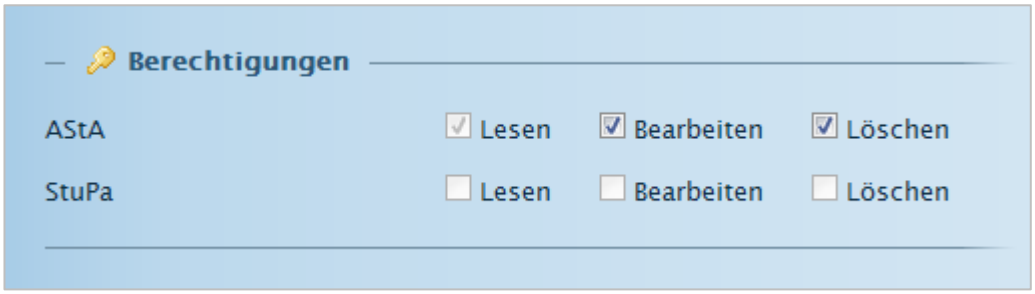


Abbildung 10: Eingabemaske für Dokumentrechte

Abbildung 10 veranschaulicht dies beispielhaft. Im vorliegenden Fall hat der aktuelle Benutzer, ein Mitglieder der AStA-Benutzergruppe, bei der Erzeugung eines neuen Dokuments keine Möglichkeit, der Benutzergruppe StuPa Berechtigungen für das Dokument zu geben, da der übergeordnete Ordner dies verhindert. Ebenso wenig kann der Benutzer sich selbst die Leserechte nehmen[8].

4.3.3 Suchfunktion

4.3.3.1 Umsetzung der Suchfunktion mit externen Bibliotheken

Die Implementierung der Suchfunktionalität erfolgt bei DAVe unter Zuhilfenahme einer externen Bibliothek. Die offizielle Symfony-Dokumentation empfiehlt den Einsatz des Zend Frameworks, speziell der `Zend_Search_Lucene`-Komponente, die eine PHP-Portierung des gleichnamigen Java-Projekts darstellt (vgl. ZANINOTTO/POTENCIER 2009B, S. 208).

[8]Administratoren bilden die einzige Ausnahme.

Lucene wird im Handbuch zum Zend Framework als eine Mehrzwecksuchmaschine beschrieben, die unter anderem nach Relevanz sortierte Ergebnisse liefert, verschiedene Suchoperatoren erlaubt und eine feldorientierte Suche ermöglicht (vgl. ZEND LUCENE).

Die für die Dokumentversionen hochgeladenen Schriftstücke, die in verschiedensten Formaten vorliegen, können von Lucene jedoch nicht indiziert werden. Eine Volltextindizierung über alle Inhalte wäre somit nicht möglich. Um diesem Problem zu begegnen, verwendet DAVe eine weitere Bibliothek namens Tika, welche Möglichkeiten zur Inhaltsextraktion für diverse Dateiformate bereitstellt (s. TIKA). Tika ist ebenfalls Teil des Lucene-Projekts, allerdings existiert für sie keine PHP-Portierung.

Der folgende Abschnitt beschreibt deshalb zunächst die Schnittstelle, die DAVe für den Zugriff auf die Java-basierte Tika-Bibliothek verwendet. Anschließend wird die eigentliche Umsetzung der Indizierung und des Retrievals auf Basis von Lucene vorgestellt.

4.3.3.2 Inhaltsextraktion mit Tika

Tika liegt in Form eines Java-Archives vor, welches sich über die Kommandoebene aufrufen lässt. Neben der zu analysierenden Datei muss beim Aufruf auch ein Ausgabeformat als Parameter übergeben werden. Je nach gewähltem Format gibt Tika den Inhalt der Datei in HTML, XHTML oder als reinen Text direkt in die Kommandozeile aus.

Den Aufruf von Tika kapselt DAVe in der Klasse `dvContentAnalyzer`. Um den Inhalt einer Datei zu extrahieren, muss eine Instanz der `dvContentAnalyzers`-Klasse initialisiert werden. Als einziger Parameter wird dem Konstruktor der Pfad zur Datei übergeben. Die eigentliche Extraktion des Inhalts erfolgt erst mit dem Aufruf der `getContent()`-Methode. Sie liefert den Inhalt der Datei in Form eines Arrays zurück, wobei jede Zeile des Ergebnisses ein Element dieses Arrays darstellt. Standardmäßig liefert `getContent()` den Inhalt im Text-Format, alternativ kann aber auch ein Parameter übergeben werden, der die Rückgabe als XHTML oder HTML bewirkt.

Listing 9 zeigt die beispielhafte Extraktion des Inhalts einer Datei, deren Pfad hier mit `$filePath` übergeben wird.

```
1  $fileContent = new dvContentAnalyzer($filePath);
2  $contentText = $fileContent->getContent();
3
4  echo implode('\n', $contentText);
```

Listing 9: Beispiel einer Inhaltsextraktion

Zunächst erfolgt die Initialisierung von `dvContentAnalyzer` mit dem Dateipfad (Zeile 1). Anschließend wird mit dem Aufruf von `getContent()` die Inhaltsextraktion angestoßen (Zeile 2). Die Ausgabe der Inhaltsextraktionsergebnisse geschieht in Zeile 4. Da dies in Form eines Arrays vorliegt, muss es zunächst mithilfe von `implode()` in einen String umgewandelt werden.

Da der Aufruf von Tika abhängig von der Größe der zu analysierenden Datei sehr rechenintensiv sein kann, werden die Ergebnisse der Tika-Ausführungen in der Datenbanktabelle `file_cache` für den schnellen Abruf gespeichert. Die Funktionsweise schildert Abbildung 11.

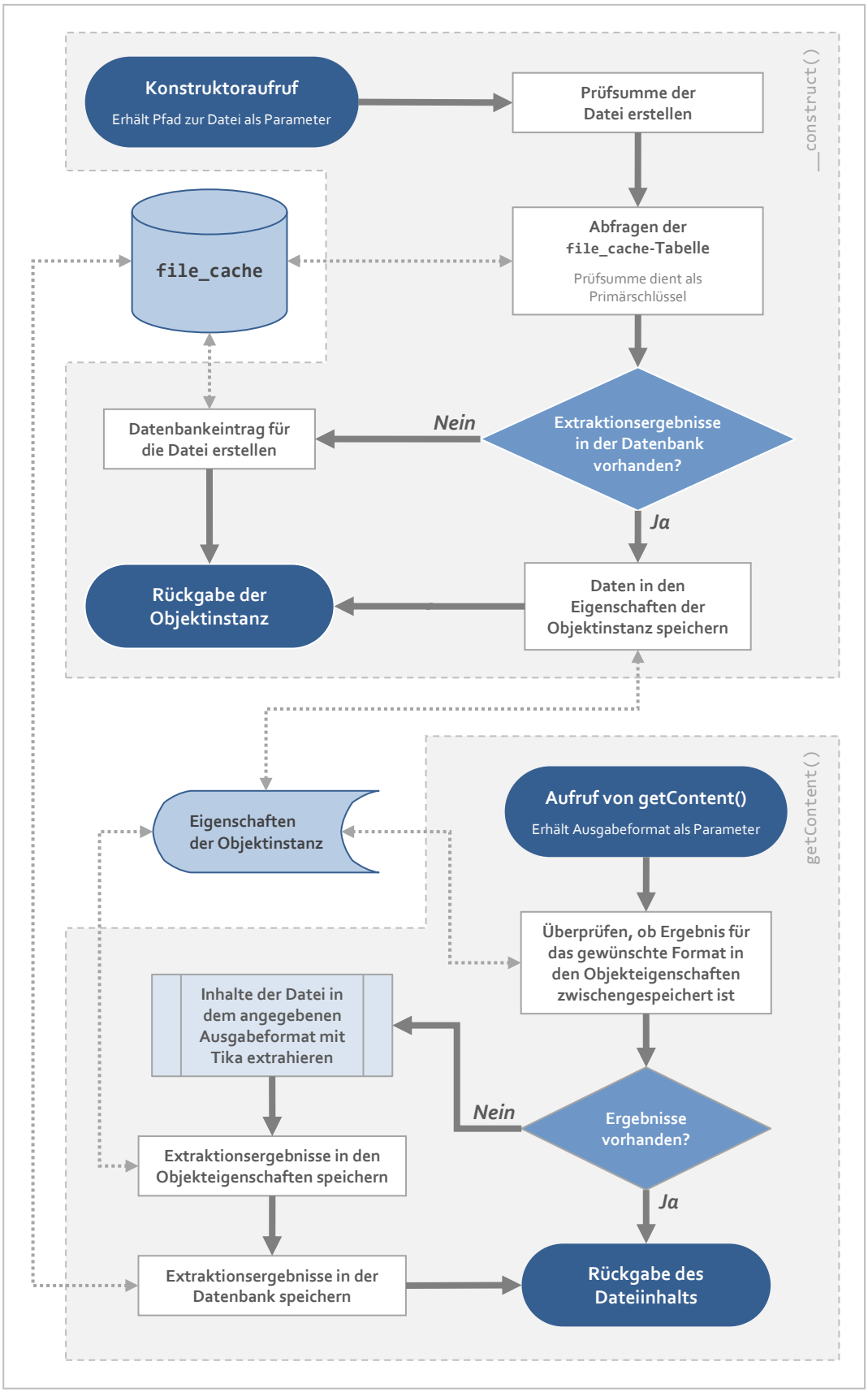


Abbildung 11: Flussdiagramm für die Inhaltsextraktion mit dvContentAnalyzer

Wird der Konstruktor für eine Datei initialisiert, so ermittelt dieser zunächst die eindeutige Prüfsumme der zu analysierenden Datei. Da diese in der `file_cache`-Tabelle als Primärschlüssel dient, kann der `dvContentAnalyzer` überprüfen, ob Tika für diese Datei bereits ausgeführt wurde. Ist dies der Fall, werden die gespeicherten Ergebnisse aus der Datenbank in die Objektinstanz geladen. Dabei existiert für jeden der Formatparameter, mit denen Tika aufgerufen werden kann, ein Feld in der Tabelle beziehungsweise eine Eigenschaft in der Objektklasse.

Bei einem Aufruf der `getContent()`-Methode mit einem der Formatparameter, wird zunächst überprüft, ob im Cache bereits ein Ergebnis für das Format vorliegt. Ist dies der Fall, gibt die Methode den Dateiinhalt aus dem Cache zurück. Sollte noch kein Ergebnis für das angegebene Format vorliegen, wird Tika ausgeführt, das Ergebnis der Ausführung dem Datensatz in der `file_cache`-Tabelle hinzugefügt und anschließend zurückgegeben.

Zusätzlich zu den bereits genannten drei Formaten existiert noch ein viertes Ausgabeformat, welches nur die Meta-Informationen zur Datei extrahiert. Diese Option nutzt DAVe, um den Mime-Type einer Datei zu ermitteln. Das Ergebnis wird, wie auch die Dateiinhalte, im Dateicache abgelegt.

```
/lib/vendor/dave/dvContentAnalyzer.class.php
12  class dvContentAnalyzer
13  {
225    protected function runTika($parameter)
226    {
227       $result = array();
228
229       exec(
230          sprintf('%s -%s "%s"',
231             $this->tika,
232             $parameter,
233             $this->file
234          ),
235          $result
236       );
237
238       return $result;
239    }
240 }
```

Listing 10: Die runTika()-Methode der dvContentAnalyzer-Klasse

Listing 10 zeigt die geschützte `runTika()`-Methode der `dvContentAnalyzer`-Klasse, die den Aufruf der Tika-Bibliothek kapselt. Wie in Zeile 229 zu erkennen ist, wird der PHP-Befehl `exec()` verwendet, um die Anwendung aufzurufen. Deren erster Parameter setzt sich aus drei Teilen zusammen: Dem Pfad zum Tika-Java-Archive (Zeile 231), der vom

Konstruktor der Klasse in der Klasseneigenschaft `tika` hinterlegt wurde, dem Formatparameter (Zeile 232) sowie dem Pfad zur Datei (Zeile 233). Über den zweiten Parameter der `exec()`-Funktion wird PHP angewiesen, die Ausgabe der Ausführung in dem Array `$result` zu speichern.

Der Pfad zur Tika-Anwendung kann in der Anwendungskonfiguration (s. ZANINOTTO/POTENCIER 2009A, S. 61f.) geändert werden, da es möglich ist, dass der Java-Aufruf sich abhängig vom verwendeten Betriebssystem unterscheidet. Ein Beispiel für einen gültigen Pfad wäre `java -jar "..\lib\vendor\apache\tika\tika-0.3.jar"`.

4.3.3.3 Indizierung und Retrieval mit Zend Lucene

DAVe indiziert Ordner, Dokumente, Dokumentversionen, Akten sowie Kommentare und Aufgaben und verwendet hierfür fünf separate Indizes, die den jeweiligen Eigenschaften der Klassen angepasst sind. Den Zugriff auf die Indizes stellt die statische Klasse `dvIndex` über die Methode `getIndex()` bereit, die auch über die `getIndex()`-Methoden der entsprechenden Peer-Klassen aufgerufen werden kann (s. Abbildung 12). Der Rückgabewert der Methode ist ein Objekt der Klasse `Zend_Search_Lucene_Interface`, einer Komponente von Zend Lucene, die wiederum Methoden bereitstellt, mit denen der Index bearbeitet und durchsucht werden kann (s. ZEND LUCENE).

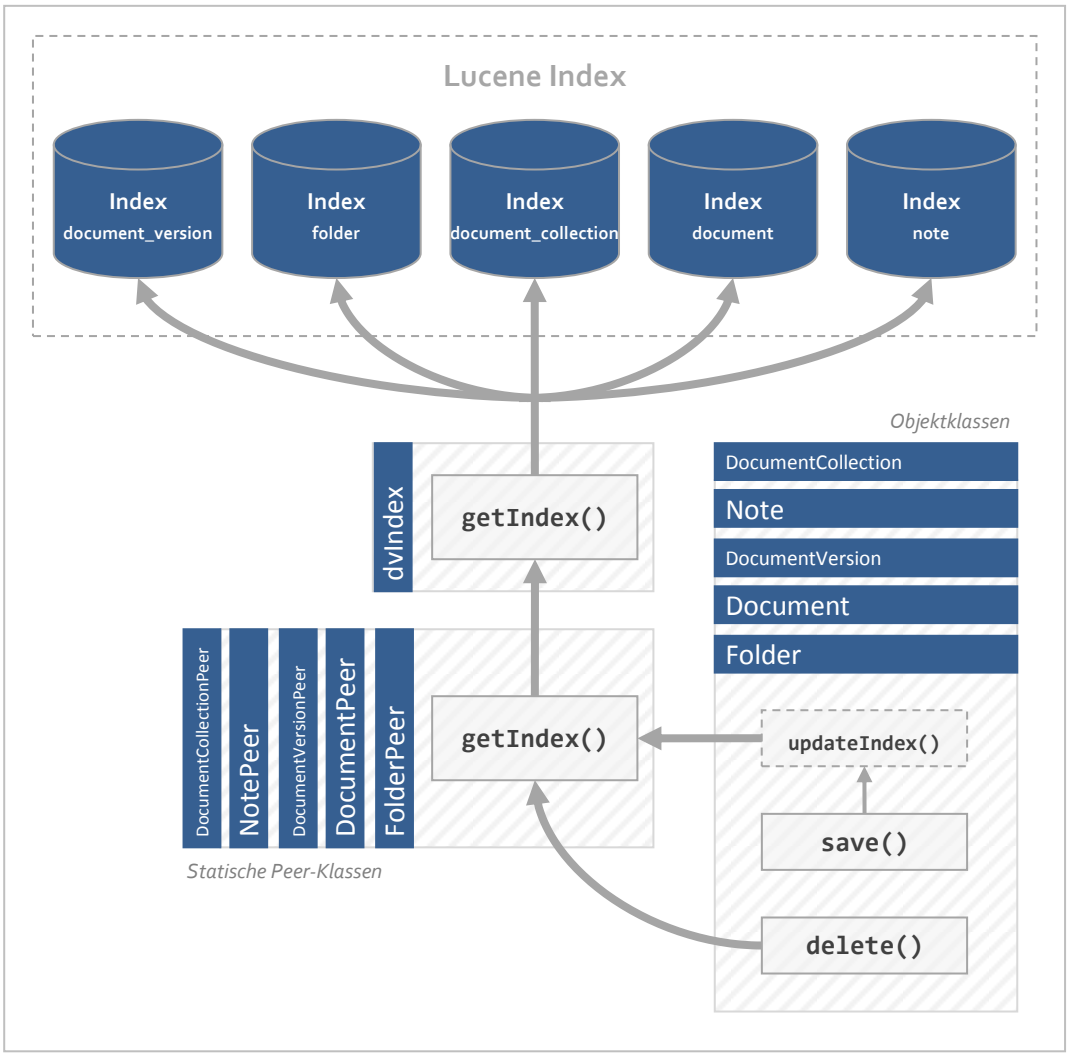


Abbildung 12: Schematische Darstellung der automatischen Indexaktualisierung

Die Aktualisierung der Indizes erfolgt automatisch in der Modellschicht und orientiert sich dabei an der in der offiziellen Symfony-Dokumentation beschriebenen Vorgehensweise (s. ZANINOTTO/POTENCIER 2009B, S. 208ff.). Bei der Erzeugung und Bearbeitung von Objekten wird hierfür die den Objektklassen hinzugefügte `updateIndex()`-Methode aufgerufen, die stets einen neuen Indexeintrag erstellt und im Falle einer Bearbeitung den bereits vorhandenen Eintrag aus dem Index löscht, da Lucene keine Möglichkeit zur Modifikation bestehender Index-Einträge bietet. Der Aufruf von `updateIndex()` erfolgt aus der von Propel für die Objektklasse generierten `save()`-Methode, die für diesen Zweck überschrieben wird. Das Löschen von Index-Einträgen geschieht in der ebenfalls überschriebenen `delete()`-Methode der Objektklasse.

Listing 11 zeigt am Beispiel der `updateIndex()`-Methode der `Folder`-Objektklasse das Zusammenspiel von DAVe und Zend Lucene.

```
/lib/model/Folder.php
10  class Folder extends BaseFolderNestedSet
11  {
441    public function updateIndex()
442    {
443       $index = FolderPeer::getIndex();
444
445
446       // Vorhandene Einträge über diesen Ordner aus dem Index löschen
447       $hits = $index->find('pk:' . $this->getId());
448
449       foreach($hits as $hit) {
450          $index->delete($hit->id);
451       }
452
453
454       // Ordner dem Index hinzufügen
455       $document = new Zend_Search_Lucene_Document();
456
457       $document->addField(Zend_Search_Lucene_Field::keyword('pk', $this-
    >getId()));
458       $document->addField(Zend_Search_Lucene_Field::unStored('name', $this-
    >getName(), 'utf-8'));
459       $document->addField(Zend_Search_Lucene_Field::unStored('description',
    $this->getDescription(), 'utf-8'));
460
461       $index->addDocument($document);
462       $index->commit();
463    }
613 }
```

Listing 11: Die updateIndex()-Methode der Folder-Objektklasse

Über die `getIndex()`-Methode der Peer-Klasse (die wiederum die statische `getIndex()`-Methode der `dvIndex`-Klasse aufruft) wird zunächst eine Instanz von `Zend_Search_Lucene_Interface` geladen und in der Variablen `$index` gespeichert (Zeile 443). Dieses Objekt repräsentiert innerhalb der Methode nun den Lucene-Index für die `Folder`-Klasse und stellt entsprechende Methoden für den Zugriff bereit.

Eine dieser Methoden ist `find()`, welche in Zeile 447 verwendet wird, um bereits existierende Einträge für diesen Ordner im Index zu ermitteln. Da in jedem Indexeintrag ein `pk`-Feld hinterlegt ist, welches die Objekt-ID des indizierten Objekts enthält, ist es so möglich, einen Indexeintrag einem Datensatz zuzuordnen. Die gefundenen Einträge werden dann mithilfe der Lucene-Methode `delete()` aus dem Index gelöscht (Zeile 449-451).

Anschließend wird ein neuer Indexeintrag in Form einer Instanz der `Zend_Search_Lucene_Document`-Klasse erzeugt[9] (Zeile 455). Diesem Eintrag können nun beliebige Felder hinzugefügt werden, welche sich in DAVe abhängig von der indizierten Klasse unterscheiden. Im Falle der Ordner werden der Primärschlüssel des Datensatzes (die Objekt-ID), der Name sowie die Beschreibung dem Indexeintrag hinzugefügt und somit durchsuchbar gemacht (Zeile 457-459).

Über die Lucene-Methoden `addDocument()` und `commit()` erfolgt die Erweiterung des Index um den neuen Eintrag und die Änderungen werden in die Indexdatei geschrieben (Zeile 461f.).

Die `updateIndex()`-Methoden der fünf indizierten Klassen funktionieren alle nach dem beschrieben Prinzip, unterscheiden sich aber in den verwendeten Indexfeldern. Tabelle 13 listet alle indizierten Eigenschaften der durchsuchbaren Informationsobjekte auf.

	Feld	Beschreibung
Ordner	`pk`	Der Primärschlüssel des Datensatzes
	`name`	Der Name des Ordners
	`description`	Die Ordnerbeschreibung
Dokument	`pk`	Der Primärschlüssel des Datensatzes
	`title`	Der Titel des Dokuments
	`description`	Die Dokumentbeschreibung
Dokumentversion	`pk`	Der Primärschlüssel des Datensatzes
	`comment`	Die Anmerkung zur Dokumentversion
	`filename`	Der von DAVE erzeugte Dateiname
	`text`	Der Volltext der hochgeladenen Datei
Kommentar/Aufgabe	`pk`	Der Primärschlüssel des Datensatzes
	`text`	Der Text des Kommentars bzw. der Aufgabe
	`refers_to`	Im Falle einer Aufgabe der Freitextbezug
Akte	`pk`	Der Primärschlüssel des Datensatzes
	`title`	Der Titel der Akte

Tabelle 13: Die Indexfelder der indizierten Informationsobjekte

Eine vom Benutzer in das Suchfeld eingegebene Suchanfrage wird zunächst auf eine Mindestlänge geprüft und anschließend unverändert an die `find()`-Methoden der fünf

[9] Lucene bezeichnet Index-Einträge als „Document“. Dies darf nicht mit den Dokument-Objekten in DAVe verwechselt werden.

Indizes weitergegeben. Aus den zurückgegebenen, nach Relevanz sortierten Ergebnissen werden für jeden Treffer anhand des pk-Felds die entsprechenden Datensätze aus der Datenbank geladen und dem Benutzer anschließend in einer den fünf Indizes entsprechend kategorisierten Ergebnisliste präsentiert.

Da die Suchanfragen direkt an die `find()`-Methode übergeben werden, ist neben der alle indizierten Felder umfassenden Volltextsuche auch das feldbasierte Retrieval möglich. Hierfür muss der entsprechende Feldbezeichner mit Doppelpunkt dem Suchbegriff vorangestellt sein. Eine Suche nach `comment:discovery` würde so alle Dokumentversionen hervorbringen, die das Wort „discovery“ in ihren Anmerkungen enthalten. Darüber hinaus umfasst Lucenes Abfragesprache viele weitere Elemente, wie beispielsweise boolesche Operatoren, Wildcards sowie die Möglichkeit, eine unscharfe Suche durchzuführen. Eine vollständige Referenz befindet sich in der Dokumentation zum Zend Framework (s. ZEND LUCENE).

4.3.4 Ereignisprotokollierung

4.3.4.1 Protokollierung von Vorgängen

Das Ereignisprotokoll verwendet statt statischer Meldungen dynamisch generierte Beschreibungen der Vorgänge, die der jeweilige Logeintrag dokumentiert. Das gewährleistet, dass die in den Einträgen angezeigten Informationen über die referenzierten Objekte stets aktuell sind und Administratoren zwar die Vorgänge sehen, die ihrer Benutzergruppe normalerweise verborgen blieben, diese aber keine Hinweise auf den Inhalt der referenzierten Objekte enthalten.

Die Protokollkomponente von DAVe besteht aus den bereits vorgestellten Datenbanktabellen `log` und `log_entry_type` sowie einer umfangreichen Klassenbibliothek, die für die Erzeugung neuer Logeinträge verwendet wird. Im Folgenden gilt es, zunächst die Struktur der Datenbanktabellen zu erläutern, um das Prinzip der Ereignisprotokollierung zu verdeutlichen. Anschließend soll näher auf die Erstellung neuer Logeinträge eingegangen werden.

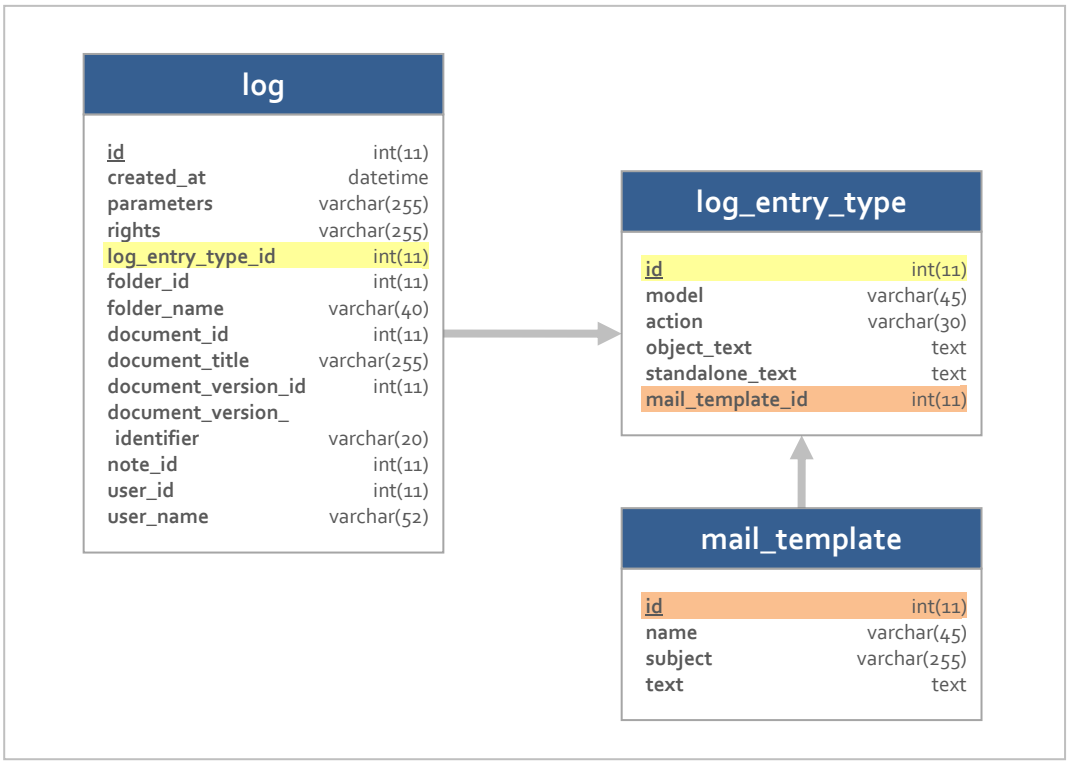


Abbildung 13: Schematische Darstellung der Tabellen des Ereignisprotokolls

Wie in Abbildung 13 zu erkennen ist, enthält die `log`-Tabelle eine ganze Reihe von Feldern, in denen die Informationen zum Vorgang abgelegt werden, welcher von einem Datensatz in der Tabelle repräsentiert wird. Diese umfassen neben dem Zeitpunkt (`created_at`) und dem Verursacher (`user_id` und `user_name`) vor allem Referenzen auf das Objekt, auf welches im Rahmen des Vorgangs zugegriffen wurde.

Im Falle von dokumentbezogenen Objekten, wie Kommentaren und Dokumentversionen, wird dabei nicht nur das Objekt selbst referenziert, sondern auch das entsprechende Dokument. Erfolgt also beispielsweise eine Logeintragung zu einem Kommentar, enthält der Datensatz nicht nur eine Referenz auf diesen, sondern auch auf das Dokument, das ihn enthält. Dadurch ist es unter anderem möglich, mit einer einzelnen Datenbankabfrage alle Ereignisse abzurufen, mit einem Dokument in Zusammenhang stehen.

Die Referenzen liegen dabei als Fremdschlüssel vor sowie zusätzlich in Form einer Zeichenkette, die zum Beispiel den Namen des Dokuments enthält. Dies geschieht, damit der Logeintrag auch dann noch angezeigt werden kann, wenn das betreffende Objekt nicht mehr zur Verfügung steht.[10]

Tabelle 14 bietet eine Übersicht über alle Referenzfelder.

[10] Gleiches gilt für die Tabellenfelder `user_id` und `user_name`.

Feld	Fremdschlüssel	Beschreibung
folder_id	Ja	Referenz auf einen Ordner
folder_name	Nein	Der Name des Ordners
document_id	Ja	Referenz auf ein Dokument
document_title	Nein	Der Titel des Dokuments
document_version_id	Ja	Referenz auf eine Dokumentversion
document_version_identifier	Nein	Die Versionsnummer der Dokumentversion
note_id	Ja	Referenz auf einen Kommentar bzw. eine Aufgabe

Tabelle 14: Die Referenzfelder der log-Tabelle

Weitere Informationen zu einem Vorgang können in dem Feld `parameters` in Form eines serialisierten Arrays abgelegt werden.

Was tatsächlich mit dem referenzierten Objekt geschehen ist, verbirgt sich hinter dem Fremdschlüssel `log_entry_type_id`. Dieser verweist auf die in der Tabelle `log_entry_types` hinterlegten Vorgangsarten. Eine Liste der zur Verfügung stehenden Vorgangsarten zeigt Tabelle 11 in Abschnitt ***3.3.5 Ereignisprotokoll und Benachrichtigungen***.

Für jede Vorgangsart enthält die Tabelle zwei Textvorlagen, aus denen die Vorgangsbeschreibung des Logeintrags bei der Ausgabe erzeugt wird, sowie gegebenenfalls einen Verweis auf die `mail_template`-Tabelle, die die entsprechende Textvorlage für die E-Mail-Benachrichtigung bereitstellt. Neben der ID kann eine Vorgangsart auch anhand der in Kombination eindeutigen Felder `model` und `action` ermittelt werden. So steht die Vorgangsart mit dem `model` „document“ und der `action` „create“ für den Vorgang einer Dokumenterzeugung.

Ein neuer Logeintrag wird nicht – wie sonst bei Symfony üblich – über die Initialisierung eines neuen `Log`-Objekts erstellt, welchem dann mit den von Propel bereitgestellten Setter-Methoden die entsprechenden Werte zugewiesen werden. Dies übernimmt eine Reihe von Klassen der `dvEvent`-Familie, deren Hierarchie Abbildung 14 darstellt.

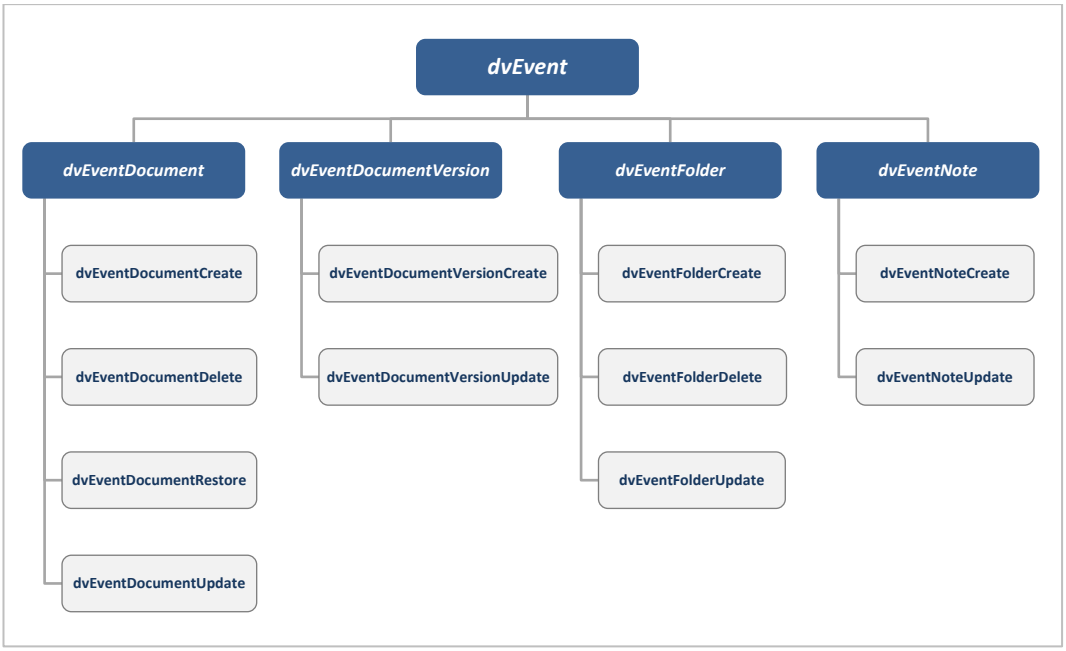


Abbildung 14: Hierarchische Darstellung der dvEvent-Klassenfamilie

Wie die Namen der Objektklassen schon erkennen lassen, repräsentieren die Einträge auf der untersten Ebene die einzelnen Vorgangsarten. Jede dieser Klassen hat somit ein Pendant in der Tabelle `log_entry_type`. In der darüber liegenden Ebene befinden sich abstrakte Klassen, die die einzelnen Vorgangsarten nach der Klasse des referenzierten Objekts gruppieren. Die Stammklasse der Familie befindet sich auf erster Ebene in Form der abstrakten `dvEvent`-Klasse.

Diese Hierarchie erlaubt es, Prozeduren, die sich für mehrere Vorgangsarten wiederholen, in der übergeordneten Klasse zu kapseln. Ebenso enthält die `dvEvent`-Klasse die Funktionalitäten, welche für alle Vorgangsarten benötigt werden. Tabelle 15 veranschaulicht diese Arbeitsteilung anhand der Klasse `dvEventDocumentCreate`. Die einzelnen Schritte der Erzeugung eines neuen Logeintrags werden hier jeweils für die Klasse aufgeführt, in der sie stattfinden.

dvEvent	dvEventDocument	dvEventDocumentCreate
Konstruktor Erzeugen einer neuen Instanz des Propel-Objekts Log		

Konstruktor
Setzen der Felder
`user_id`
`user_name`
 	
`created_at` wird automatisch von Propel gefüllt		

	configure()-Methode
	Setzen der Felder
	`document_id`
	`document_title`
	`rights`
	`parameters`	

		configure()-Methode **Setzen des Felds** `log_entry_type`
sendNotifications()-Methode Senden der E-Mail-Benachrichtigungen		
Konstruktor Speichern des Log-Objekts in der Datenbank		

Tabelle 15: Arbeitsteilung in der dvEvent-Klassenfamilie am Beispiel von dvEventDocumentCreate

Um einen Vorgang zu dokumentieren, reicht es, eine Objektinstanz der entsprechenden Klasse zu erstellen und als Parameter das vom Logeintrag referenzierte Objekt zu übergeben. Der entsprechende Datensatz wird bereits mit dem Konstruktoraufruf des Log-Objekts erzeugt. Die Übergabe des Parameters kann auf vier Wegen erfolgen, wie Listing 12 veranschaulicht.

```
1  $document = new Document();
2  /* ... */
3  $document->save();
4  $documentId = $document->getId();
5
6  // Objekt übergeben
7  $event = new dvEventDocumentCreate($document);
8
9  // Objekt-ID übergeben
10 $event = new dvEventDocumentCreate($documentId);
11
12 // Objekt als Teil eines Parameter-Arrays übergeben
13 $event = new dvEventDocumentCreate(array('document' => $document));
14
15 // Objekt-ID als Teil eines Parameter-Arrays übergeben
16 $event = new dvEventDocumentCreate(array('document_id' => $documentId));
```

Listing 12: Beispiele für das Erzeugen eines Logeintrags

In dem Beispiel wird zunächst ein `Document`-Objekt erstellt und in der Datenbank gespeichert[11] (Zeile 1-3). Im weiteren Verlauf stehen das `Document`-Objekt in der Variablen `$document` und die ID des Datensatzes in der Variablen `$documentId` zur Verfügung.

Die Zeilen 6 bis 16 zeigen die vier Möglichkeiten, einen Logeintrag für den zuvor stattgefundenen Vorgang zu erstellen und das zu referenzierende Objekt als Parameter zu übergeben. In den Zeilen 7 und 10 wird das Objekt selbst beziehungsweise dessen ID direkt als Parameter verwendet. Die Beispiele in den Zeilen 13 und 16 verwenden jeweils ein Array, in welchem das Objekt dem Schlüssel „document“ bzw. die Objekt-ID dem Schlüssel „document_id“ zugewiesen wird. Die Möglichkeit, Parameter in Form eines Arrays zu übergeben erlaubt es, neben dem zu referenzierenden Objekt noch weitere Parameter für den Logeintrag zu definieren, die anschließend entweder beim Erzeugen des Logeintrags verarbeitet oder im `parameters`-Feld der `log`-Tabelle abgelegt werden.

Die übrigen Informationen, die für die Erzeugung des Logeintrags benötigt werden, ermitteln die `dvEvent`-Klassen automatisch. Bei obigem Beispiel wird als Zeitpunkt die aktuelle Systemzeit verwendet und der Verursacher des Vorgangs ist der derzeit angemeldete Benutzer (es sei denn, im Parameter-Array existiert ein gültiger „user“- oder „user_id“-Schlüssel). Die Referenz auf den Ordner sowie die Zugriffsrechte für den Logeintrag werden dem übergebenen `Document`-Objekt entnommen. Das

[11] Das Setzen der Dokumenteigenschaften wurde hier aus Platzgründen herausgenommen, muss allerdings vor dem Aufruf der `save()`-Methode stattfinden, da sonst eine Exception geworfen wird.

log_entry_type_id-Feld wird anhand von in den Klassen hinterlegten Konstanten für die Felder model (hier „Document“) und action (hier „create“) gesetzt.

Da es sehr aufwändige Datenbankabfragen zur Folge hätte, würden die Berechtigungen des Logeintrags über die referenzierten Dokumente und Ordner ermittelt werden, speichert DAVe die Leserechte des Datensatzes in Form eines serialisierten Arrays mit den IDs der berechtigten Benutzergruppen und gewährleistet, dass diese im Falle einer Änderung der Berechtigungen des referenzierten Objekts aktualisiert werden.

4.3.4.2 Abfragen des Ereignisprotokolls

Die Ausgabe von Logeinträgen findet in DAVe ausschließlich über die Klasse dvEventList statt. Sie kapselt die Abfrage der Ereignisse nach den angegebenen Kriterien und die Umsetzung der in den Logeinträgen festgelegten Leserechte. Durch die Implementierung der PHP-Schnittstelle Iterator ermöglicht sie zudem die Iteration der Ergebnisse mittels einer foreach-Schleife (s. BERGMANN 2005, S. 45f.).

Der Konstruktor der dvEventList-Klasse erwartet neben den Parametern $limit und $offset, die den Umfang der Ergebnisse bestimmen, ein Array, welches Filterkriterien für die Auswahl der Ereignisse enthält. Beispiele für verfügbare Filter zeigt Listing 13. In jedem der fünf Beispiele werden die zehn neuesten Logeinträge beginnend mit dem ersten ermittelt.

```
1  // Alle Ereignisse ohne Einschränkung
2  $events = new dvEventList(10);
3
4  // Ereignisse, die Dokumente oder deren Unterelemente in dem Ordner mit der
   ID 42 betreffen
5  $events = new dvEventList(10, 0, array('restrictToFolderId' => 42));
6
7  // Ereignisse, das Dokument mit der ID 47 oder deren Unterelemente betreffen
8  $events = new dvEventList(10, 0, array('restrictToDocumentId' => 47));
9
10 // Ereignisse, die den Kommentar bzw. die Aufgabe mit der ID 89 betreffen
11 $events = new dvEventList(10, 0, array(
12    'model' => LogEntryTypePeer::MODEL_NOTE,
13    'objectId' => 89
14 ));
15
16 // Ereignisse, die Dokumente oder Dokumentversionen betreffen
17 $events = new dvEventList(10, 0, array(
18    'model' => array(
19       LogEntryTypePeer::MODEL_DOCUMENT,
20       LogEntryTypePeer::MODEL_DOCUMENT_VERSION
21    )
22 ));
```

Listing 13: Beispiele für den Einsatz von dvEventList mit verschiedenen Filterkriterien

Die Filterung der Datensätze erfolgt innerhalb von `dvEventList` über die von Propel erzeugte `doSelect()`-Methode der `LogPeer`-Klasse. Anschließend werden sie zu Instanzen der Klasse `LogEntry` konvertiert, welche von der Propel-Objektklasse `Log` ableitet. Über `LogEntry` lässt sich die dynamisch generierte Vorgangsbeschreibung ausgeben und durch Überschreiben der entsprechenden Methoden ist es nicht mehr möglich, dass der zugrundeliegende Datensatz modifiziert und damit verfälscht wird.

Da die Leserechte in Form eines serialisierten Arrays in der Datenbank gespeichert werden, kann einer Überprüfung erst nach der Abfrage stattfinden. Hierfür stellt die `Log`-Klasse die `isReadable()`-Methode bereit, die die im Array hinterlegten Benutzergruppen-IDs mit der des aktuellen Benutzers vergleicht.

Die Paginierung der Ergebnisse übernimmt die `dvEventListPager`-Klasse, die von der abstrakten Symfony-Klasse `sfPager` ableitet und das Prinzip von `sfPropelPager`[12] auf die `dvEventList`-Klasse überträgt (s. ZANINOTTO/POTENCIER 2009B, S. 91ff.).

4.3.4.3 Erzeugen der Vorgangsbeschreibung

Die Ausgabe der Vorgangsbeschreibung erfolgt über die `getText()`-Methode des `LogEntry`-Objekts. Diese überprüft zunächst, ob das referenzierte Objekt noch verfügbar ist. Ist dies der Fall, so dient der Inhalt des `object_text`-Felds der `log_entry_type`-Tabelle als Textvorlage. Ist das referenzierte Objekt nicht mehr verfügbar, wird der Inhalt des `standalone_text`-Felds verwendet.

Die Textvorlagen enthalten durch Prozentzeichen gekennzeichnete Platzhalter, die bei der Generierung der Vorgangsbeschreibung mit den Informationen des referenzierten Objekts ersetzt werden. Ein Platzhalter besteht dabei aus zwei Teilen, die durch einen Schrägstrich voneinander getrennt werden. Der erste Teil beschreibt den Namensraum, der zweite Teil die Eigenschaft, die eingefügt werden soll. Ein Beispiel für einen solchen Platzhalter wäre `%document/title%`.

Zusätzlich kann ein Platzhalter noch von Ausrufezeichen umschlossen werden. Dies weist den Template-Parser an, den einzufügenden Text zusätzlich mit dem entsprechenden Objekt zu verlinken. Der obige Platzhalter mit Verlinkung wäre demnach `%!document/title!%`.

Für das Parsen der Templates verwendet die `getText()`-Methode die Klasse `dvEventTemplateParser`. Die Initialisierung einer neuen Objektinstanz erwartet als

[12] Die abstrakte sfPager-Klasse ist von Symfony nicht dokumentiert

Parameter den zu parsenden Text sowie das LogEntry-Objekt, für welches der Text geparst wird. Über die Methode getText() liefert das Parserobjekt anschließend den angepassten Text zurück. Dieser wird in der LogEntry-Objektinstanz zwischengespeichert und muss somit nicht bei jedem Aufruf von getText() von Neuem erzeugt werden.

Das eigentliche Parsen des Textes übernimmt die geschützte Methode parse() der dvEventTemplateParser-Klasse. Die wiederum sucht unter Verwendung eines regulären Ausdrucks alle im Vorlagentext vorhandenen Platzhalter und ersetzt diese mithilfe der privaten Methode replaceTextVariables().

Den einzufügenden Text ermittelt der Parser anhand der Bestandteile des Platzhalters. Der Namensraum verweist auf das Objekt, aus dem die Information bezogen wird, und die Eigenschaft auf die Objekteigenschaft, in welcher die Information abgelegt ist. Der Platzhalter %note/teaser% referenziert demnach die teaser-Eigenschaft des Note-Objekts.

Den Zugriff auf diese Eigenschaft verschafft sich der Parser über das übergebene LogEntry-Objekt. Er verwendet dabei die von Propel bereitgestellte Methode getNote() der LogEntry-Klasse und anschließend die getTeaser()-Methode der zurückgelieferten Note-Objektinstanz.

Analog wird bei dem Platzhalter %document/title% verfahren. Über getDocument() erhält der Parser Zugriff auf das referenzierte Document-Objekt und dessen getTitle()-Methode liefert den gewünschten Text für den Platzhalter. Ist als Namensraum „this“ angegeben, wird statt eines referenzierten Objekts die entsprechende Eigenschaft des LogEntry-Objekts verwendet. %this/document_title% gibt somit über die LogEntry-Methode getDocumentTitle() den im Logeintrag hinterlegten Dokumenttitel zurück.

```
/lib/vendor/dave/event/dvEventTemplateParser.class.php
12  class dvEventTemplateParser
13  {
72     private function replaceTextVariables($match)
73     {
97        if($namespace == 'this') {
98           $addLink = false;
99           $object = $this->logEntry;
103       } else {
104          if(method_exists($this->logEntry, 'get' .
    dvHelper::toCamelCase($namespace, true))) {
105             $object = call_user_func(array($this->logEntry, 'get' .
    dvHelper::toCamelCase($namespace, true)));
106          } else {
107             $object = null;
108          }
109       }
110
115       if(!is_null($object) && method_exists($object, 'get' .
    dvHelper::toCamelCase($property, true))) {
116          $text = call_user_func(array($object, 'get' .
    dvHelper::toCamelCase($property, true)));
117       } else {
118          $text = null;
119       }
131    }
132 }
```

Listing 14: Auszug aus der replaceTextVariables()-Methode der dvEventTemplateParser-Klasse

Die entsprechenden Getter-Methoden werden anhand des Platzhalters dynamisch zur Laufzeit generiert, wie der Ausschnitt aus der `dvEventTemplateParser`-Klasse in Listing 14 zeigt.

Ist der im Platzhalter angegebene Namensraum „this“, wird als vom Platzhalter referenziertes Objekt das übergebene `LogEntry`-Objekt verwendet (Zeile 97-99). Andernfalls erfolgt eine Überprüfung, ob für den Namensraum eine Getter-Methode existiert. Dabei wird der im Underscore-Format angegebene Namensraum in das CamelCase-Format umgewandelt und mit dem Suffix „get“ versehen (Zeile 104). Existiert die Methode, wird diese ausgeführt und das zurückgelieferte Objekt in der Variablen `$object` gespeichert (Zeile 105). So führt beispielsweise der Namensraum „document_version“ zu der von Propel erzeugten `LogEntry`-Methode `getDocumentVersion()`.

Ab Zeile 115 wiederholt sich der Vorgang für die im Platzhalter angegebene Eigenschaft. Zunächst erfolgt die Überprüfung, ob eine entsprechende Getter-Methode für das in `$object` abgelegte Objekt existiert (Zeile 115). Ist dies der Fall, wird der Wert der Eigenschaft über diese Methode ermittelt und in der Variablen `$text` gespeichert (Zeile 116).

Für die gegebenenfalls gewünschte Verlinkung des Textes verwendet der Parser den Namensraum als Routennamen für das Symfony-Routingsystem und die über `getId()` ermittelte ID des referenzierten Objekts als URL-Parameter (vgl. ZANINOTTO/POTENCIER 2009A, S. 152ff.).

5 Visuelle Umsetzung

5.1 Grafisches Konzept

Das Erscheinungsbild – insbesondere der erste Eindruck – darf bei der Entwicklung einer Webanwendung nicht unterschätzt werden. Sue Batley geht sogar so weit zu sagen, dass „the attractiveness (or unattractiveness) of the display may determine whether or not people will use a system" (BATLEY 2007, S. 136). Sie schreibt weiter, dass auch die Qualität der angebotenen Informationen, eine gute Navigationsstruktur, leistungsfähige Retrievalfunktionen oder eine starke Ausrichtung an den Bedürfnissen der Benutzer den negativen Effekt, den ein unattraktives Design auf den Anwender hat, nicht ausgleichen können (vgl. ebd.).

Deshalb gilt es, bei der Bedienoberfläche von DAVe eine Balance zwischen Benutzerfreundlichkeit, Übersichtlichkeit und Attraktivität zu finden. Die im Folgenden beschriebene Umsetzung versucht, diesen Kriterien gerecht zu werden.

Die technische Realisierung der grafischen Oberfläche erfolgt bei Symfony mithilfe einfacher HTML-Templates, in die mit regulären PHP-Befehlen sowie vom Framework bereitgestellten Helper-Funktionen um dynamischen Inhalt ergänzt werden (vgl. ZANINOTTO/POTENCIER 2009A, S. 98ff.). Dies erfolgt selbstverständlich konform zu aktuellen Webstandards und unter Verwendung eines semantischen Markups.

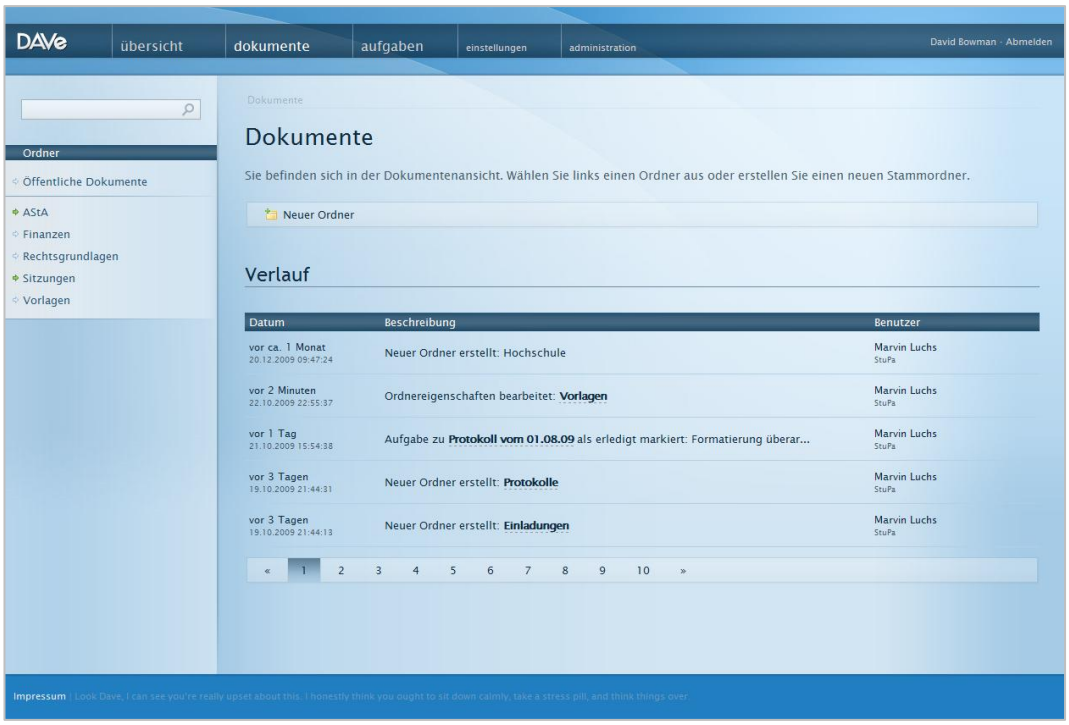


Abbildung 15: Das Design von DAVe

Abbildung 15 zeigt das Standarddesign von DAVe, hier in der Dokumentenansicht. Die vorherrschende Farbe in diesem Design ist blau. Dadurch wird die Seriosität und Professionalität der Anwendung hervorgehoben (vgl. JACOBSEN 2007, S. 174). Variiert wird die Farbgebung durch Schatten, Verläufe und Transparenzeffekte, die dem Design Tiefe verleihen und die Gliederung der Seite ebenso unterstützen wie die klare Trennung von inhaltlichen Elementen mithilfe von White-Space, Trennlinien und dunkel unterlegten Titelelementen (vgl. LYNCH/HORTON 2008, S. 182ff.).

Sowohl in der Struktur als auch in den Inhalten wurde darauf geachtet, stets eine klar zu erkennende Hierarchie aufzubauen (vgl. JACOBSEN 2007, S. 182ff.). Diese äußert sich in der Reihenfolge der Inhaltselemente sowie in den verwendeten Schriftgrößen und -farben. Abkürzungen und Expertenfunktionen, wie beispielsweise der direkte Zugriff auf Dokumentversionen aus der Dokumentliste heraus, sind in der Regel weiter rechts positioniert oder unauffällig gestaltet, während die wichtigsten Informationen und Navigationselemente stets links ausgerichtet und entsprechend akzentuiert sind (vgl. Abbildung 16). Nur selten benötigte Seitenelemente werden zunächst nicht angezeigt und erst bei Bedarf eingeblendet (vgl. Abbildung 17). Das stärkt die Übersichtlichkeit des Seitenaufbaus und vermeidet, dass der Anwender durch sekundäre Informationen irritiert wird (vgl. TATE 2009).

Abbildung 16: Standardansicht der Dokumentenliste

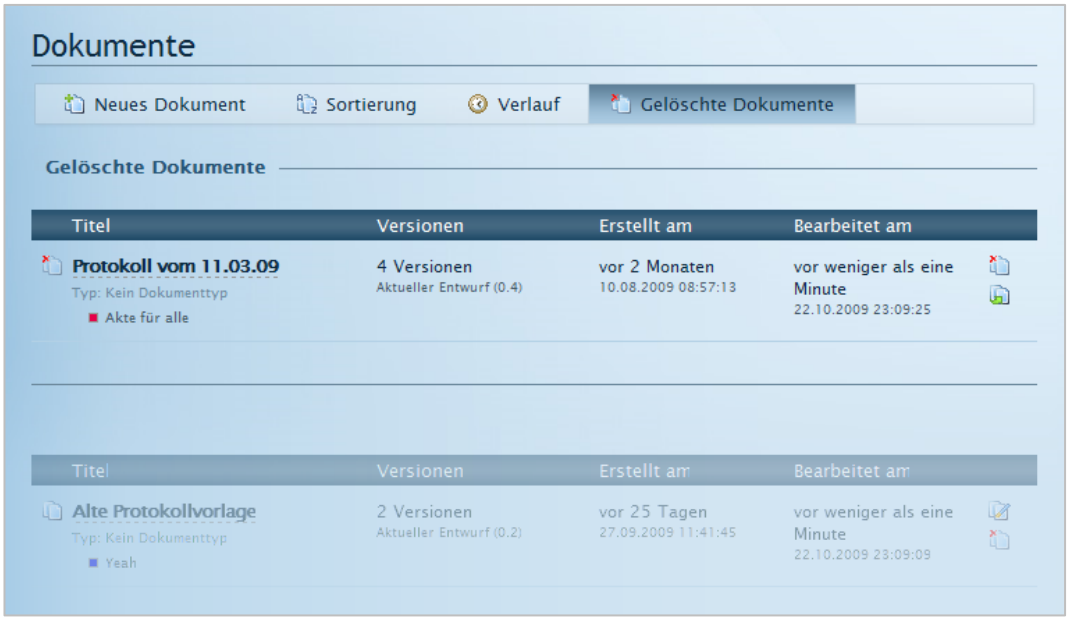


Abbildung 17: Dokumentenliste mit eingeblendeten gelöschten Dokumenten

Stets verfügbare Hinweise auf die momentane Position innerhalb des Systems, wie beispielsweise die Pfadnavigation (s. Abschnitt ***5.3.3 Breadcrumb-Navigation***) und die Hervorhebung von aktiven Menüelementen, sollen den Benutzer bei der Orientierung unterstützen (vgl. JACOBSEN 2007, S. 152).

Die konsequente und einheitliche Verwendung von Icons hilft dem Anwender, wiederkehrende Funktionen und Inhaltsobjekte schneller zu erkennen und ein adäquates mentales Modell der Anwendung zu entwickeln (vgl. BATLEY 2007, S. 127.).

Über AJAX-Anfragen, die im Hintergrund ablaufen und keinen Seitenwechsel zur Folge haben, wird der Anwender über einen Indikator im unteren rechten Bildschirmrand informiert. Dies vermeidet den Eindruck, dass ein Interaktionselement nicht reagiert. Gleiches gilt für das Abschicken von Eingabemasken, deren Verarbeitung länger als gewöhnlich dauert, da beispielsweise auf der Serverseite zunächst eine Indizierung der übermittelten Daten erfolgt. Hier wird der Benutzer über einen entsprechenden Screen

darauf hingewiesen, dass seine Angaben momentan verarbeitet werden (vgl. Abbildung 18).

Abbildung 18: Hinweismeldung bei Vorgängen mit längerer Bearbeitungsdauer

5.2 Design

Entsprechend der vom Konzept vorgegebenen Seitenaufteilung ist das Layout von DAVe in drei große Bereiche gegliedert. Das Design besitzt dabei keine feste Breite, sondern passt sich der Bildschirmauflösung beziehungsweise der Fenstergröße des Anwenders an.

Am oberen Rand befindet sich der Hauptnavigationsbereich (s. ***5.3.1 Hauptnavigation***) mit integriertem Logo sowie Informationen zur aktuellen Sitzung und der Link zum Abmelden (vgl. Abbildung 19). Die horizontale Anordnung der Elemente ermöglicht einen in der Höhe kleinen Header, der mehr Platz für den eigentlichen Inhalt der Seite lässt. Unterlegt ist dieser Bereich mit einem leicht durchscheinenden Dunkelblau, das mit dem helleren Rest des Designs kontrastiert und so angemessen zur Geltung kommt. Das Logo ist, wie der Benutzer es erwartet, mit der Startseite der Anwendung verlinkt (vgl. LYNCH/HORTON 2008, S. 154).

Abbildung 19: Der Header mit Logo, Hauptnavigation und Sitzungsinformationen

Der in Abbildung 20 dargestellte Hauptbereich der Seite weist einen Hintergrund in Milchglasoptik auf. Dies soll dem aufgrund seiner Helligkeit sehr dominanten Bereich des Designs Leichtigkeit vermitteln und zudem gewährleisten, dass der Kontrast zur Schrift weder zu stark noch zu schwach ist und somit angenehm für das Auge des Betrachters erscheint. Der großzügige Einsatz von White-Space und großen Überschriften gliedert die Inhaltselemente dieses Bereichs. Die auffällige Betitelung der einzelnen Elemente trägt bei Listen zudem dazu bei, dass der Anwender schnell erkennt, welchen Inhalt diese enthalten (vgl. JACOBSEN 2007, S. 189ff.).

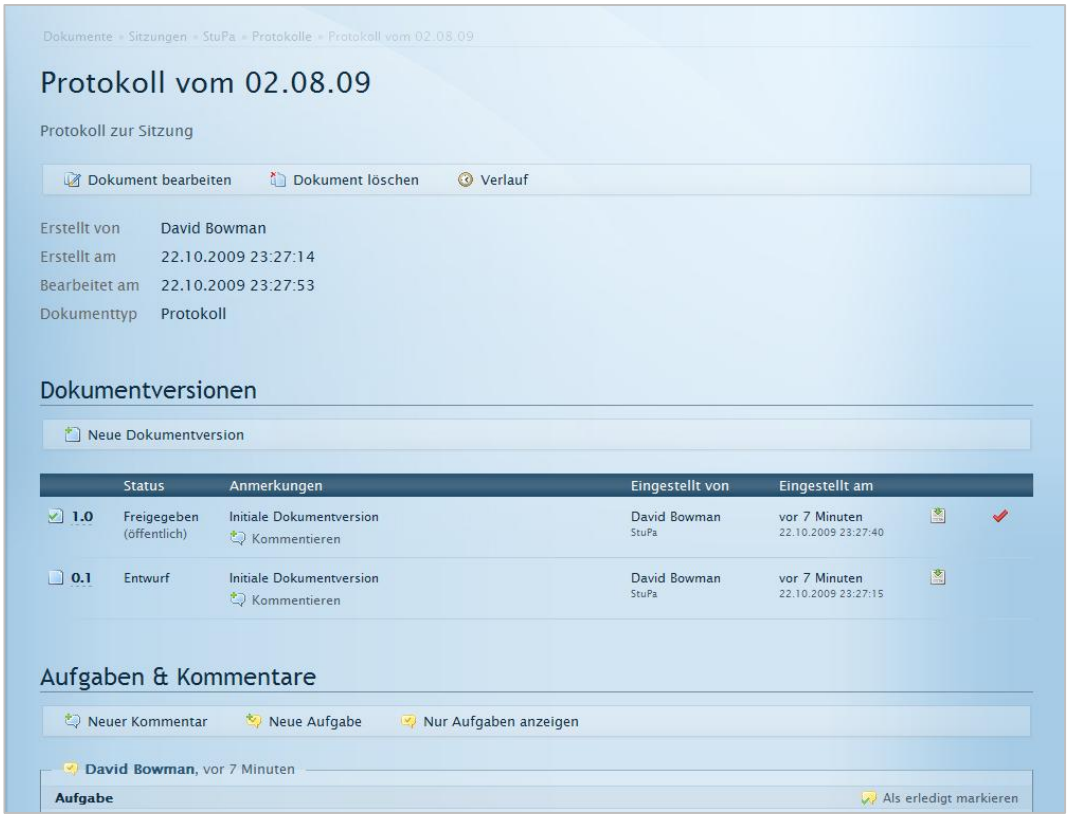


Abbildung 20: Der Hauptbereich des Seitenlayouts

Der linke Seitenbereich hebt sich durch die Farbverläufe im Hintergrund beider Bereiche sowie durch einen Schatten optisch klar vom Hauptbereich ab. Die Inhalte dieses Bereichs werden durch einen boxartigen Charakter gruppiert und sind dank der dunkel unterlegten Überschriften einfach zuzuordnen. Die Positionierung der Suche im Kopf des Seitenbereichs ermöglicht den schnellen Zugriff auf diese Funktion und rückt sie zudem ins Bewusstsein des Anwenders (vgl. LYNCH/HORTON 2008, S. 91f.). Darunter folgen abhängig von der aktuellen Ansicht und dem Kontext variierende Navigationselemente (s. ***5.3.2 Sekundäre Navigation***). Das Beispiel in Abbildung 21 zeigt die Navigationselemente des Aufgabenbereichs.

Abbildung 21: Der Seitenbereich mit Suchfeld und sekundären Navigationselementen

5.3 Navigation

5.3.1 Hauptnavigation

Die Hauptnavigation stellt den Zugang zu den einzelnen Bereichen der Anwendung bereit. Neben der Übersicht sowie dem Dokument- und dem Aufgabenbereich sind dies noch die Benutzereinstellungen und die Systemadministration. Die drei wichtigen Hauptbereiche werden dabei durch ihre Größe klar in den Vordergrund gerückt, die weniger wichtigen Bereiche Einstellungen und Administration werden kleiner dargestellt und treten somit angemessen zurück. Eine Hervorhebung des aktuellen Menüpunkts informiert den Benutzer darüber, in welchem Bereich der Anwendung er sich befindet.

5.3.2 Sekundäre Navigation

Die sekundäre Navigation variiert in Abhängigkeit zur aktuellen Ansicht.

Der Dokumentenbereich verwendet einen interaktiven Ordnerbaum, der in Abschnitt 5.3.4 detaillierter vorgestellt wird. Gegebenenfalls wird auch noch ein zweiter Bereich

angezeigt, der die Akten des aktuellen Ordners auflistet. Diese sind farblich gekennzeichnet und lassen sich so leichter den einzelnen Dokumenten zuordnen.

Der Aufgabenbereich besitzt ebenfalls zwei sekundäre Navigationselemente. Das erste listet die beiden allgemeinen Funktionen dieser Ansicht auf. Das zweite ist ein benutzerspezifisches Menü, welches Zugriff auf die gespeicherten Aufgabenfilter des Benutzers bietet. So hat der Benutzer schnellen Zugriff auf seine persönlichen Aufgabenlisten.

Die übrigen Bereiche verwenden einfache Auflistungen der in der aktuellen Ansicht verfügbaren Kategorien. Dabei wird stets der Menüpunkt hervorgehoben, in dem sich der Benutzer momentan befindet.

5.3.3 Breadcrumb-Navigation

Eine der wichtigsten Navigationshilfen ist die Breadcrumb-Navigation. Sie gibt nicht nur Aufschluss darüber, wo der Benutzer sich in der Anwendung befindet, sondern auch, welchen Weg er dorthin zurückgelegt hat. So hat der Anwender die Möglichkeit, ohne weiteres einen oder mehrere Schritte zurückzugehen.

Abbildung 22: Die Breadcrumb-Navigation in der regulären Darstellung

Die Breadcrumb-Navigation visualisiert zudem besonders im Dokumentenbereich die Hierarchie, in welcher der Benutzer sich befindet und unterstützt ihn somit bei der Entwicklung eines mentalen Modells der Anwendung (vgl. BATLEY 2007, S. 151).

Standardmäßig wird die Pfadnavigation in DAVe transparent dargestellt, um dem Seitendesign die Komplexität zu nehmen und der Überschrift mehr Freiraum zu gewähren (vgl. Abbildung 22). Sobald der Benutzer mit der Maus in die Nähe der Navigationszeile kommt, wird diese mit voller Opazität dargestellt (vgl. Abbildung 23).

Abbildung 23: Die Breadcrumb-Navigation in voller Opazität

5.3.4 Ordnerbaum

Die Navigation im Dokumentenbereich erfolgt über eine Darstellung der Ordnerhierarchie im Seitenbereich, die der Anwender in ihrer Funktionsweise vom Windows Explorer kennt (vgl. Abbildung 24). Das Ein- und Ausblenden von Hierarchieebenen reduziert insbesondere bei vielen Ordnern auf mehreren Ebenen die Komplexität dieser Darstellung, ohne dass die Möglichkeit genommen wird, den kompletten Ordnerbaum darzustellen.

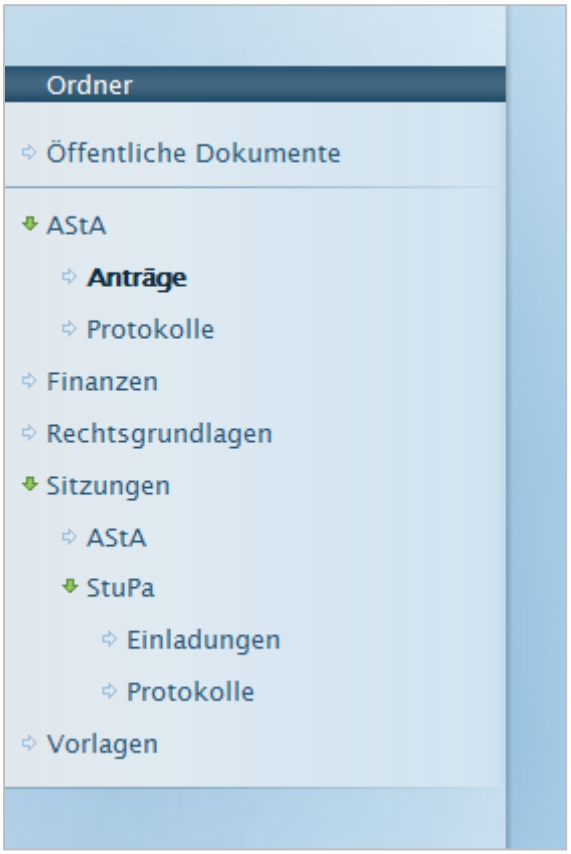


Abbildung 24: Der Ordnerbaum der Dokumentenansicht

Den Ordnern wird hierbei ein kleines Symbol vorangestellt, das den Benutzer darüber informiert, ob dieser Ordner noch Unterordner enthält. Ist dies der Fall, kann durch

einen Klick auf das Icon die darunterliegende Ordnerebene eingeblendet werden, ohne dass die komplette Seite neu geladen wird. Klickt der Anwender hingegen auf den Namen des Ordners, so wird der Inhalt dieses Ordners im Hauptbereich angezeigt. Auf diese Weise können auch tief in der Hierarchie gelegene Ordner schnell erreicht werden. Das stufenweise Einblenden von Hierarchieebenen gewährleistet dabei, dass der Benutzer nicht die Orientierung verliert.

Ruft der Benutzer einen Ordner oder ein Dokument auf, kann er nicht nur anhand der Pfadnavigation erkennen, wo er sich innerhalb des Ordnerbaums befindet, sondern auch an der Darstellung im Seitenbereich. Hier wird stets der komplette Pfad bis zum aktuellen Ordner eingeblendet, so dass der Anwender Zugriff auf die benachbarten Ordner hat, ohne sich zunächst zur aktuellen Position durchklicken zu müssen.

Die Umsetzung dieser interaktiven Darstellung des Ordnerbaums erfolgt unter Verwendung der Methoden, die jQuery für das Unobtrusive DOM Scripting bereitstellt (vgl. HOLT 2008). Der Ordnerbaum wird hierbei zunächst ganz regulär mit ineinander verschachtelten HTML-Listen erzeugt. Erst wenn die Seite vollständig geladen ist, wird die JavaScript-Komponente aufgerufen, die alle Unterebenen versteckt und anschließend den einzelnen Ordnern entsprechende Symbole voranstellt. Steht kein JavaScript zur Verfügung, ist es somit dennoch möglich, den kompletten Ordnerbaum zu verwenden (vgl. Abbildung 25).

Abbildung 25: Der Ordnerbaum bei deaktiviertem JavaScript

Die Umsetzung ist dank der umfangreichen Methoden, die jQuery für die Traversierung des DOM-Baums bereitstellt (s. JQUERY), sehr schlank und übersichtlich.

```
/web/js/folder_tree.js
2  var arrowDown = '/images/icons/arrow_down.gif';
3  var arrowRight = '/images/icons/arrow_right.gif';
4  var arrowRightDisabled = '/images/icons/arrow_right_disabled.gif';
5
7  $(function() {
9     $('#folder_tree ul li ul li').hide();
10
12    var icon = document.createElement('img');
13    icon.src = arrowRightDisabled;
14
17    $('#folder_tree li').prepend(icon);
18
20    $('#folder_tree ul li:has(ul) > img').attr('src', arrowRight);
21    $('#folder_tree ul li:has(ul) > img').css('cursor', 'pointer');
45 });
```

Listing 15: Auszug aus dem JavaScript für die interaktive Darstellung des Ordnerbaums

Listing 15 zeigt einen Auszug aus dem JavaScript, das dem Ordnerbaum seine interaktive Funktionalität gibt. In den ersten drei Zeilen werden zunächst die Pfade zu den drei verschiedenen Symbolen definiert. Der in den Zeilen 7 bis 45 beschriebene Auszug ist der Teil, der nach dem Laden der Seite ausgeführt wird.

Hier werden zunächst alle `LI`-Elemente die innerhalb einer anderen Liste stehen, demnach also Unterebenen darstellen, über die `hide()`-Methode von jQuery ausgeblendet (Zeile 9). Anschließend wird ein neues DOM-Objekt für das Symbol erzeugt, das vor allen Ordnern eingefügt werden soll (Zeile 12f.). Alle Ordner erhalten dabei zunächst das Symbol, welches Ordner ohne Unterordner kennzeichnet (Zeile 17). Über den jQuery-Filter-Selektor `:has()` wird nun bei allen Grafiken in `LI`-Elementen, in denen ein `UL`-Element, also eine Unterebene existiert, das `src`-Attribut modifziert sowie die CSS-Eigenschaft `cursor` mit dem Wert `pointer` hinzugefügt (Zeile 20f.).

Dieses Beispiel zeigt, wie mithilfe von jQuery die sehr komplexe Modifikation des Ordnerbaums schlank und übersichtlich umgesetzt werden kann und dabei außerdem der Vorteil entsteht, dass die grundlegende Funktionalität auch ohne JavaScript bestehen bleibt.

5.3.5 Werkzeugleisten

Ein wichtiges Element der grafischen Oberfläche von DAVe sind die Werkzeugleisten. Sie präsentieren dem Anwender die Funktionen, die ihm im aktuellen Kontext zur

Verfügung stehen. Befindet sich der Benutzer beispielsweise in einem Ordner und besitzt Administratorenrechte, so wird ihm über die Darstellung einer Werkzeugleiste angeboten, einen neuen Ordner zu erstellen oder den Ordner, in dem er sich gerade befindet, zu modifizieren oder zu löschen (vgl. Abbildung 26).

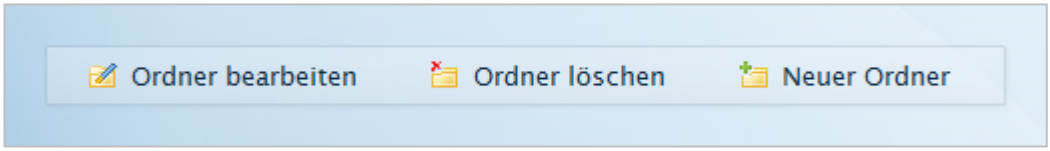


Abbildung 26: Die Werkzeugleiste für Ordner

Ist die aktuelle Ansicht in mehrere Abschnitte gegliedert, so existiert oft für jeden Abschnitt eine Werkzeugleiste, die passend zu den Inhalten dieses Abschnitts die entsprechenden Funktionen bereitstellt. Ein Beispiel hierfür ist die Ansicht eines Dokuments, die Abbildung 27 zeigt. Sowohl für das ganze Dokument, für den Abschnitt „Dokumentversionen“ als auch für den Bereich „Aufgaben & Kommentare“ existieren jeweils eigene Werkzeugleisten, die die entsprechenden Funktionalitäten bereitstellen.

Abbildung 27: Werkzeugleisten in einer Ansicht mit mehreren Abschnitten

Die einzelnen Buttons der Werkzeugleisten sind dabei sowohl mit einer Beschriftung als auch mit einem Symbol versehen. So können sich neue Benutzer, die mit den verschiedenen Symbolen noch nicht vertraut sind, an den Texten der Schaltflächen orientieren, während erfahrene Benutzer eine Funktion schneller an ihrem Symbol erkennen.

Auch bei den Eingabemasken werden Werkzeugleisten verwendet. Sie enthalten die Funktionalitäten zum Abschicken eines Formulars oder zum Abbrechen der Aktion. Auch hier erfolgt der konsequente Einsatz von Symbolen, um die Funktionalität der einzelnen Schaltflächen zu verdeutlichen. Vorgänge, die neue Datensätze oder Änderungen derselben in der Datenbank speichern, sind beispielsweise einheitlich mit einer Diskette gekennzeichnet (vgl. Abbildung 28).

Abbildung 28: Werkzeugleiste zur Verarbeitung einer Eingabemaske

Diese Symbolik ist dem Benutzer aus vielen anderen Anwendungen bekannt und hilft ihm somit, sich schneller zu orientieren. Ebenso werden Signalfarben für das Bestätigen oder Abbrechen von Vorgängen verwendet (vgl. Abbildung 29).

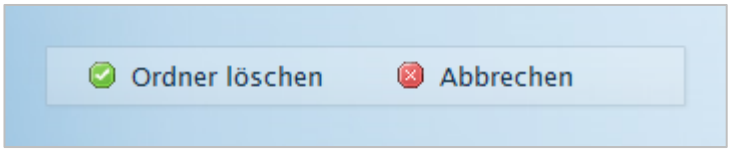


Abbildung 29: Werkzeugleiste zur Bestätigung eines Vorgangs

6 Schlussbetrachtung

6.1 Kritische Reflexion

Das Ziel der Arbeit war die Entwicklung eines Informationssystems, das einen schnellen und einfachen Zugriff auf die Dokumente und das Wissen des Studierendenparlaments realisiert. In funktionaler Hinsicht schafft es DAVe, dieses Ziel umzusetzen. Unabhängig von Ort, Zeit und verwendetem System hat der Anwender die Möglichkeit, auf das Wissen des Studierendenparlaments zuzugreifen und dieses durch die bereitgestellten Mitwirkungsmöglichkeiten zu erweitern. Die Volltextindizierung macht auch alte Dokumente nutzbar und formt zusammen mit den im System abgelegten Ideen und Anmerkungen in Form von Aufgaben und Kommentaren ein kollektives Gedächtnis für die Studierendenschaft, welches über Legislaturperioden hinweg Bestand hat.

Die Funktionen, die DAVe für die Organisation des Dokumentenbestands bereitstellt, wie die hierarchische Ablage in Ordnern, die Gruppierung in Akten sowie die Versionierung und Kennzeichnung von Dokumenten, ermöglichen darüber hinaus einen effektiveren und nachhaltigeren Umgang mit der wichtigsten Wissensressource des Parlaments. Sie gewährleisten nicht nur das schnelle Auffinden von Informationen, sondern unterstützen möglicherweise auch nachfolgende Generationen dabei, zurückliegende Vorgänge besser beziehungsweise mit geringerem Zeitaufwand nachvollziehen zu können.

Der Erfolg der Anwendung hängt jedoch maßgeblich von den zukünftigen Benutzern ab. Auch wenn DAVe alle funktionalen Ziele erfüllen kann, so bringt das System nur dann einen Mehrwert, wenn die Anwender es als Bestandteil ihrer Parlamentstätigkeit akzeptieren und es aktiv mit Inhalten füllen. Im Rahmen dieser Arbeit wurde mit der benutzerorientierten Oberflächengestaltung bereits versucht, die Hürden für die Nutzung von DAVe möglichst niedrig anzusetzen. Ein Erfolg ist damit jedoch noch nicht garantiert. Besonders in der kritischen Phase der Einführung des Systems ist es wichtig, die Parlamentsmitglieder von den Vorteilen der Anwendung zu überzeugen. Empfehlenswert ist es deshalb, wenn bei der Vorstellung von DAVe alle wichtigen Dokumente sowie ein Archiv der alten Sitzungsprotokolle im Vorfeld bereits in die

Anwendung eingepflegt worden sind und sie somit von Anfang an einen Nutzen für die Anwender hat.

Abschließend gilt es jedoch auch deutlich zu machen, dass DAVe kein Allheilmittel ist. Die Anwendung hat das Potenzial, die Informationssituation im Studierendenparlament – und möglicherweise auch in der Studierendenschaft allgemein – zu verbessern, alle Probleme wird aber auch sie nicht lösen können. Insbesondere auch deshalb nicht, da ihr Erfolg nachwievor vom Faktor Mensch abhängt.

6.2 Ausblick

Viele Internetanwendungen befinden sich in einem fortwährenden Prozess der Entwicklung und werden niemals in einer „finalen Version“ vorliegen. Besonders im Zusammenhang mit dem Web 2.0 ist dabei der Begriff „Perpetual Beta“ entstanden, der genau diesen Zustand der kontinuierlichen Weiterentwicklung verdeutlichen soll.

Vorausgesetzt, die Ergebnisse dieser Arbeit werden von den Mitgliedern der Studierendenschaft angenommen und es finden sich interessierte Studenten, die sich der Pflege und Weiterentwicklung des Systems widmen, wäre ein solcher Prozess auch für DAVe eine wünschenswerte Perspektive.

An Ideen und noch offenen Punkten mangelt es in jedem Fall nicht, denn aufgrund der begrenzten Bearbeitungszeit dieser Thesis kam es immer wieder zu Situationen, in denen ein Feature oder eine mögliche Verbesserung nicht realisiert werden konnte, da es einen Zeitplan einzuhalten galt und zunächst die grundlegende Funktionalität des Systems oberste Priorität hatte.

Nun existiert DAVe funktionstüchtig in seiner Version 1.0 und der Blick ist nach vorne auf mögliche Folgeversionen gerichtet.

Im Mittelpunkt sollte dabei auch weiterhin die Benutzerfreundlichkeit der Anwendung stehen. DAVe stellt für den Anwender ein Werkzeug dar, das ihn bei seiner Parlamentstätigkeit unterstützt. Durch zusätzliche Hilfestellungen besteht hier die Möglichkeit, die Verwendung des Systems noch einfacher und intuitiver zu gestalten. Ideen, die im Rahmen dieser Thesis nicht mehr umgesetzt werden konnten, sind beispielsweise kontextsensitive Hilfetexte sowie die Echtzeitvalidierung von Eingabemasken, welche frustrierende Fehlermeldungen nach dem Abschicken eines

Formulars vermeidet. Auch sollte eine Kontaktmöglichkeit eingerichtet werden, über die der Benutzer individuellen Support erhält.

Viel bisher ungenutztes Potenzial steckt auch in der Suchfunktionalität der Anwendung. Die Einrichtung einer erweiterten Suche, die den Benutzer dabei unterstützt, auch komplexe, feldbasierte Suchanfragen zu formulieren, wäre dank der von Lucene bereitgestellten Funktionen ein realistisches Ziel für eine folgende Systemversion. Insbesondere bei steigendem Dokumentenbestand ist eine differenziertere Suchmöglichkeit obligatorisch.

Schließlich wird die Zukunft auch zeigen, ob es sinnvoll ist, DAVe um spezielle Konzepte der Dokumentenverwaltung zu erweitern. Ein Thema hierbei wäre beispielweise die Implementierung eines Lebenszyklus‘ für Dokumente oder Dokumentversionen. So könnte darüber nachgedacht werden, ob es sinnvoll ist, für alle Dokumente sämtliche Bearbeitungsstände bereitzuhalten oder vielleicht Gründe dafür sprechen, eine geordnete Aussonderung alter Dokumente zu realisieren. Auch die Implementierung eines Workflow-Managements wäre denkbar, sollte die zu erwartende Nutzung dem Aufwand gerecht werden.

Nicht zuletzt wird der produktive Einsatz zeigen, wo für DAVe noch Verbesserungsbedarf besteht und welche Ideen dieses Konzepts tatsächlich den gewünschten Erfolg bringen.

Literaturverzeichnis

BATLEY 2007
Batley, Sue: *Information Architecture For Information Professionals*. Oxford : Chandos Publishing, 2007. – ISBN 1-84334-233-2

BERGMANN 2005
Bergmann, Sebastian: *Professionelle Softwareentwicklung mit PHP 5*. Heidelberg : dpunkt-Verlag, 2005. – ISBN 3-89864-229-1.

FRANZ 2008
Franz, Markus: PHP-Frameworks : Zend, eZ Components, Symfony und CakePHP [online]. In: *heise Developer*. – URL: http://www.heise.de/developer/artikel/↵ PHP-Frameworks-Zend-eZ-Components-Symfony-und-CakePHP-227084.html [zit. 2009-10-24]

HAW HAMBURG 2009
Web Server Statistics for www.haw-hamburg.de : *Monthly Reports* : *2009-09* [online]. – URL: http://www.haw-hamburg.de/accounting/reports/2009/2009-09.html [zit. 2009-10-24]

HEILMANN 2006
Heilmann, Christian: *Beginning JavaScript with DOM scripting and Ajax: from novice to professional*. New York : Apress, 2006. – ISBN 1-59059-680-3

HOEKMAN 2007
Hoekman, Robert: *Designing the Obvious*. Berkeley : New Riders, 2007. – ISBN 0-321-45345-X

HOLT 2008
Holt, Alex: jQuery and JavaScript Coding : Examples and Best Practices [online]. In: *Smashing Magazine*. – URL: http://www.smashingmagazine.com/2008/09/16/↵ jquery-examples-and-best-practices [zit. 2009-10-24]

JACOBSEN 2007
Jacobsen, Jens: *Website-Konzeption* : *Erfolgreiche Websites planen und umsetzen*. 4. aktualisierte u. erweiterte Aufl. München : Addison-Wesley, 2006. – ISBN 978-3-8273-2473-3

JQUERY
Traversing [online]. In: *jQuery Documentation* : *API Reference* – URL: http://docs.jquery.com/Traversing [zit. 2009-10-24]

KLEMPERT 2003
Klempert, Arne: *Jede Menge Bäume* : *Nested Sets* : *Verschachtelte Bäume mit MySQL* [online].– URL: http://www.klempert.de/nested_sets/artikel [zit. 2009-10-24]

LYNCH/HORTON 2008
Lynch, Patrick J. ; Horton, Sarah: *Web Style Guide* : *3rd Edition* : *Basic Design Principles for Creating Web Sites*. New Haven : Yale University Press, 2008. – ISBN 978-0-300-13737-8

LAZARIS 2009
Lazaris, Louis: CSS Differences in Internet Explorer 6, 7 and 8 [online]. In: *Smashing Magazine*. – URL: http://www.smashingmagazine.com/2009/10/14/↵ css-differences-in-internet-explorer-6-7-and-8 [zit. 2009-10-24]

NIELSEN 2001
Nielsen, Jakob: *Designing Web Usability* : *Erfolg des Einfachen*. 2., überarb. Aufl. München : Markt+Technik Verlag, 2001. – ISBN 3-8272-6206-2

PROPEL
NestedSet support in Propel [online]. In: *Propel 1.3 Documentation*. – URL: http://propel.phpdb.org/trac/wiki/Users/Documentation/1.3/Tree/NestedSet [zit. 2009-10-24]

TATE 2009
Tate, Tyler: Minimizing Complexity In User Interfaces [online]. In: *Smashing Magazine*. – URL: http://www.smashingmagazine.com/2009/10/07/↵ minimizing-complexity-in-user-interfaces [zit. 2009-10-24]

TIKA
Supported Document Formats. In: *Apache Tika.* – URL: http://lucene.apache.org/tika/formats.html [zit. 2009-10-24]

ZANINOTTO/POTENCIER 2009A
Zaninotto, Francois ; Potencier, Fabien: *The Definitive Guide to symfony* [online]. – URL: http://www.symfony-project.org/get/pdf/book-1.2-en.pdf [zit. 2009-10-24]

ZANINOTTO/POTENCIER 2009B
Zaninotto, Francois ; Potencier, Fabien: *Practical symfony : second edition* [online]. – URL: http://www.symfony-project.org/get/pdf/jobeet-1.2-propel-en.pdf [zit. 2009-10-24]

ZANINOTTO/POTENCIER 2009C
Zaninotto, Francois ; Potencier, Fabien: *symfony Forms in Action* [online]. – URL: http://www.symfony-project.org/get/pdf/forms-1.2-en.pdf [zit. 2009-10-24]

ZEND LUCENE
Zend_Search_Lucene. In: *Zend Framework* : *Programmer's Reference Guide.* – URL: http://framework.zend.com/manual/en/zend.search.lucene.html [zit. 2009-10-24]

Anhang A

Interne Bezeichnungen

DAVe verwendet in seinen Systemdateien ausschließlich englischsprachige Bezeichner. Die folgende Liste führt für alle konzeptionellen Begriffe die entsprechende interne Bezeichnung auf.

Konzeptionelle Bezeichnung	Interne Bezeichnung
Akte	document collection
Aufgabe	task
Aufgabenfilter	task filter
Benachrichtigung	notification
Benutzer	user
Benutzergruppe	user group
Dateicache	file cache
Dokument	document
Dokumentrechte	document rights
Dokumenttyp	document type
Dokumentversion	document version
Ereignis	event
Ereignisprotokoll	event list
Kommentar	note
Logeintrag	log entry
Logeintragstyp	log entry type
Ordner	folder
Ordnerrechte	folder rights

Anhang B

Datenbankschema

Da ein Großteil der Tabellenfelder des Datenbankschemas selbsterklärend ist, erfolgt an dieser Stelle lediglich eine mit Anmerkungen versehene Darstellung aller Datenbanktabellen.

document

Tabelle für Dokumente.

Feld	Typ	Beschreibung
`id`	`int(11)`	Primärschlüssel
`title`	`varchar(255)`	Dokumenttitel
`slug`	`varchar(255)`	Darstellung des Titels für URLs
`description`	`text`	Dokumentbeschreibung
`is_deleted`	`tinyint(4)`	Kennzeichnung als gelöscht
`created_at`	`datetime`	Datum der Erzeugung
`updated_at`	`datetime`	Datum der letzten Bearbeitung
`folder_id`	`int(11)`	Referenz auf den Ordner
`document_type_id`	`int(11)`	Referenz auf den Dokumenttyp
`document_collection_id`	`int(11)`	Referenz auf eine Akte
`user_id`	`int(11)`	Referenz auf den Benutzer

document_collection

Tabelle für Akten.

Feld	Typ	Beschreibung
`id`	`int(11)`	Primärschlüssel

Feld	Typ	Beschreibung
title	varchar(100)	Aktentitel
color	varchar(7)	Aktenfarbe

document_right

Tabelle für Dokumentrechte.

Feld	Typ	Beschreibung
document_id	int(11)	Referenz auf das Dokument
user_group_id	int(11)	Referenz auf Benutzergruppe
read	tinyint(4)	Leserecht
delete	tinyint(4)	Löschrecht
edit	tinyint(4)	Bearbeitungsrecht

document_type

Tabelle für Dokumenttypen.

Feld	Typ	Beschreibung
id	int(11)	Primärschlüssel
name	varchar(30)	Name des Dokumenttyps

document_version

Tabelle für Dokumentversionen.

Feld	Typ	Beschreibung
id	int(11)	Primärschlüssel
comment	text	Anmerkungen zur Version
status	varchar(15)	Versionsstatus (Entwurf oder Freigegeben)
version	varchar(10)	Versionsnummer
filename	varchar(100)	Dateiname des Schriftstücks

Feld	Typ	Beschreibung
mime_type	varchar(60)	MIME-Type der hochgeladenen Datei
is_indexed	tinyint(4)	Flag für die Indizierung
is_public	tinyint(4)	Kennzeichnung als öffentlich
created_at	datetime	Datum der Erzeugung
document_id	int(11)	Referenz auf das Dokument
user_id	int(11)	Referenz auf den Benutzer

file_cache

Tabelle für den Dateicache.

Feld	Typ	Beschreibung
checksum	varchar(40)	Prüfsumme der Datei, dient als Primärschlüssel
metadata	text	Ergebnis der Metadatenextraktion
content_html	text	Ergebnis der Inhaltsextraktion (HTML)
content_text	text	Ergebnis der Inhaltsextraktion (Text)
content_xhtml	text	Ergebnis der Inhaltsextraktion (XHTML)

folder

Tabelle für Ordner.

Feld	Typ	Beschreibung
id	int(11)	Primärschlüssel
name	varchar(40)	Ordnername
slug	varchar(40)	Darstellung des Namen für URLs
description	text	Ordnerbeschreibung
sorting_field	varchar(45)	Dokumentfeld, nach dem sortiert wird
sorting_order	varchar(4)	Reihenfolge, in der sortiert wird (aufsteigend oder absteigend)

Feld	Typ	Beschreibung
tree_left	int(11)	Nested-Set-Feld für Hierarchie
tree_right	int(11)	Nested-Set-Feld für Hierarchie
created_at	datetime	Datum der Erzeugung
updated_at	datetime	Datum der letzten Bearbeitung

folder_right

Tabelle für Ordnerrechte.

Feld	Typ	Beschreibung
document_id	int(11)	Referenz auf den Ordner
user_group_id	int(11)	Referenz auf Benutzergruppe
read	tinyint(4)	Leserecht
create	tinyint(4)	Erzeugungsrecht

log

Tabelle für Logeinträge.

Feld	Typ	Beschreibung
id	int(11)	Primärschlüssel
created_at	datetime	Datum der Erzeugung
parameters	varchar(255)	Parameter für die Vorgangsbeschreibung
rights	varchar(255)	Logrechte in Form eines serialisierten Arrays
log_entry_type_id	int(11)	Referenz auf den Logeintragstyp
folder_id	int(11)	Referenz auf einen Ordner
folder_name	varchar(40)	Der Ordnername
document_id	int(11)	Referenz auf ein Dokument
document_title	varchar(255)	Der Dokumenttitel
document_version_id	int(11)	Referenz auf eine Dokumentversion
document_version_↵	varchar(20)	Die Versionsnummer der Dokumentversion

Feld	Typ	Beschreibung
identifier		
note_id	int(11)	Referenz auf einen Kommentar bzw. eine Aufgabe
user_id	int(11)	Referenz auf den Benutzer
user_name	varchar(52)	Der Name des Benutzers

log_entry_type

Tabelle für Logeintragstypen.

Feld	Typ	Beschreibung
id	int(11)	Primärschlüssel
model	varchar(45)	Klasse des referenzierten Objekts
action	varchar(30)	Aktion, die mit dem referenzierten Objekt durchgeführt wurde
object_text	text	Textvorlage für Logeinträge mit Referenz auf das Objekt
standalone_text	text	Textvorlage für Logeinträge ohne Referenz auf das Objekt
mail_template_id	int(11)	Referenz auf die E-Mail-Vorlage

mail_template

Tabelle für E-Mail-Vorlagen.

Feld	Typ	Beschreibung
id	int(11)	Primärschlüssel
name	varchar(45)	Vorlagenname
subject	varchar(255)	E-Mail-Betreff
text	text	E-Mail-Textvorlage

note

Tabelle für Kommentare und Aufgaben.

Feld	Typ	Beschreibung
id	int(11)	Primärschlüssel
text	text	Kommentar-/Aufgabentext
refers_to	varchar(100)	Aufgabenfreitextbezug
is_task	tinyint(4)	Kennzeichnung als Aufgabe
is_completed	tinyint(4)	Kennzeichnung als erledigt
scope	int(11)	Nested-Set-Feld für Hierarchie
tree_left	int(11)	Nested-Set-Feld für Hierarchie
tree_right	int(11)	Nested-Set-Feld für Hierarchie
created_at	datetime	Datum der Erzeugung
due_at	datetime	Datum der Fälligkeit (Aufgabe)
user_id	int(11)	Referenz auf den Benutzer
completed_at	datetime	Datum der Erledigung (Aufgabe)
completed_by	int(11)	Referenz auf den Benutzer, der die Aufgabe als erledigt gekennzeichnet hat
document_id	int(11)	Referenz auf das Dokument
document_version_id	int(11)	Referenz auf eine Dokumentversion

task_filter

Tabelle für Aufgabenfilter.

Feld	Typ	Beschreibung
id	int(11)	Primärschlüssel
name	varchar(100)	Filtername
task_type	varchar(8)	Aufgabentyp (Alle, Dokumentaufgaben oder Allgemeine Aufgaben)
sorting_field	varchar(45)	Feld für die Sortierung
sorting_order	varchar(4)	Reihenfolge der Sortierung (aufsteigend oder absteigend)
uncompleted_first	tinyint(4)	Nicht erledigte Aufgaben zuerst
is_completed	tinyint(4)	Kennzeichnung ob Aufgabe erledigt ist

Feld	Typ	Beschreibung
include_subfolders	tinyint(4)	Kennzeichnung ob bei Filterung nach Ordner auch Unterordner mit einbezogen werden
created_at_from	datetime	Datum der Erzeugung (von)
created_at_to	datetime	Datum der Erzeugung (bis)
due_at_from	datetime	Datum der Fälligkeit (von)
due_at_to	datetime	Datum der Fälligkeit (bis)
completed_at_from	datetime	Datum der Erledigung (von)
completed_at_to	datetime	Datum der Erledigung (bis)
created_by	int(11)	Referenz auf den Benutzer, der die Aufgabe als erzeugt hat
completed_by	int(11)	Referenz auf den Benutzer, der die Aufgabe als erledigt gekennzeichnet hat
folder_id	int(11)	Referenz auf einen Ordner
document_id	int(11)	Referenz auf ein Dokument
user_id	int(11)	Referenz auf den Benutzer (wenn null, dann öffentlicher Filter)

user

Tabelle für Benutzer.

Feld	Typ	Beschreibung
id	int(11)	Primärschlüssel
username	varchar(52)	Benutzername
password	varchar(45)	Kennwort
remember_key	varchar(32)	Key für den Autologin
first_name	varchar(25)	Vorname
last_name	varchar(25)	Nachname
email	varchar(60)	E-Mail-Adresse
department	varchar(50)	Studiendepartment
course	varchar(50)	Studiengang
is_admin	tinyint(4)	Kennzeichnung als Administrator

Feld	Typ	Beschreibung
is_deactivated	tinyint(4)	Kennzeichnung als deaktiviert
last_login	datetime	Datum des letzten Logins
valid_until	datetime	Ende der Gültigkeit des Benutzers
created_at	datetime	Datum der Erzeugung
user_group_id	int(11)	Referenz auf Benutzergruppe
notification_document_new	tinyint(4)	Benachrichtigung über neue Dokumente
notification_document_version_new	tinyint(4)	Benachrichtigung über neue Dokumentversionen
notification_note_new	tinyint(4)	Benachrichtigung über neue Kommentare
notification_task_new	tinyint(4)	Benachrichtigung über neue Aufgaben
notification_task_completed	tinyint(4)	Benachrichtigung über erledigte Aufgaben

user_group

Tabelle für Benutzergruppen.

Feld	Typ	Beschreibung
id	int(11)	Primärschlüssel
name	varchar(30)	Benutzergruppenname

Anhang C

Verzeichnisübersicht

Die folgende Tabelle enthält eine Übersicht über alle wichtigen Verzeichnisse der Anwendung. Für weitere Informationen zur Verzeichnisstruktur von Symfony, siehe ZANINOTTO/POTENCIER 2009A, S. 29ff.

Verzeichnis/Datei	Anmerkungen
apps	Anwendungen des Symfony-Projekts (s. ebd., S. 40ff.)
frontend	
config	Anwendungskonfiguration (s. ebd., S. 61ff.)
i18n	Dateien für die Mehrsprachigkeit (nicht genutzt)
lib	Klassen der Anwendung
filter	Filter der Anwendung (s. ebd., S. 93ff.)
modules	Module der Anwendung
administration	Administrationsbereich
actions	Die Actions des Moduls (s. ebd., S. 50 u. S. 76ff.)
config	Die Konfiguration des Moduls (s. ebd., S. 63f.)
templates	Die Designtemplates für die Actions des Moduls (s. ebd., S. 51 u. S. 100ff.)
ajax	Enthält Actions die AJAX-Anfragen verarbeiten
authentication	Login-Bereich
default	Allgemeine Fehlermeldungen
document	Dokumentenbereich
global	Globale Components (s. ebd., S. 107ff.)
meta	Meta-Seiten (Impressum)
overview	Übersichtsbereich
rss	RSS-Feed
search	Suchansicht
task	Aufgabenbereich

Verzeichnis/Datei	Anmerkungen
user	Benutzereinstellungen
templates	Globale Designtemplates, Page Layout (s. ebd., S. 103ff.)
cache	Symfony Cache (s. ebd., S. 217ff.)
config	Symfony Projektkonfiguration (s. ebd., S. 61)
data	Datenverzeichnis
documents	Hochgeladene Dateien der Dokumentversionen
fixtures	
index	Lucene Indizes
rights	
sql	Datenbankschema (s. ebd., S. 127f.)
temp	Temporäres Verzeichnis für hochgeladene Dokumentversionen
doc	Klassendokumentation
lib	Projektklassen
filter	
form	Formularklassen (s. ZANINOTTO/POTENCIER 2009C, S. 8ff. u. S. 55ff.)
model	Propel-Modellklassen (s. ZANINOTTO/POTENCIER 2009A, S. 129f.)
task	
vendor	Klassenfamilien, externe Bibliotheken
apache	Apache Tika Bibliothek
dave	Für DAVe entwickelte Klassen
event	dvEvent-Klassenfamilie
helper	Helper-Funktionen
validator	Für DAVe entwickelte Formularvalidatoren
symfony	Symfony Klassen
Zend	Zend Framework Klassen (`Zend_Mail` und `Zend_Lucene`)
log	Symfony Logs (s. ebd., S. 298ff.)
plugins	Symfony Plugins (s. ebd., S. 331ff.)
test	
web	Document Root der Webanwendung (s. ebd., S. 31ff.)
css	Stylesheets
images	Grafiken
icons	Icons
js	JavaScripts

Anhang D

Browserstatistik

Quelle: HAW HAMBURG 2009, „Browser Summary"

This report lists the vendors of visitors' browsers.

Listing the top 20 browsers by the number of requests for pages, sorted by the number of requests for pages.

no.	reqs	pages	browser
1	**1676078**	**539608**	**Firefox**
	1596152	514103	Firefox/3
	66856	20994	Firefox/2
	12739	4358	Firefox/1
	212	108	Firefox/0
	18	10	Firefox/
	6	2	Firefox/8967016903
2	**1582008**	**354019**	**MSIE**
	763951	166143	MSIE/7
	500929	108898	MSIE/8
	313314	76690	MSIE/6
	3191	1815	MSIE/5
	487	414	MSIE/4
	106	30	MSIE/999
	25	25	MSIE/2
	4	4	MSIE/3
3	**142884**	**83506**	**Netscape (compatible)**
4	**228709**	**63534**	**Safari**
	93654	26509	Safari/531
	60341	14533	Safari/530
	28127	10003	Safari/525
	20601	6234	Safari/532
	14937	3492	Safari/528

no.	reqs	pages	browser
	4694	1669	Safari/523
	2268	586	Safari/413
	670	187	Safari/419
	517	148	Safari/312
	305	61	Safari/5530
5	**46375**	**40453**	**msnbot**
	32476	28179	msnbot/2
	13899	12274	msnbot/1
6	**32162**	**32162**	**check_http**
	32162	32162	check_http/v1991
7	**88488**	**22536**	**Opera**
	88079	22402	Opera/9
	247	84	Opera/8
	121	40	Opera/7
	37	6	Opera/10
	4	4	Opera/6
8	**63775**	**20970**	**Mozilla**
	59930	17968	Mozilla/1
	3	3	Mozilla/0
9	**16629**	**14251**	**Java**
	16629	14251	Java/1
10	**4838**	**3905**	**Netscape**
	4046	3691	Netscape/4
	744	195	Netscape/7
	30	7	Netscape/3
	2	2	Netscape/0
	2	2	Netscape/8
	1	1	Netscape/Nutch-1
11	**2359**	**2051**	**Clewwa-Bot**
	2359	2051	Clewwa-Bot/Nutch-1
12	**2840**	**1947**	**FAST Enterprise Crawler 6**
	2840	1947	FAST Enterprise Crawler 6 /
13	**2463**	**1890**	**NG**
	2126	1650	NG/2
	337	240	NG/4
14	**1969**	**1663**	**yacybot (amd64 Linux 2.6.28-15-generic; java 1.6.0_0;**

no.	reqs	pages	browser
			Europe
	1967	1661	yacybot (amd64 Linux 2.6.28-15-generic; java 1.6.0_0; Europe/en)
	2	2	yacybot (amd64 Linux 2.6.28-15-generic; java 1.6.0_0; Europe/de)
15	**1598**	**1598**	**SiteUptime.com**
16	**1644**	**1416**	**Xenu Link Sleuth 1.3c**
17	**1343**	**1332**	**Baiduspider+(+http:**
	1343	1332	Baiduspider+(+http://www
18	**796**	**732**	**Wget**
	796	732	Wget/1
19	**748**	**694**	**Eurobot**
	748	694	Eurobot/1
20	**592**	**592**	**Sosospider+(+http:**
	592	592	Sosospider+(+http://help
	17867	10828	[not listed: 284 browsers]

Anhang E

Information Audit

Der Information Audit als Grundlage für eine Informationsarchitektur

„Individual knowledge is a valuable asset in any organisation. One of the main drivers for developing and improving information systems is to allow for more effective knowledge sharing."
(BATLEY 2007, S. 33)

Einleitung

Sue Batley formuliert mit diesem Zitat aus Ihrem Buch „Information Architecture for Information Professionals" sehr treffend eines der Hauptziele, welches mit der Entwicklung von DAVe verfolgt wird. DAVe steht für Dokumenten- und Aufgabenverwaltung und stellt ein Informationssystem dar, welches das Studierendenparlament der HAW Hamburg in dem Vorhaben unterstützen soll, die Bereitstellung und gemeinsame Bearbeitung von Dokumenten zu erleichtern und den Austausch von Informationen zu fördern.

Die Realisierung von DAVe erfolgt im Rahmen der Bachelorthesis „DAVe – Entwicklung und Konzeption einer webbasierten Dokumenten- und Aufgabenverwaltung für das Studierendenparlament der HAW Hamburg". Wie der Titel schon erahnen lässt, erfolgt die Umsetzung in zwei Phasen: Die Konzeptions- und die Entwicklungsphase.

Die Konzeption von DAVe orientiert sich an dem bereits erwähnten Buch „Information Architecture For Information Professionals" von Sue Batley. Demnach ist das Ziel von Information Architecture die Entwicklung eines Informationssystems, das sinnvoll strukturiert, attraktiv und gut gepflegt ist und den Benutzern erlaubt, Informationen schnell und effizient abzurufen (vgl. BATLEY 2007, S. 14).

Im Rahmen der Konzeptionsphase soll deshalb eine Informationsarchitektur erarbeitet werden, die sich an den Bedürfnissen der Benutzer, also vorrangig an den Mitgliedern

des Studierendenparlaments orientiert und diese in den Mittelpunkt stellt. Hierfür findet eine vierteilige Analyse statt, die sich an den „four preliminary stages“ für das Design eines Informationssystems von Batley orientiert:

- Information Needs Analysis
- Task Analysis
- Resource Analysis
- User Modelling

(vgl. BATLEY 2007, S. 13ff.)

Die ersten drei Schritte lassen sich unter dem Begriff des Information Audits zusammenfassen (vgl. BATLEY 2007, S. 14), welcher im Rahmen dieser Fallstudie für das Projekt DAVe durchgeführt werden soll.

Hierfür soll zunächst allgemein das Konzept des Information Audits vorgestellt werden: Wofür eignet sich dieser Prozess und wie sehen mögliche Vorgehensweisen aus? Anschließend wird das Konzept auf den konkreten Anwendungsfall übertragen und spezifiziert.

In den folgenden Abschnitten wird der Information Audit durchgeführt. Hierbei werden die verwendeten Methoden erläutert sowie die Ergebnisse der einzelnen Arbeitsschritte präsentiert. Das Fazit beleuchtet abschließend den Nutzen des Audits im konkreten Anwendungsfall.

Der Information Audit

Für den Information Audit gibt es zahlreiche Definitionen, jedoch keine universell akzeptierte. Vor allem an der Zielsetzung, die mit dem Audit verfolgt wird, scheiden sich die Geister.

Eine in Großbritannien häufig verwendete Definition ist die von Elizabeth Orna: “A systematic evaluation of information use, resources and flows, with a verification by reference to both people and existing documents, in order to establish the extent to which they are contributing to an organization's objectives.” (n. HENCZEL 2007, S. XXII).

Etwas greifbarer hat es die auf Informations- und Wissensmanagement spezialisierte Beratungsfirma TFPL formuliert: „A systematic process through which an organisation can understand its knowledge and information needs, what it knows, the information flows and gaps.“ (n. BATLEY 2007, S. 14).

Die Erkenntnisse, die mit einem Information Audit gewonnen werden sollen, decken sich in den verschiedenen Definitionen weitestgehend. Batley formuliert sie folgendermaßen:

- Die Informationsbedürfnisse der ganzen Organisation, einzelner Abteilungen und des einzelnen Mitarbeiters
- Informationen, die in der Organisation erstellt werden, und der Wert dieser Informationen
- In der Organisation vorhandene Fachkompetenzen und Wissenskapital
- Informationslücken
- Externe Informationsressourcen und wie diese effektiver genutzt werden können
- Interne Informationsressourcen, deren Wert und bessere Nutzung
- Informationsflüsse

(vgl. BATLEY 2007, S. 16f.)

Auch wenn sich die Erkenntnisse ähneln, können unterschiedliche Zielsetzung mit dem Information Audit verfolgt werden. Henczel bspw. konzentriert sich auf die Erarbeitung einer Informationsrichtlinie (Information Policy) für die betreffende Organisation, welche den Umgang mit Information regelt. Hier kommt der Information Audit als Werkzeug für Knowledge Management in einem Unternehmen zum Tragen (vgl. HENCZEL 2007, S. 11ff.).

Batley verwendet den Information Audit im Kontext der Informationsarchitektur und zitiert hierfür ebenfalls TFPL: „Resulting from an information audit is an 'information map' which can be used as the basis for designing the content of intranets, as well as for the foundation of a corporate information strategy or a knowledge management strategy.” (n. BATLEY 2007, S. 14). Hier wird neben der Informationsstrategie der Information Audit auch als Grundlage für die Entwicklung von Intranets genannt.

Jedoch geht Batley nicht ausführlicher darauf ein, wie ein Information Audit mit einer solchen Zielsetzung abzulaufen hat. Der folgende Information Audit nutzt deshalb das von Henczel entwickelte Seven-Stage Information Audit Model als Ausgangspunkt und passt dieses an die Bedürfnisse eines Information Audits als Grundlage für eine Informationsarchitektur an.

Das Seven-Stage Information Audit Model

Henczels beschränkt sich in ihrem Seven-Stage Modell nicht nur auf die Kernaufgaben bei der Durchführung eines Information Audits, sondern gibt auch Empfehlungen bei der Umsetzung des Audits in einem Unternehmen. Die Unterstützung des Audits durch die Unternehmensleitung ist bei ihr deshalb ebenso ein Thema wie die Kommunikation der Ergebnisse und die Implementierung der Empfehlungen in die Unternehmensstrategie.

Viele dieser Punkte sind in dem konkreten Anwendungsfall jedoch nicht relevant. Im Folgenden wird deshalb kurz das komplette Modell Henczels vorgestellt, um dieses anschließend dann für die Anforderungen dieses Projekts anzupassen.

Wie der Name schon deutlich macht, umfasst das Modell sieben Schritte für die Durchführung eines Information Audits.

1. Planning
2. Data Collection
3. Data Analysis
4. Data Evaluation
5. Communicating Recommendations
6. Implementing Recommendations
7. The Information Audit as a Continuum

(vgl. HENCZEL 2007, S. 16ff.)

Die Kernprozesse des Information Audits finden hierbei in den ersten vier Schritten statt. Hier werden Konzepte für das Vorgehen aufgestellt, die Daten gesammelt und ausgewertet sowie anschließend Evaluiert und Empfehlungen formuliert. Die Schritte fünf und sechs beschreiben den Weg zu einer Informationsstrategie für das Unternehmen und im siebten Schritt macht Henczel deutlich, dass der Information Audit idealerweise ein sich wiederholender Prozess ist, der fortwährend Informationssituation des Unternehmens analysiert und gegebenenfalls die Informationsstrategie anpasst.

Da im Falle von DAVe die erarbeiteten Empfehlungen nicht an ein Management kommuniziert werden müssen, sondern in die Konzeption des Systems einfließen, kann der Schritt fünf außer Acht gelassen werden. Ebenso entfällt für diese Fallstudie der Schritt sechs, da hier Implementierung der Empfehlungen die Konzeption und Entwicklung des Systems selbst betrifft und somit in den Bereich der Bachelorthesis fällt. Der abschließende Schritt und damit die Etablierung des Information Audits als regelmäßig stattfindende Maßnahme der Studierendenschaft ist gerade im Hinblick auf die Evaluation von DAVe interessant. So könnte analysiert werden, inwieweit die

Einführung des Systems tatsächlich die Informationssituation im Studierendenparlament verbessert hat und in welchen Bereichen noch Nachbesserungsbedarf besteht. Da zwischen dieser Fallstudie und der Einführung von DAVe jedoch die sehr zeitaufwändige Entwicklungsphase steht, kann die Etablierung eines regelmäßigen Audits nicht Teil dieser Arbeit sein. Ebenso ist fraglich, ob die erforderlichen Ressourcen zur Verfügung stehen, um eine solche Evaluation durchzuführen und inwieweit dies überhaupt im Verhältnis zum tatsächlichen Nutzen steht.

Somit beschränkt sich die Durchführung des Information Audits für die Konzeption von DAVe auf die folgenden vier Schritte:

1. Planning
2. Data Collection
3. Data Analysis
4. Data Evaluation

In Henczels Modell besteht jeder der sieben Schritte aus verschiedenen Teilaufgaben, die in den folgenden Abschnitten vorgestellt werden. Auch hier sind einige Teilaufgaben für die vorliegende Fallstudie nicht relevant, was an den entsprechenden Stellen näher erläutert wird.

Stage 1: Planning

Die erste Phase des Information Audits befasst sich mit der Planung des Vorhabens. Hierfür definiert Henczel fünf Aufgaben:

1. Develop Clear Objectives
2. Determine Scope and Resource Allocation
3. Choose a Methodology
4. Develop a Communication Strategy
5. Enlist Management Support

(vgl. HENCZEL 2007, S. 24)

Für die Fallstudie relevant sind vor allem die Festlegung von Zielen und Umfang sowie die zur Anwendung kommenden Methoden. Die Entwicklung einer Kommunikationsstrategie, in der es darum geht, die Mitarbeiter auf den bevorstehenden Information Audit vorzubereiten, ist im vorliegenden Fall nicht notwendig, da die Gruppe der Befragten überschaubar ist und der Sinn und Zweck des Audits einfach zu kommunizieren ist.

Zielsetzung

In einem ersten Schritt sollte überlegt werden, aus welchen Gründen der Information Audit durchgeführt wird und was man sich erhofft, mit ihm zu erreichen. Um eine Zielsetzung formulieren zu können, ist es jedoch zunächst wichtig, die Organisation, in welcher man den Audit durchführen will, zu verstehen.

Das Studierendenparlament (StuPa), für welches DAVe entwickelt wird, entscheidet weitestgehend in allen Angelegenheiten der Studierendenschaft. Es ist das höchste Organ und nimmt eine legislative Funktion innerhalb der Studierendenschaft ein. Zu den wichtigsten Aufgaben gehören die Wahl und Kontrolle des Allgemeinen Studierendenausschusses (AStA) und Beschlussfassungen über die Finanzen der Studierendenschaft sowie über Änderungen der Satzung und der in ihr vorgesehenen Ordnungen. Die Mitglieder des Studierendenparlaments werden jährlich von der Gesamtheit der Studenten gewählt, was erfahrungsgemäß eine hohe Personalfluktuation zur Folge hat.

Das Verhältnis von AStA und StuPa lässt sich grob mit dem Verhältnis von Bundestag und Bundesregierung bzw. der öffentlichen Verwaltung vergleichen. Das StuPa gibt in Form der Satzung der Studierendenschaft und der in ihr genannten Ordnungen den rechtlichen Rahmen für die Tätigkeiten des AStAs vor, welcher an die Beschlüsse des StuPas gebunden ist. Ebenso entscheidet das StuPa über die finanziellen Mittel, die dem AStA für seine Arbeit zur Verfügung stehen.

Wie die beiden Organe zueinander in Beziehung stehen und sich in die studentische Selbstverwaltung einordnen, veranschaulicht ***Anhang C: Schaubild «Gremien der HAW Hamburg***.

Im Gegensatz zum AStA, der wöchentliche Sitzungen abhält und während der Vorlesungszeiten täglich in den Geschäftsräumen der Studierendenschaft anzutreffen ist, tritt das StuPa in der Regel nur alle zwei Wochen zusammen. Zwischen den Sitzungen findet nur unregelmäßig ein Informationsaustausch via E-Mail oder in Form von Arbeitsgruppen statt.

DAVe soll den daraus resultierenden Folgen entgegenwirken. Die Mitglieder des StuPas sollen auch außerhalb der Sitzungen schnellen Zugriff auf Dokumente haben und dazu motiviert werden, Informationen auszutauschen. Der Information Audit soll die Konzeption von DAVe dahingehend unterstützen, die Bedürfnisse der zukünftigen Nutzer zu ermitteln und die Gegebenheiten, auf die bei der Entwicklung Rücksicht genommen werden muss, festzuhalten.

Die Zielsetzung dieses Information Audits ist es deshalb, allgemeine Anforderungen zu formulieren, die bei der Realisierung von DAVe beachtet werden müssen.

Umfang und Ressourcen

DAVe soll als Informationssystem die Arbeit des Studierendenparlaments unterstützen. Zwar ist auch vorgesehen, dass andere Organe und Gremien Zugriff auf die Plattform haben, im Mittelpunkt der Entwicklung stehen aber die Mitglieder des StuPas. Dementsprechend sollte der Information Audit dimensioniert werden. Relevant sind vor allem die Situation im Parlament, sowie die Schnittstellen nach außen. Die geringe Mitgliederzahl des StuPas ermöglicht zudem eine weitestgehend umfassende Befragung während einer Sitzung.

Methodik

In diesem Schritt wird festgelegt, mit welchen Methoden die erforderlichen Daten erhoben und ausgewertet werden.

Um dem von Batley angeführten Aspekt des „user-centred design" gerecht zu werden (s. BATLEY 2007, S. 9f.), ist die wichtigste Ressource bei der Erhebung der Daten die Befragung der Mitglieder. Diese wird in Form eines Fragebogens mit qualitativen und quantitativen Fragen stattfinden. Weitere Informationen werden über ein Gruppeninterview während der Sitzung und formlose Gespräche mit einzelnen StuPa-Mitgliedern sowie mit dem AStA-Vorstand gesammelt. Die Befragungen sollen vor allem Erkenntnisse über die Arbeitsweise des Parlaments hervorbringen und Auskunft darüber geben, wo Informationsengpässe und Unzufriedenheit herrschen.

Der zweite Teil der Datenerhebung konzentriert sich auf die vorhandenen Dokumente und Informationen. Geklärt wird hierbei, welche Dokumenttypen im StuPa existieren und wie mit diesen gearbeitet wird. Hierfür wird eine Stichprobe der sich im Umlauf befindlichen Dokumente erstellt.

Im letzten Schritt erfolgt eine Sichtung der rechtlichen Grundlagen, die der Arbeit des Studierendenparlaments den Rahmen geben.

Die Auswertung kann aufgrund des geringen Umfangs manuell oder mithilfe einfacher Tabellenkalkulation erfolgen. Spezielle Analysewerkzeuge sind nicht erforderlich.

Stage 2: Data Collection

Für die zweite Phase des Information Audits empfiehlt Henczel zunächst den Aufbau einer Information Resources Database. Darauf wird in diesem Information Audit jedoch verzichtet, da der zu erwartende Umfang der Ergebnisse einem solchen Aufwand nicht gerecht würde.

Die Datenerhebung besteht deshalb aus zwei Schritten: Die Vorbereitung und die Durchführung.

In der Vorbereitung wird in einem ersten Schritt geklärt, was man mit der Erhebung in Erfahrung bringen will. Für DAVe relevant sind vor allem folgende Fragen:

- Welche Dokumente und Informationen existieren überhaupt?
- Wie fließen diese Informationen im StuPa und zwischen StuPa und AStA?
- Haben die Mitglieder Zugriff auf alle Dokumente und Informationen, die sie für Ihre Arbeit benötigen?

Diese Informationen werden über eine Reihe von quantitativen und qualitativen Fragen ermittelt. Der quantitative Teil der Befragung soll vor allem dazu dienen, klare Tendenzen feststellen zu können, wohingegen der qualitative Teil besonders darauf zielt, den Befragten Informationen zu entlocken, die zuvor nicht in Erwägung gezogen wurde.

Der daraus resultierende Fragebogen befindet sich in ***Anhang A: Fragebogen***.

In den anschließenden formlosen Gesprächen soll den Befragten noch einmal die Idee von DAVe präsentiert werden, um anschließend Anregungen und Fragen aufzunehmen. Diese werden aufgezeichnet und fließen ebenfalls in die Datenerhebung mit ein.

Der dritte Teil der Erhebung, die Erfassung von Parlamentsdokumenten, erstreckt sich über mehrere Wochen. Hierbei werden aktuell im Umlauf befindliche Dokumente gesammelt sowie eine Stichprobe von Dokumenten aus privaten Unterlagen zusammengestellt. In dieser Dokumentensammlung enthalten sind auch alle Rechtsgrundlagen, die die Arbeit der Studierendenschaft betreffen.

Nachdem die Vorbereitung abgeschlossen ist, wird mit der Datenerhebung begonnen. Die Ergebnisse werden in der Phase der Datenanalyse vorgestellt.

Stage 3: Data Analysis

Die Phase der Datenanalyse sorgt zunächst dafür, dass die Ergebnisse aus der Datenerhebung digitalisiert und in ein evaluierbares Format gebracht werden.

Dokumenttypen

Die Dokumentenverwaltung ist ein integraler Bestandteil von DAVe. Deshalb ist es besonders wichtig zu wissen, für welche Arten von Dokumenten DAVe konzipiert werden muss. Die folgende Darstellung umfasst alle relevanten Typen, die in der Arbeit des Studierendenparlaments eine Rolle spielen. Eine vergrößerte Version befindet sich in ***Anhang B:***

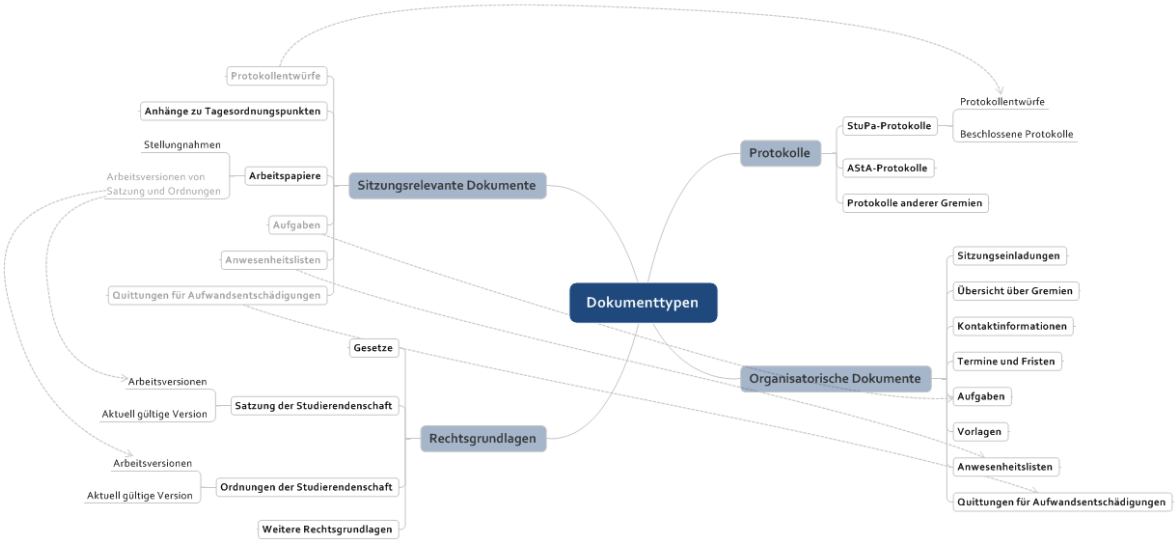


Abbildung 1: Dokumenttypen

Verfügbarkeit von Dokumenten und Informationen

Der vorangegangene Abschnitt hat dargestellt, welche Arten von Dokumenten im Studierendenparlament anfallen. Das bedeutet jedoch nicht, dass diese auch ohne Weiteres zugänglich sind. Der leichte und schnelle Zugriff auf Dokumente soll ein wesentliches Merkmal von DAVe sein. Die Frage nach Informationen, die nicht zur Verfügung stehen, aber bei der Arbeit im StuPa helfen würden, hat ergeben, dass vor allem bei folgenden Informationsarten ein Engpass besteht.

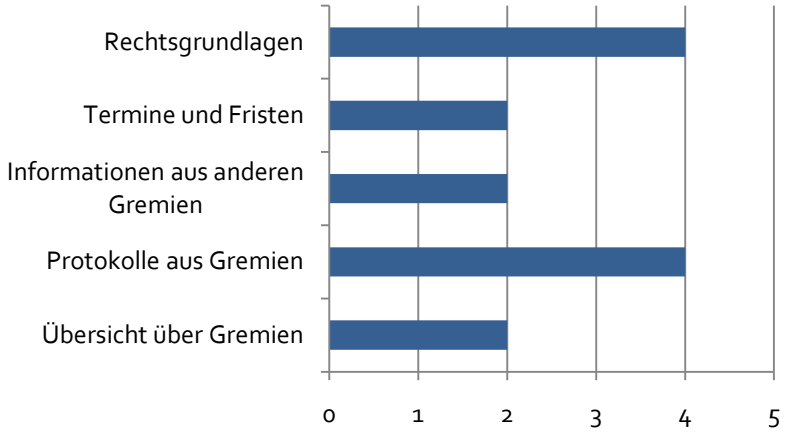


Abbildung 2: *„Welche Informationen und Dokumente, die dir momentan nicht zu Verfügung stehen, würden die bei der Arbeit für das StuPa helfen?“* (normalisiert, nur Mehrfachnennungen)

Der Schwerpunkt auf Informationen aus anderen Gremien, insbesondere Protokollen wird durch den Fragebogen an anderer Stelle noch bestätigt.

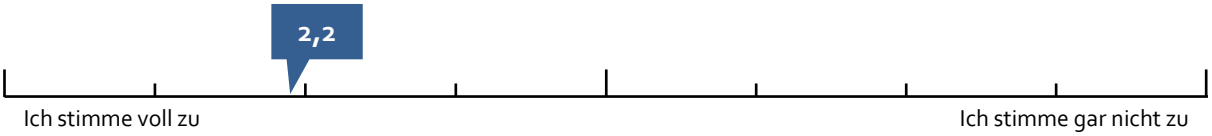


Abbildung 3: *„Protokolle aus anderen Gremien helfen mir bei der Arbeit im Studierendenparlament.“*

Auch allgemein wird der Zugriff auf Dokumente im Fragebogen eher durchschnittlich bewertet.

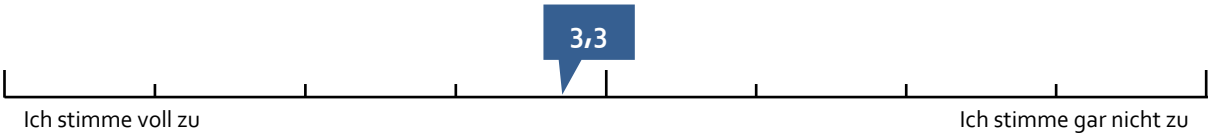


Abbildung 4: *„Wenn ich Dokumente bzw. Informationen benötige oder sie mich interessieren, habe ich ohne großen Aufwand Zugriff auf diese.“*

Die Verfügbarkeit von aktuellen Versionen der Satzung und Ordnungen wird im Fragebogen hingegen positiv bewertet. Dies ist vor allem deshalb überraschend, da im Gruppengespräch und informellen Einzelgesprächen oftmals die Problematik genannt wurde, dass nicht eindeutig war, welche Fassung der Satzung oder einer Ordnung die aktuell gültige ist.

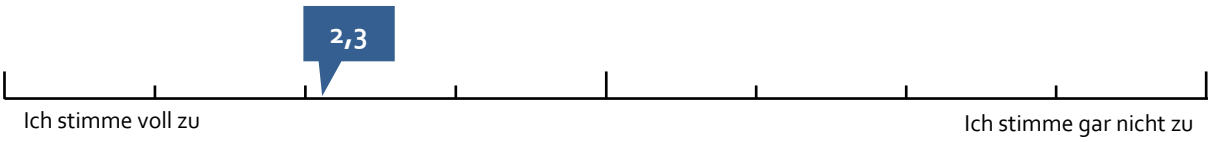
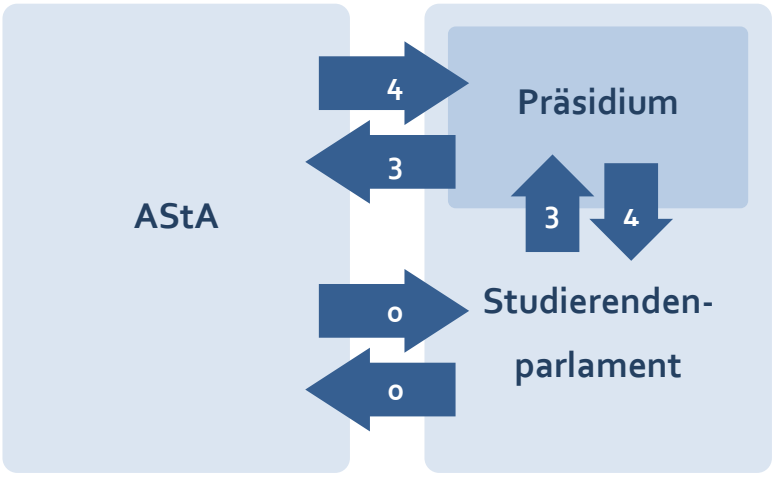


Abbildung 5: *„Aktuelle Versionen von Satzungen und Ordnungen stehen mir zur Verfügung.“*

Ein weiterer Aspekt, der in dem Gruppengespräch deutlich zur Aussprache kam, ist der Informationsverlust, der beim Wechsel der Legislaturperiode entsteht. Nur zirka ein Fünftel der Mitglieder war bereits in der vorangegangenen Legislaturperiode im Studierendenparlament aktiv. Das führt dazu, dass der Informationsfluss von einer Periode zur nächsten hauptsächlich von der Übergabe des alten Präsidiums ans neue sowie den wenigen wiedergewählten Mitgliedern abhängt. Dies führt dazu, dass viele noch ausstehende Aufgaben und Informationen mit dem Wechsel verloren gehen.

Informationsflüsse

Bei der Bewertung des Informationsflusses zwischen den Organen StuPa und AStA sowie zwischen den regulären Mitgliedern des Parlaments und dem Präsidium, zeigt sich, dass besonders zwischen AStA und regulären StuPa-Mitgliedern nicht genügend Informationsaustausch stattfindet.



Bewertungsmöglichkeiten

+	Gut
O	Befriedigend/Ausreichend
–	Schlecht

Die Darstellung links errechnet sich aus der Verwendung von +1, 0 und -1 für die obenstehenden Bewertungs-möglichkeiten

Abbildung 6: Bewertung der Informationsflüsse

Arbeitsabläufe

Die am häufigsten vorkommenden Arbeitsabläufe im Zusammenhang mit Dokumenten sind die Bearbeitung von Arbeitspapieren sowie die Erstellung von Protokollen.

Arbeitspapiere

Befindet sich die Satzung der Studierendenschaft oder eine ihrer Ordnungen in einer Überarbeitung, so wird in der Regel eine Arbeitsgruppe gebildet, die das Dokument unabhängig voneinander überarbeitet. Im Idealfall finden dabei auch Änderungswünsche Berücksichtigung, die in der Zeit seit der letzten Überarbeitung in den Sitzungen angefallen sind. Anschließend werden die Änderungen zu einem ersten Entwurf zusammengetragen und in einer Sitzung vorgestellt.

Bis zu diesem Zeitpunkt geht eine Arbeitsgruppe in der Regel sehr eigenständig vor, weshalb sich die übrigen Parlamentsmitglieder in den Bearbeitungsprozess erst einbringen können, wenn die erste Lesung erfolgt. Dies spiegelt sich auch in der Befragung wider.

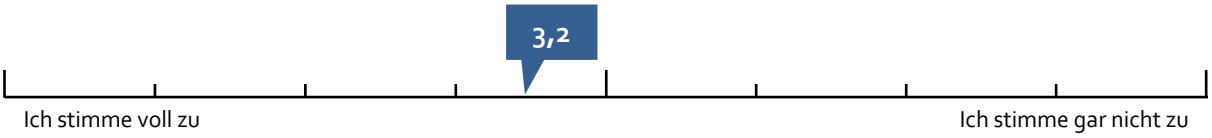


Abbildung 7: ***„Auch außerhalb der Sitzungen habe ich die Möglichkeit, mich in die Arbeit an Dokumenten einzubringen (auch wenn ich nicht Mitglied einer Arbeitsgruppe bin).“***

Auch in den Gesprächen mit Parlamentsmitgliedern wurde deutlich, dass sich einige eine stärkere Einbindung in die Bearbeitung von Dokumenten wünschen, jedoch nicht die zeitlichen Ressourcen haben, um sich als vollwertiger Teilnehmer einer Arbeitsgruppe einbringen zu können.

Ein weiteres Problem ist die Tatsache, dass die Mitglieder oftmals erst während der Sitzung einen Arbeitsentwurf zum ersten Mal zu Gesicht bekommen und sich somit nur ungenügend auf die Lesung vorbereiten können. Dabei besteht durchaus die Bereitschaft, sich im Vorfeld auf Sitzungen vorzubereiten, stünden die Dokumente rechtzeitig zur Verfügung.

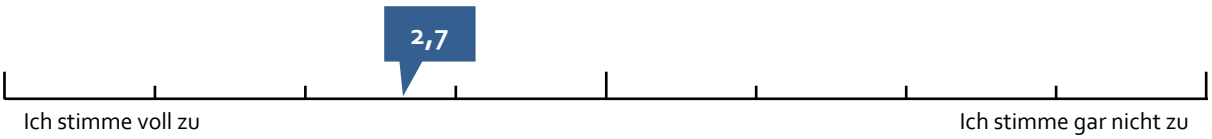


Abbildung 8: ***„Ich würde mich durch das Lesen von Dokumenten auf Sitzungen vorbereiten, stünden Sie rechtzeitig mir vor der Sitzung auf einer Online-Plattform zur Verfügung.“***

So können die StuPa-Mitglieder erst in der ersten Lesung Ihre Anmerkungen zum Entwurf einbringen, welche dann zur zweiten Lesung in der nachfolgenden Sitzung eingearbeitet werden. Stimmt das StuPa in der zweiten Lesung für den Entwurf, wird dieser an die Hochschulverwaltung weitergereicht, die ihrerseits die Möglichkeit hat,

Änderungswünsche anzubringen. In einer letzten Sitzung werden diese dann in den Entwurf eingearbeitet und das Parlament stimmt über das Dokument ab. Dieser Prozess kann sich besonders aufgrund von Verzögerungen in der Hochschulverwaltung über Monate hinweg ziehen und nicht selten bis in eine neue Amtszeit des StuPas hereinreichen. Erst nach Abschluss dieses Prozesses ist eine Satzung oder Ordnung gültig.

Sitzungsprotokolle

Der Arbeitsablauf bei den Sitzungsprotokollen gestaltet sich vergleichsweise einfach. Sie werden in der Regel ein bis zwei Tage nach einer Sitzung an die StuPa-Mitglieder per E-Mail geschickt. Diese haben dann bis zur nächsten Sitzung die Möglichkeit, Korrekturen einzuschicken. In der darauffolgenden Sitzung findet dann eine gemeinsame Lesung des Dokuments statt, nach welcher über das Protokoll abgestimmt wird. Oftmals werden aber auch umfangreiche Änderungsanträge erst während der Sitzung eingereicht, was zu Verzögerungen im Sitzungsablauf führt.

Stage 4: Data Evaluation

Nachdem alle relevanten Daten gesammelt wurden, gilt es nun, diese Ergebnisse zu bewerten und anschließend zu interpretieren. Die daraus gewonnen Erkenntnisse sollen helfen, konkrete Empfehlungen für die Konzeption von DAVe zu entwickeln.

Bewertung

Die größte Problematik des Studierendenparlaments ist der schnelle Zugriff auf Informationen. Durch die dezentrale Struktur des Organs und die Bereitstellung via E-Mail hängt die Verfügbarkeit von Dokumenten stark von der Ordnungsliebe der einzelnen Mitglieder ab. Oftmals ist jedoch so, dass die benötigten Informationen zwar per E-Mail kommuniziert werden oder in Papierform während der Sitzung herumgereicht werden, diese jedoch schnell im Posteingang oder der eigenen Zettelwirtschaft verloren gehen und nicht zur schnellen Verfügung bereitstehen, sollten sie einmal wieder benötigt werden.

Besonders bemerkenswert sind hierbei die Antworten auf die Frage nach Informationen, die nicht zur Verfügung stehen, aber hilfreich für die Arbeit im Studierendenparlament wären. Alle genannten Informationen sind in der Erhebung der Dokumenttypen vorhanden, stehen aber offensichtlich nicht zur Verfügung. Um Zugriff auf die Protokolle oder andere Dokumente des AStAs zu bekommen, müsste ein Mitglied des StuPas

beispielsweise die Räumlichkeiten der Studierendenschaft aufsuchen. Sogar für die Recherche in den Parlamentsprotokollen wäre dies erforderlich, sofern man nicht den eigenen Posteingang durchsuchen will oder kann. Oder man fordert Sie auf elektronischem Wege an, wobei man allerdings auch mit bis zu zwei Tagen Beantwortungszeit rechnen muss.

Ähnliches gilt für die Satzung und die Ordnungen, die wichtigsten Arbeitsmittel des Studierendenparlaments. Oftmals besteht Unklarheit darüber, welche der vielen Versionen, die mit der Zeit per E-Mail verschickt oder in den Sitzungen verteilt wurden, die aktuell gültige ist bzw. welche der aktuellste Arbeitsentwurf ist. Nicht selten muss diese dann beim AStA oder StuPa-Präsidium erfragt werden – ein schneller Zugriff ist also nicht möglich.

Bedenklich ist auch die schlechte Bewertung des Informationsflusses zwischen den regulären StuPa-Mitgliedern und dem AStA. Im Gegensatz zum Präsidium, welches häufig an den Sitzungen des AStA teilnimmt, fehlt den regulären Mitgliedern oft die Zeit, um an den wöchentlich stattfindenden Treffen des Studierendenausschusses teilzunehmen. Sie müssen sich also auf die Informationen verlassen, die sie über das Präsidium erreichen und werden somit auch häufig erst mit zwei Wochen Verzug über Entwicklungen im Tagesgeschäft der Studierendenschaft informiert. Der Wunsch nach Protokollen aus anderen Gremien zeigt, dass hier ein Bedürfnis der StuPa-Mitglieder besteht, zeitnäher und umfassender informiert zu werden und besonders im Hinblick auf ihre Kontrollfunktion, die sie dem AStA gegenüber auszuüben haben, ist dies auch ein wichtiger Aspekt, den es zu beachten gilt.

Generell ist im Hinblick auf die Verfügbarkeit von Dokumenten und Informationen zu sagen, dass diese zwar existieren, der Zugriff auf diese aber mit Hürden verbunden ist und somit Verbesserungspotenzial besteht.

Aber auch bei den Arbeitsabläufen existieren Optimierungsmöglichkeiten. Die mangelnde Transparenz bei der Bearbeitung von Dokumenten in Arbeitsgruppen resultiert in einer schlechteren Kontrolle der Arbeitsergebnisse und oftmals auch zu einem Mehraufwand, wenn umfangreiche Änderungswünsche nachträglich eingepflegt werden müssen.

Ein weiteres Problem ist der häufig auftretende Verlust Informationen, die die Bearbeitung von Dokumenten betreffen, beispielsweise notwendige Änderungen, die bei der nächsten Überarbeitung vorgenommen werden sollen, oder Informationen darüber, in welchen Punkten sich eine Arbeitsversion von der anderen unterscheidet.

Empfehlungen

Das größte Potenzial zur Verbesserung der Informationssituation hat DAVe bei der Umsetzung eines schnellen Zugriffs auf Dokumente und Informationen. Um hier eine spürbare Verbesserung zu schaffen, sollte der Zugang vor allem einfach und unkompliziert sowie möglichst intuitiv sein, damit für neue Benutzer keine Hürden bei der Nutzung der wichtigsten Funktion des Systems entstehen.

Da die Parlamentsmitglieder es gewohnt sind, Ihre Dokumente per E-Mail zu erhalten, sollte auch bei DAVe das Push-Prinzip Anwendung finden. Neue Protokolle und Arbeitsversionen sollten Sie auch weiterhin ohne eigenes Zutun erreichen, jedoch mit dem Unterschied, dass diese dann nicht im Posteingang verschwinden, sondern über die Webplattform jederzeit griffbereit sind und vor allem auch schnell gefunden werden. Wenn auch in DAVe erst langwierig nach dem gewünschten Dokument gesucht werden muss, wird das System seiner Aufgabe nicht gerecht.

Ein weiterer wichtiger Punkt ist weniger für die Konzeption des Systems als vielmehr für die tatsächliche Verwendung relevant. Damit das System von den Benutzern akzeptiert wird, muss es von Anfang an deren Informationsbedürfnisse befriedigen können. Das bedeutet, DAVe sollte nicht mit einer leeren Ordnerstruktur starten, sondern bereits die Dokumente im Angebot haben, die für die Arbeit des Parlaments wichtig sind – beispielsweise ein Archiv der Sitzungsprotokolle, Rechtsgrundlagen und organisatorische Dokumente.

Insbesondere um den Informationsfluss von außen zu stärken, müssen zudem Schnittstellen für andere Teile der Studierendenschaft bereitgestellt werden. So wäre es sinnvoll, wenn nicht nur Mitglieder des Studierendenparlaments Zugriff auf das System haben, sondern auch der AStA oder die Fakultätsratskonferenz ihre Sitzungsprotokolle dort einstellen kann. Damit DAVe trotzdem auch für die Bearbeitung von StuPa-internen Informationen genutzt werden kann, wäre eine entsprechende Rechteverwaltung nötig, die interne Dokumente nur für Mitglieder des Studierendenparlaments einsehbar macht.

Um dem Chaos verschiedener Dokumentversionen entgegenzuwirken, sollte das System eine Funktionalität bereitstellen, mit der verschiedene Versionen eines Dokuments zu einer Gruppe zusammenfassen lassen. Hierbei muss darauf geachtet werden, dass stets ersichtlich ist, welche Version die aktuell gültige ist und welche der letzte Bearbeitungsstand. Ebenso muss es möglich sein, diese einzelnen Versionen mit Meta-Informationen zu versehen, damit auch Dritte, die nicht direkt an der Bearbeitung des Dokuments beteiligt waren, den Arbeitsstand einzelner Versionen nachvollziehen

können. In Form von Anmerkungen könnten diese Meta-Informationen auch dazu dienen, für einzelne Versionen festzuhalten, welche Änderungen bei diesen noch notwendig sind.

Eine solche Versionsverwaltung würde für mehr Transparenz bei der Bearbeitung von Dokumenten sorgen und zudem anderen StuPa-Mitgliedern die Möglichkeit eröffnen, sich frühzeitig an der Bearbeitung eines Dokuments zu beteiligen.

Abschließend muss jedoch auch klargestellt werden, dass sich nicht alle Informationsdefizite im Studierendenparlament oder allgemein in der Studierendenschaft mit der Einführung eines Informationssystems lösen lassen. DAVe hat das Potenzial, die Situation zu verbessern und die Arbeit der Parlamentsmitglieder zu vereinfachen, indem es Informationen schneller zur Verfügung stellt und neue Kommunikationsmöglichkeiten bietet. Letztendlich hängt der Erfolg aber vor allem von der Akzeptanz der Nutzer ab, denn diese dürfen sich nicht darauf beschränken, DAVe als Informationsressource zu nutzen, sondern es auch selbst mit neuen Informationen füllen.

Fazit

Die Durchführung eines Information Audits als Grundlage für die Gestaltung eines Informationssystems ist zumindest in Deutschland noch eine Seltenheit. Dabei hat dieses Verfahren zwei wesentliche Vorteile. Zunächst hilft der Audit dem Systemarchitekten, bei der Erfassung der Anforderungen an ein System systematisch und dadurch auch umfassend vorzugehen. Nicht selten geschieht es, dass aufgrund einer mangelhaften Untersuchung der tatsächlichen Anforderungen[1] an ein System, Aspekte außer Acht gelassen werden und erst nach der Einführung des Systems entdeckt werden. Der andere wichtige Vorteil sind die Erkenntnisse, die man bei der Durchführung des Audits gewinnt. Denn sie bilden eine fundierte Grundlage, auf der man verlässliche Entscheidungen treffen kann – und das führt nicht nur zu einem System, das den Bedürfnissen der Benutzer möglichst nahe kommt, sondern sorgt auch für Sicherheit auf Seiten des Entwicklers.

Die Anwendung des Information Audits erfordert jedoch auch Flexibilität. In der Fachliteratur zum Information Audit liegt der Fokus auf der Entwicklung von

[1] Die tatsächlichen Anforderungen, die ein System erfüllen muss, differieren nicht selten stark von den Anforderungen, die der Auftraggeber zuvor definiert hat.

Informationsrichtlinien und -strategien. Der Audit als Werkzeug für die Konzeption von Informationssystemen rückt dabei eher in den Hintergrund, weshalb die vorhandenen Leitfäden nur bedingt anwendbar sind. Das in dieser Fallstudie verwendete Modell von Susan Henczel war zwar sehr ausführlich und auch hilfreich, jedoch galt es stets zu überprüfen, ob eine von ihr empfohlene Vorgehensweise im vorliegenden Fall überhaupt sinnvoll und effektiv ist.

Im Hinblick auf die Ergebnisse fällt das Fazit ebenfalls sehr positiv auf. Trotz der Tätigkeit des Autors im Studierendenparlament, konnte der Information Audit viele Aspekte ans Tageslicht bringen, die andernfalls in der Entwicklung von DAVe nicht berücksichtigt worden wären.

Auch wenn der Information Audit ein sehr zeit- und ressourcenintensives Verfahren ist, rechtfertigen die umfangreichen Erkenntnisse die Durchführung und ermöglichen es so, ein Informationssystem zu entwickeln, das tatsächlich den Nutzer in den Mittelpunkt stellt.

Literaturverzeichnis

BATLEY 2007

Batley, Sue: *Information Architecture For Information Professionals*. Oxford : Chandos Publishing, 2007. – ISBN 1-84334-233-2

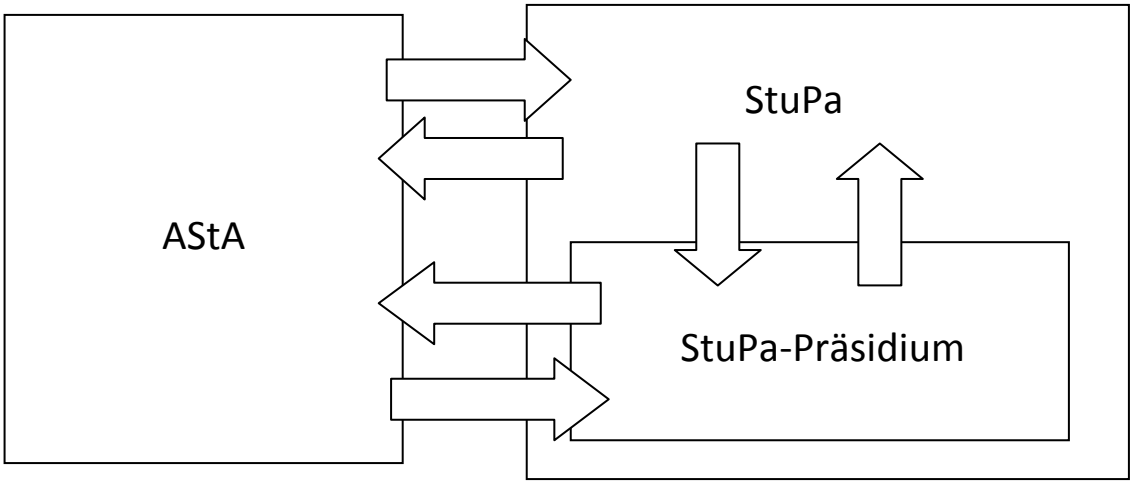
HENCZEL 2007

Henczel, Susan: *The Information Audit : A Practical Guide*. München : Saur, 2001. – ISBN 9-598-24367-7

Anhang A: Fragebogen

1. Welche Informationen und Dokumente, die dir momentan nicht zu Verfügung stehen, würden die bei der Arbeit für das StuPa helfen?

2. Trage in die Pfeile im folgenden Schema ein, wie du den Informationsfluss zwischen den einzelnen Gruppen bewertest.
   + = Gut
   O = Befriedigend/Ausreichend
   — = Schlecht



3. Besitzt du selbst Informationen (Dokumente, Informationsblätter, Nachrichten, etc.), die nicht direkt sitzungsrelevant sind, aber für andere Mitglieder vom StuPa und AStA interessant sein könnten?

   O JA O NEIN

   Hast du solche Informationen bislang kommuniziert? Falls ja, auf welchem Wege? Falls nein, warum nicht?

4. Inwieweit stimmst du mit folgenden Aussagen überein?
   (1 = Ich stimme voll zu, 6 = Ich stimme gar nicht zu)

   a. Wenn ich Dokumente bzw. Informationen benötige oder sie mich interessieren, habe ich ohne großen Aufwand Zugriff auf diese.

      O 1 O 2 O 3 O 4 O 5 O 6

   b. Aktuelle Versionen von Satzungen und Ordnungen stehen mir zur Verfügung.

      O 1 O 2 O 3 O 4 O 5 O 6

   c. Auch außerhalb der Sitzungen habe ich die Möglichkeit, mich in die Arbeit an Dokumenten einzubringen (auch wenn ich nicht Mitglied einer Arbeitsgruppe bin).

      O 1 O 2 O 3 O 4 O 5 O 6

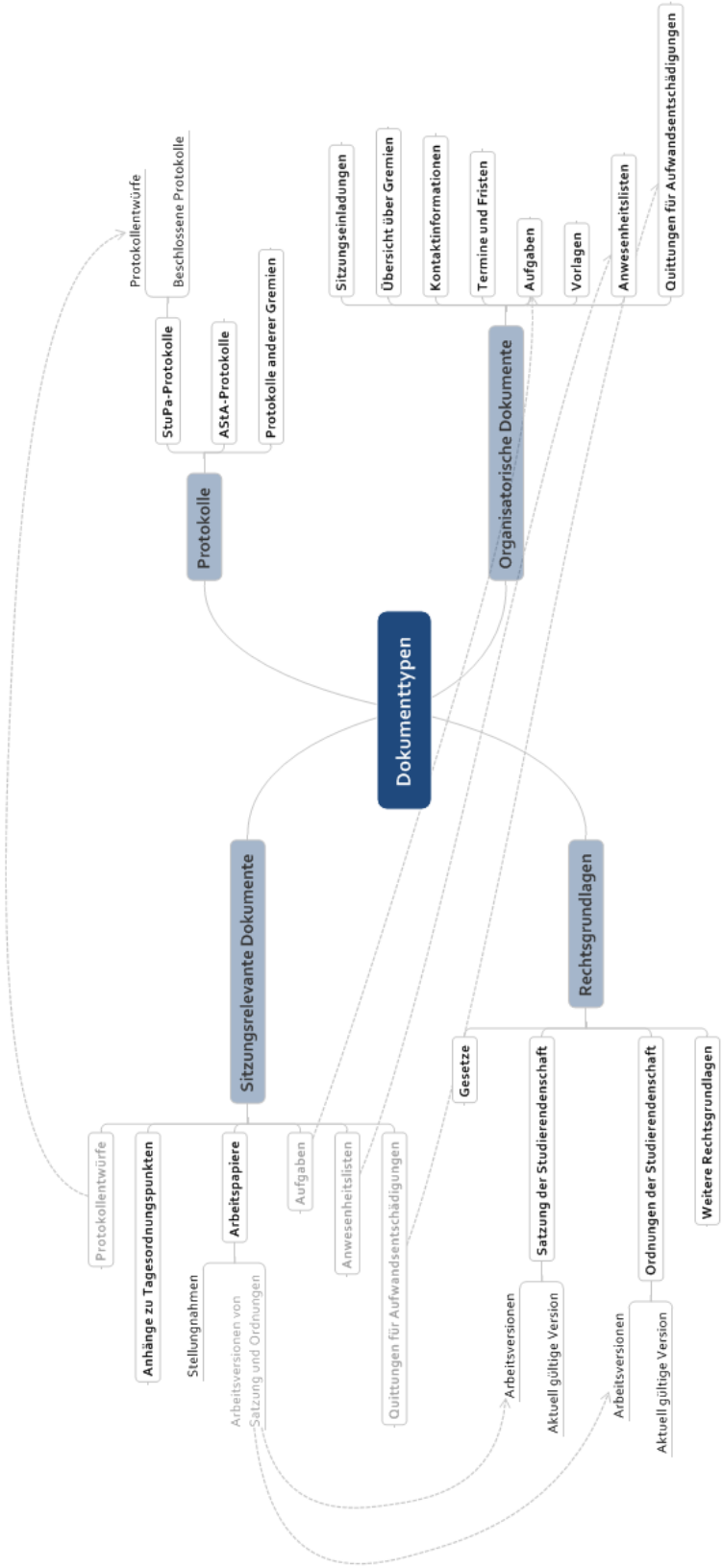
   d. Protokolle aus anderen Gremien helfen mir bei der Arbeit im StuPa.

      O 1 O 2 O 3 O 4 O 5 O 6

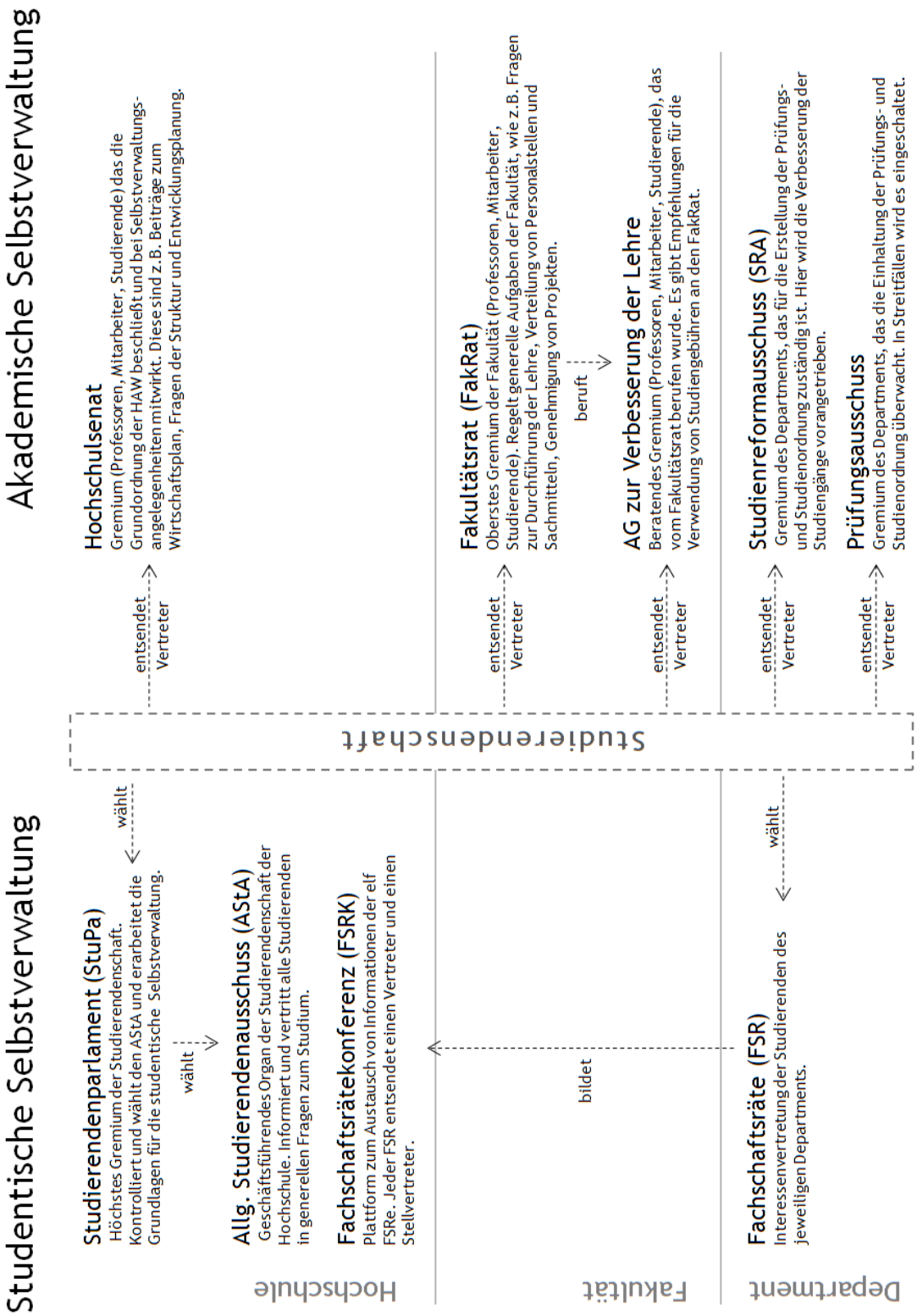
   e. Ich würde mich durch das Lesen von Dokumenten auf Sitzungen vorbereiten, stünden Sie rechtzeitig mir vor der Sitzung auf einer Online-Plattform zur Verfügung.

      O 1 O 2 O 3 O 4 O 5 O 6

Anhang B: Dokumenttypen



Anhang C: Schaubild «Gremien der HAW Hamburg»



Eidesstattliche Versicherung

Ich versichere, die vorliegende Arbeit selbständig ohne fremde Hilfe verfasst und keine anderen Quellen und Hilfsmittel als die angegebenen benutzt zu haben. Die aus anderen Werken wörtlich entnommenen Stellen oder dem Sinn nach entlehnten Passagen sind durch Quellenangabe kenntlich gemacht.

Ort, Datum Unterschrift