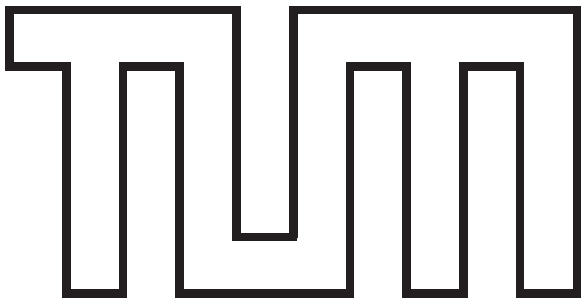

TECHNISCHE UNIVERSITÄT MÜNCHEN
INSTITUT FÜR INFORMATIK

Eine Software-Architektur für webbasierte Publikationssysteme

Thomas Schöpf

TECHNISCHE UNIVERSITÄT MÜNCHEN
INSTITUT FÜR INFORMATIK

Eine Software-Architektur für webbasierte Publikationssysteme

Thomas Schöpf

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender:		Univ.-Prof. G. J. Klinker, Ph.D.
Prüfer der Dissertation:	1.	Univ.-Prof. Dr. A. Brüggemann-Klein
	2.	Univ.-Prof. Dr. E. Jessen, em.

Die Dissertation wurde am 12.03.2009 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 07.09.2009 angenommen.

Zusammenfassung

Die zu beherrschende Komplexität der Erstellung von Publikationsanwendungen ist in den letzten Jahren deutlich gestiegen. Durch diese Komplexität wurde die erfolgreiche Erstellung von Publikationsanwendungen, die alle technischen Möglichkeiten nutzen, erheblich schwieriger. Neben technischen Innovationen tragen auch sich häufig ändernde Anforderungen durch den Markt und den gestiegenen globalen Wettbewerb zu diesen Problemen bei. Erschwerend kommt hinzu, dass die Vorgehensweisen bei der systematischen Realisierung von Publikationsanwendungen noch nicht gut verstanden sind, sondern wie so oft für ein neues Einsatzgebiet die Softwaresysteme wieder neu entwickelt werden. Dadurch kommt bereits erlangtes Wissen in der Software-Technik nicht zum Tragen.

Im Fachgebiet des elektronischen Publizierens wurden häufig unterschiedliche Begriffe für den gleichen zugrunde liegenden Sachverhalt verwendet und einzelne Begriff konnten auch unterschiedliche Bedeutungen besitzen. Daher werden in dieser Arbeit zunächst einheitliche Definitionen der wesentlichen Begriffe gegeben sowie grundlegende Konzepte und Prozesse geklärt, um eine klar definierte Begriffsbasis zu schaffen. Als weiterer grundlegender Schritt wird eine Menge von Anforderungen erarbeitet, die auf den Möglichkeiten strukturierter Dokumente und der Publikationskette basieren.

Zur gesammelten Betrachtung werden einander ähnliche Anforderungen zu Funktionsclustern zusammengefasst. Die Signifikanz eines jeden Funktionsclusters wird anhand einer Vielzahl von Einsatzszenarien eruiert, die in dieser Arbeit erarbeitet werden. Aus dieser Betrachtung ergibt sich eine Gruppierung von Einsatzszenarien mit gleichen Signifikanzen von Funktionsclustern. Damit ist es einfacher, für ein neues Szenario eine Ausgangsbasis für eine Implementierung zu finden.

Für künftige Neuentwicklungen von Publikationssystemen wird eine Software-Architektur erarbeitet und vorgeschlagen, die besonders dem Qualitätsziel der Modifizierbarkeit Rechnung trägt. Da der Modifizierbarkeit sehr hohe Bedeutung beigemessen wird, wird speziell dafür ein Meta-Modell entwickelt, welches die leichte Anpassung an veränderte Anforderungen ermöglicht und das Wissen zur Erstellung von Publikationsanwendungen auf eine höhere, abstraktere Ebene befördert.

Diese Arbeit trägt somit zu einem tieferen und umfassenderen Verständnis des Fachgebiets des elektronischen Publizierens bei. Sie hebt das das Fachwissen auf eine höhere Ebene und eröffnet Perspektiven für die Gestaltung von zukünftigen Publikationssystemen, weg von spezialisierten, spezifischen Publikationssystemen hin zu einer allgemeinen Verwendbarkeit und der Bewahrung und Weiterentwicklung des erworbenen Wissens.

Danksagung

Das Gelingen dieser Arbeit wäre nicht möglich gewesen ohne die Hilfe einer ganzen Reihe von Personen, denen ich an dieser Stelle herzlich dafür danken will.

An erster Stelle bedanke ich mich bei meinen beiden Betreuern Frau Prof. Dr. Anne Brüggemann-Klein und Herrn Prof. Dr. Eike Jessen dafür, dass sie mir diese Arbeit ermöglicht und durch viele hilfreiche Anregungen und geduldige konstruktive Unterstützung zur Fertigstellung beigetragen haben.

Des Weiteren danke ich meinen Eltern, Freunden und Kollegen am Lehrstuhl, die mich mit einem offenen Ohr für meine Gedanken unterstützt und mit Geduld meine zeitlichen Einschränkungen akzeptiert haben. Dies gilt besonders für Frau Stephanie Schnitzler, Frau Julia Fleischer, Frau Dr. Andrea Winter, Herrn Dr. Martin Stumpf, Herrn Stefan Behr und Herrn Friedrich Behr.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

1.1 Hintergrund

In den Jahren seit der Entwicklung des World Wide Web (WWW) am Anfang der 1990er Jahre[1] haben sich in vielen Bereichen bedeutende Veränderungen vollzogen. Am Anfang war das World Wide Web nur für wissenschaftliche Publikationen gedacht: In dem ursprünglichen Vorschlag[2] für das WWW von Tim Berners-Lee von 1989 wurde das Problem in den Mittelpunkt gestellt, dass am Forschungszentrum CERN, an dem Berners-Lee tätig war, zwar viele Informationen über Teilchenbeschleuniger und die damit durchgeführten Experimente gesammelt wurden, aber keine geeignete Verwaltung dieser Informationen existierte. Dies führte dazu, dass andere Forschungsgruppen nicht in der Lage waren, auf diese Informationen zuzugreifen und sie ebenfalls für ihre Arbeit zu nutzen. Deshalb sollte den Mitarbeitern am CERN ein Hypertext-System angeboten werden, das diese Informationen verwaltet und allen Mitarbeitern gleichermaßen den leichten Zugang zu diesen Informationen ermöglicht.

Das World Wide Web etablierte eine Reihe offener[3] Standards und führte dazu, dass Wissenschaftler Informationen über ihre eigenen Arbeiten weltweit veröffentlichen und ebenso die Texte anderer Wissenschaftler nutzen können. Für den Zugriff auf diese Dokumente waren sie nicht auf einen bestimmten Softwarehersteller oder eine bestimmte Hardwareausstattung angewiesen. Über das offene Hypertext Transfer Protocol (HTTP) wurden die Informationen, die in der Hypertext Markup Language (HTML) beschrieben waren, von einem Web-Server auf den Clientrechner übertragen, auf dem ein so genannter Browser dem Benutzer die Inhalte präsentierte. Außerdem besitzt das World Wide Web eine verteilte Architektur: Der Nutzer sieht nicht, dass er mit einer Verknüpfung oftmals auf einen anderen Rechner überwechselt.

Damit war der erste Schritt einer Entwicklung gemacht, die bis heute nicht zum Stillstand gekommen ist. War zu Anfang der 1990er Jahren noch das Problem, die Informationen überhaupt in einer geeigneten Form zur Verfügung stellen zu können, hat sich der Fokus inzwischen mehr auf die weiter gehende semantische Beschreibbarkeit der Daten und die Integration verschiedener Dienste verlagert. Es geht vor allem darum, Dokumente mit semantischen Informationen anzureichern und damit eine sinnvolle Weiterverarbeitung zu ermöglichen. Ein Beispielszenario hierfür kann folgendermaßen aussehen [Bun02]:

[1] A Little History of the World Wide Web, `http://www.w3.org/History.html`

[2] The original proposal of the WWW, HTMLized, `http://www.w3.org/History/1989/proposal.html`

[3] Offen in dem Sinne, dass die Standards öffentlich verfügbar waren und es keine finanziellen oder rechtlichen Hürden gab, die einer Implementierung durch Dritte entgegenstanden.

> Eine Wissenschaftlerin kommt morgens ins Labor. Sie holt sich aus der internen Datenbank des Rauminformationssystems den aktuellen Versuchsaufbau als Flussdiagramm auf die Anzeigetafel an der Laborwand. Während sie ihren Arbeitsplatz vorbereitet, lässt sie das System extern im Internet und in kommerziellen Datenbanken recherchieren, ob es zu einem Teilbereich ihres Versuches bereits bekannte Forschungsergebnisse gibt. Einen Reaktionszyklus, über den die Forscherin auf dem Weg zur Arbeit besonders intensiv nachgedacht hat, zeichnet sie als Strukturformel direkt auf die Tafel. Das System übernimmt das Formelbild als Frage und liefert wenig später mehrere Treffer, die es aus hochwertigen, qualitätsgeprüften Chemiedatenbanken hervorgeholt hat. Dann liest das Rauminformationssystem die Ergebnisse nach Rangfolge der Übereinstimmung zwischen Suchanfrage und Treffer laut vor. Auf Zuruf entscheidet die Wissenschaftlerin, welche Publikation verworfen und welche zur späteren Auswertung als Abstract oder im Volltext in ihre persönliche, digitale Wissensbibliothek übernommen werden soll.

Dieses Szenario macht deutlich, dass die Dokumente nicht mehr einfach nur übertragen werden, sondern dass die Computer in der Lage sein müssen, sie zu verstehen, um beispielsweise komplexe Suchanfragen zu beantworten, die gefundenen Dokumente in die persönliche Wissensbibliothek zu integrieren und zu verschlagworten. Es sind viele weitere Vorteile denkbar, die sich aus der Anreicherung von Dokumenten mit semantischen Informationen über ihren Inhalt ableiten lassen. Um nur ein zusätzliches Beispiel zu nennen: Ein Wissenschaftler könnte sich über das Internet zu einer Fachtagung anmelden und auf Knopfdruck wird der Termin für die Tagung in seine persönliche Terminverwaltung übernommen, weil diese in der Lage war, das Datum auf der Tagungsseite zu erkennen und zu extrahieren.

1.1.1 Elektronisches Publizieren

Unter elektronischem Publizieren ist die Verfügbarmachung von elektronischen Dokumenten zu verstehen. Der Prozess des elektronischen Publizierens umfasst dabei alle Stationen von der Erstellung eines Dokuments, seiner Verarbeitung und Aufbereitung, der Bereitstellung und Übertragung bis zu seiner Nutzung.

Moderne Dokumente sind dabei nicht nur einfache, lineare Texte, sondern sie können auch dynamisch, interaktiv sowie multimedial sein. Sie können nicht-linear sein (Hypermedia), werden über verschiedene Publikationskanäle und in unterschiedlichen Formaten angeboten, werden auf unterschiedlichen Endgeräten genutzt und sind individuell an die Nutzungsansprüche der Zielgruppe anpassbar. Einzelne Dokumente können auch aus mehreren unterschiedlichen Quellen zusammengesetzt sein (z. B. bei Content Syndication). In einem späteren Kapitel werden die Anforderungen zusammengestellt, denen sich moderne Dokumente und das elektronische Publizieren gegenübersehen.

Moderne Dokumente werden nicht ausschließlich von Menschen geschrieben, sondern sie können auch von Programmen oder Maschinen generiert worden sein. Ebenso verwenden nicht nur Menschen die so entstandenen Dokumente. Auch Programme und Maschinen können solche Dokumente einlesen und weiterverarbeiten. Daraus ergibt sich bereits eine Anforderung an

Dokumente: Sie müssen in einem Format abgefasst sein, das von Menschen und Programmen gleichermaßen gelesen bzw. verarbeitet werden kann.

Beim elektronischen Publizieren lässt sich eine ganz ähnliche Entwicklung beobachten, wie beim World Wide Web: Die Gestaltungsmöglichkeiten wurden bzw. werden immer weiter ausgebaut, gleichzeitig steigt damit auch die Komplexität. Anfänglich besaß HTML, die Auszeichnungssprache des World Wide Web, nur sehr eingeschränkte Möglichkeiten zur grafischen Gestaltung der Seiten. HTML war nicht dazu gedacht, eine Seite in einer bestimmten, festen Präsentation bereitzustellen, sondern der Inhalt stand im Vordergrund. Dies ergibt sich schon aus der Motivation, warum das World Wide Web entwickelt wurde (vgl. Abschnitt 1.1). Man konnte zwar unter anderem festlegen, was eine Überschrift war, ob etwas hervorgehoben werden sollte und konnte einfache Elemente wie z. B. Aufzählungslisten definieren. Wie diese Elemente dann tatsächlich präsentiert wurden, war die Sache des Browsers. Die Browser der verschiedenen Hersteller haben sich dabei jeweils etwas anders verhalten, so dass die Seiten zwar ähnlich dargestellt wurden, sich in den Details jedoch unterschieden.

Mit den steigenden Ansprüchen an die grafische Gestaltung der Inhalte, der zunehmenden Kommerzialisierung und der damit einhergehenden Entwicklung, sich durch eine „Corporate Identity“ auch im World Wide Web von den Konkurrenten abzuheben, war dieser Zustand jedoch nicht lange haltbar: HTML wurde um die Möglichkeit erweitert, verschiedene Elemente der Seite mit bestimmten grafischen Eigenschaften zu versehen (Hintergrundfarben, Rahmen, Schriftarten und -größen etc.). Teilweise wurden diese Erweiterungen an den Standardisierungsgremien des W3 Consortiums vorbei von den Herstellern der Browser im Alleingang durchgeführt.

Dies führte zu einer Entwicklung, die in der Praxis kaum mehr akzeptabel war: Jeder Browser besaß spezifische Eigenheiten im Verhalten, in denen er sich von den Browsern der anderen Hersteller unterschied. Dabei waren von den Webdesignern nicht nur die Browser verschiedener Hersteller zu unterscheiden, sondern auch noch die unterschiedlich aktuellen Versionen eines Browsers eines einzelnen Herstellers. Da die grafischen Auszeichnungen in der gleichen Datei gespeichert waren, wie der eigentliche Inhalt der Seiten, wären viele angepasste Dateien nötig gewesen, die aber jeweils den gleichen Inhalt haben – aber mit jeweils anderen Auszeichnungen für die unterschiedlichen Browser. Bei jeder Änderung der gewünschten Präsentation hätte jede einzelne dieser Dateien geändert werden müssen. Dies wäre nicht mehr mit vertretbarem Aufwand durchführbar gewesen.

Der frühe Erfolg des World Wide Web basierte jedoch auch darauf, dass es einen allgemein anerkannten Standard gab und man nicht gezwungen war, einen bestimmten Browser einzusetzen, der nur auf bestimmten Betriebssystemen funktionierte. Durch die eigenmächtigen und zueinander inkompatiblen Erweiterungen [Jec04] des Standards durch die großen Browserhersteller, drohte der offene Standard nutzlos zu werden.

Das W3C hat daraufhin 1996 den HTML-Standard in der Version 3.2[4] veröffentlicht[5], der HTML um viele Sprachelemente erweiterte, die der grafischen Gestaltung dienen. Die Einbettung der grafischen Auszeichnung in den HTML Markup hat sich jedoch nicht als ideal erwiesen

[4] `http://www.w3.org/TR/REC-html32`
[5] `http://www.w3.org/MarkUp/Wilbur/pr7may96.html`

und so wurde 1999 der HTML-Standard in der Version 4.0[6] veröffentlicht, der die entsprechenden Elemente und Attribute von HTML 3.2 als „deprecated“ gekennzeichnet hat. Gleichzeitig wurde mit Hilfe der so genannten „Cascading Style Sheets“ (CSS)[7] die Möglichkeit geschaffen, die grafische Auszeichnung von den eigentlichen Inhalten zu trennen. Dadurch wurde es möglich, das optische Erscheinungsbild zu verändern, indem nur eine einzige Datei angepasst wird, in welcher die Präsentationsanweisungen abgelegt sind.

Neben dem Trend, die Publikationen grafisch ansprechender und aufwändiger zu gestalten, zeichnete sich auch bald ab, dass eine gewisse Dynamik und eine Möglichkeit der Interaktivität der Seiten gewünscht wird. Dies bedeutete, dass in zunehmendem Maße die Seiten dynamisch von Programmen generiert wurden. Ein klassisches Beispiel für solche dynamisch erzeugten Seiten sind elektronische Warenkörbe, die bei Einkäufen in Online-Shops die Artikel speichern, die von einem Kunden zum Kauf vorgemerkt sind. Entsprechend höher ist jedoch inzwischen auch die Komplexität der Internetauftritte geworden.

Aus dem steigenden Umfang und der zunehmenden Komplexität ergibt sich noch ein weiterer Grund, weshalb moderne Dokumente im elektronischen Publizieren auch von Programmen verarbeitbar sein müssen: Ab einer gewissen Menge von Dokumenten, die eine elektronische Publikation umfasst, ist eine Automatisierung von Arbeitsschritten (z. B. Formatumwandlungen) unumgänglich. Anders ist der anfallende Aufwand nicht mehr zu handhaben.

Eine Möglichkeit, um die steigende Komplexität handhabbar zu halten, ist, das Gesamtproblem in diskrete Teilbereiche zu zerlegen, die über klare Schnittstellen miteinander verbunden werden können (vgl. [Kro97]). Ein Beispiel für eine solche Aufteilung ist das Model-View-Controller Konzept, das Daten (Model), die Präsentation (View) und die so genannte Programmlogik (Controller) voneinander trennt. Das Model-View-Controller Konzept kann auch auf den Prozess des elektronischen Publizierens angewendet werden. Dabei werden die verschiedenen Zuständigkeitsbereiche für die Inhalte, die Präsentation und die Programmlogik der Seiten voneinander getrennt. Im Bereich des elektronischen Publizierens bezeichnet „Logik“ den Programmcode, der von einem Prozessor oder durch einen Interpreter ausgeführt wird.

Ging es am Anfang der Entwicklung noch hauptsächlich darum, den gestiegenen Anforderungen an die Präsentation gerecht zu werden, rückte inzwischen ein ganz anderer Aspekt ins Zentrum des Interesses: Es sollte eine weiter gehende Kodierung der Struktur eines Dokuments möglich sein, als es mit HTML möglich ist. Das Ziel dabei war, automatisch auch eine semantische Verarbeitung unterstützen zu können. Es gibt bestimmte Gruppen von Anwendungen, die aufgrund der Beschränkungen von HTML nicht oder nur sehr schwer möglich wären [Bos97]:

1. Anwendungen, die versuchen, einen wesentlichen Teil der Berechnung vom Web-Server auf den Client zu übertragen.
2. Anwendungen, die erfordern, dass der Client dieselben Daten unterschiedlichen Nutzern in verschiedenen Präsentationen bereitstellt.

[6] `http://www.w3.org/TR/html4/`
[7] `http://www.w3.org/Style/CSS/`

3. Anwendungen, bei denen intelligente Agenten versuchen, die Informationsfindung angepasst an die Bedürfnisse des Nutzers durchzuführen. Dazu sind sowohl Informationen über die Nutzer als auch über die Informationen selbst nötig.

1.1.2 Publikationssysteme

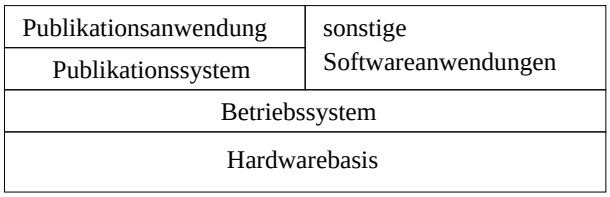

Abbildung 1.1: Einordnung von Publikationssystemen und -anwendungen

Grob ausgedrückt ist ein Publikationssystem ein Softwaresystem, das den Prozess des elektronischen Publizierens unterstützt. Es kann als Framework oder Middleware aufgefasst werden, die einen nutzbaren, äußeren Rahmen zur Verfügung stellt. Es bietet eine Reihe von Funktionen, die für elektronische Publikationen genutzt werden können. Das Publikationssystem unterstützt auch die weiter oben angesprochene Trennung der Zuständigkeitsbereiche für Inhalte, Präsentation und Programmlogik.

Wie in einem späteren Kapitel noch deutlich werden wird, gibt es bislang noch kein Publikationssystem, das alle Anforderungen an Publikationssysteme erfüllt. Es gibt aber Publikationssysteme, die sie zu einem Teil erfüllen. Die Systeme besitzen alle eine gewisse Kernfunktionalität, die sich direkt aus den Anforderungen des elektronischen Publizierens ergibt. Auf diese allgemeinen Anforderungen wird in einem späteren Kapitel noch konkret eingegangen werden. Die Publikationssysteme unterscheiden sich allerdings neben dem Funktionsumfang auch in der Art, wie bestimmte Anforderungen erfüllt werden. Teilweise sind dabei die Grenzen zwischen den verschiedenen konkreten Ansätzen unscharf.

1.1.3 Publikationsanwendungen

Ein Publikationssystem stellt die programmierbare – und somit anpassbare – Umgebung für eine Publikationsanwendung dar. Diese Umgebung wird von der Publikationsanwendung verwendet, um eine konkrete elektronische Publikation durchzuführen. Dies ist vergleichbar mit der Unterscheidung zwischen Betriebssystemen und Anwendungssystemen bei Computersystemen. Das Betriebssystem besitzt verschiedene Basisaufgaben. Dazu gehört beispielsweise die Verwaltung der Ressourcen, die den Anwendungen zur Verfügung stehen. Das Betriebssystem implementiert damit eine Art Rahmen, in dem sich die Anwendungen bewegen können. Die Anwendungen bekommen vom Betriebssystem auch eine Menge von Funktionen über Schnittstellen (APIs) zur Verfügung gestellt, die sie verwenden können, um ihre Aufgaben zu erledigen.

Analog dazu handelt es sich bei Publikationsanwendungen also um die Verwendung des Rahmens, der von einem Publikationssystem angeboten wird, um eine bestimmte elektronische Publikation zu ermöglichen.

Konkrete Beispiele für Publikationsanwendungen sind:

- Digitale Bibliotheken (ACM, CiteSeer, Springer Link).
- Portale (Geoinformation, Chemieinformation, Gesetzestexte, Nachrichten, Lifestyle).
- Persönliche digitale Bibliotheken
- eLearning-Plattformen
- Warenwirtschaftssysteme für Online-Shops

1.2 Motivation

In den vorangegangenen Abschnitten wurde der Hintergrund der Arbeit vorgestellt. Das elektronische Publizieren besteht nicht einfach darin, ein monolithisches Dokument auf einen Web-Server zu stellen, damit der Nutzer es abrufen, sich durchlesen und ausdrucken kann. Moderne Publikationsanwendungen umfassen weit mehr. Es geht dabei unter anderem um Dynamik, Interaktivität, Personalisierung und automatische Verarbeitung der Dokumente. Diese neuen Möglichkeiten sollen entsprechend auch von den Publikationssystemen unterstützt werden. Dadurch ist jedoch auch die Komplexität der Erstellung einer Publikationsanwendung deutlich höher als noch vor einigen Jahren (vgl. [Cla99]). Es gibt beispielsweise inzwischen Turing-vollständige Transformationssprachen [Kep02]. Damit erreicht die Erstellung einer Publikationsanwendung eine Komplexität, die vergleichbar ist mit der Komplexität der Erstellung von Softwareanwendungen.

Im Bereich der Softwareentwicklung wurde in den sechziger Jahren der Begriff der „Softwarekrise“ geprägt [Zus99]. Zuvor kam es zu den ersten großen gescheiterten Software-Projekten. Die Rechner wurden leistungsfähiger und gleichzeitig stieg damit die Komplexität der Software, die auf diesen Rechnern möglich war. In der Folge wurde auch die Erstellung dieser Software komplexer und aufwändiger. Der seitdem verfolgte Ansatz, die Komplexität der Softwareentwicklung in den Griff zu bekommen, umfasst u. a. die Entwicklung und den Einsatz von strukturierten Software-Entwicklungsprozessen – so genannten Vorgehensmodellen für die Softwareentwicklung.

Eine vergleichbare Entwicklung zeichnet sich auch im Bereich des elektronischen Publizierens ab: Anfangs waren die Möglichkeiten des elektronisches Publizierens sehr beschränkt, dementsprechend gab es auch keine Schwierigkeiten, diese geringe Komplexität zu beherrschen. Inzwischen werden die technischen Möglichkeiten immer umfangreicher und bieten viel mehr Spielräume und Kombinationsmöglichkeiten an. Folglich wird es auch immer schwieriger, Publikationsanwendungen, die diese Möglichkeiten ausnutzen, erfolgreich umzusetzen. Die Handhabung der Komplexität, ein Publikationssystem und auf dieser Basis dann eine Publikationsanwendung zu implementieren, stellt inzwischen eine große Herausforderung dar.

Die besondere Erschwernis dabei ist, dass die Vorgehensweisen bei der systematischen Realisierung einer Publikationsanwendung noch nicht gut verstanden sind. Dies liegt daran, dass heutige Publikationssysteme normalerweise nur bei Bedarf für einen bestimmten und begrenzten Einsatzzweck entwickelt werden. Das dabei erarbeitete Wissen wird nicht für die Entwicklung zukünftiger oder der Weiterentwicklung bestehender Publikationssysteme konserviert und bereitgestellt.

Der Bedarf an einer fundierten Unterstützung wird von der aktuellen Literatur und Veröffentlichungen nicht abgedeckt. Hier werden eher die statische Architektur und die Trennung von Layout, Inhalt und Struktur betrachtet, jedoch nicht oder nur sehr am Rande die neuen dynamischen Möglichkeiten des elektronischen Publizierens.

Ein weiterer Aspekt sind die sich inzwischen nahezu täglich ändernden Anforderungen durch die aktuelle Marktsituation, den globalen Wettbewerb und häufige technische Innovationen. Diese sich ändernden Anforderungen müssen immer wieder erfüllt werden. Dinge wie Optimierung spielen daher erst einmal keine Rolle (vgl. [KP02]).

1.3 Ziele

Das wesentliche Ziel dieser Arbeit ist die verbesserte Unterstützung der Möglichkeiten des elektronischen Publizierens durch Publikationssysteme und die Vereinfachung der Nutzung dieser Möglichkeiten durch Publikationsanwendungen. Zu diesem Zweck wird ein Vorschlag für eine Software-Architektur für Publikationssysteme erarbeitet. Im Rahmen dieser Architektur wird ein Meta-Modell entwickelt, das die Eigenschaften, den Aufbau und das Verhalten von Publikationsanwendungen auf einer höheren Abstraktionsebene beschreibbar macht. Alle Publikationssysteme, die diese einheitliche Modellierungssprache verstehen, sind damit in der Lage, die Anforderungen des Einsatzszenarios von derart beschriebenen Publikationsanwendungen zu erfüllen.

Ein ganz wesentlicher Vorteil davon ist, dass das Know-How zur Realisierung von Publikationsanwendungen dann nicht mehr so sehr auf das Spezialwissen zu den jeweiligen Publikationssystemen begrenzt ist, sondern deutlich breiter einsetzbar wird. Das Know-How wird damit deutlich stabiler und allgemeingültiger und würde nicht mit jeder neuen Version eines Publikationssystems wieder verschwinden. Diese Arbeit trägt dazu bei, eine angemessenere Abstraktion für Publikationsanwendungen und eine einheitliche Terminologie für das Fachgebiet zu etablieren, die bislang offenbar noch nicht existieren. Außerdem erleichtert die auf einer höheren Abstraktionsstufe stattfindende Beschreibung von Publikationsanwendungen die Kommunikation über diese Publikationsanwendungen und die Nachvollziehbarkeit und Dokumentation der Abläufe.

Eine der wesentlichen Ursachen der sogenannten Softwarekrise war die massiv gestiegene Komplexität der Software, die entwickelt werden sollte. In allen Ingenieursdisziplinen werden zur Beherrschung der Komplexität von Aufgaben u. a. folgende Grundsätze benutzt[GJM91]:

- Abstraktion durch Modelle: Der Entwurf eines Produkts läuft auf verschiedenen, nacheinander folgenden Stufen der Abstraktion. Dazu müssen Modelle gebildet werden, d. h.

es wird von den Einzelheiten, die für das zu lösende Problem gerade nicht relevant sind, abstrahiert. Dieser Grundsatz wird direkt durch das Meta-Modell verfolgt und erreicht.

- Teile und Herrsche: Das zu lösende Problem und auch der Lösungsweg werden in geeignete Teile zerlegt. Dabei werden verschiedene Aspekte betrachtet, in der Softwaretechnik z. B.: Das Gesamtprodukt (die Software) wird in Komponenten zerlegt, der Zeitverlauf des Projekts in Zeitphasen, die Verantwortung durch Arbeitsteilung im Team. Dieser Grundsatz wird zum einen durch die vorgeschlagene Komponentenaufteilung und zum anderen durch die in Kapitel 4 vorgeschlagenen Richtlinien für die inkrementelle Entwicklung von Publikationssystemen realisiert.
- Standard-Werkzeuge: Werkzeuge werden entwickelt und als Standards systematisch benutzt. Dieser Grundsatz wird ebenso wie der nächste sowohl durch den Vorschlag zur Software-Architektur als auch durch das Meta-Modell unterstützt. Standard-Werkzeuge können die Architektur und das Meta-Modell verwenden, und die Richtlinien- und Vorgehensvorschläge erlauben die Etablierung einer Standard-Methodologie.
- Standard-Methodologie: Standardisierte Techniken und Methoden werden entwickelt, zum verbindlichen Standard erklärt und systematisch benutzt.

Daneben werden weitere Ergebnisse erreicht, die entweder für die Entwicklung der Software-Architektur oder des Meta-Modells erforderlich sind:

Definition und Abgrenzung wichtiger Begriffe des elektronischen Publizierens
Viele Begriffe im Bereich des elektronischen Publizierens besitzen keine klare Definition sondern werden von den Personen, die in diesem Fachbereich arbeiten, intuitiv verstanden. Da dies zu Unklarheiten und Missverständnissen führt, werden wichtige Begriffe in dieser Arbeit definiert und von anderen Begriffen abgegrenzt.

Erarbeitung der Anforderungen an Publikationssysteme
Es wird untersucht, welchen Anforderungen Publikationssysteme genügen müssen, damit sie alle Möglichkeiten des elektronischen Publizierens unterstützen. Dabei werden ähnliche Anforderungen zu größeren Einheiten zusammengefasst.

Identifizieren von Klassen von Publikationsanwendungen
Publikationsanwendungen werden systematisch beschrieben und anhand ihrer Funktionalität klassifiziert. Ferner werden Funktionalitätsdefizite und Weiterentwicklungsmöglichkeiten identifiziert.

1.4 Vorgehen und Gliederung

Im Kapitel 2 werden die Grundlagen des Fachgebiets erarbeitet und wichtige Begriffe definiert, d. h. es wird beantwortet, was unter elektronischem Publizieren zu verstehen ist, welche unterschiedlichen Bedeutungen mit dem Begriff des elektronischen Publizierens verbunden werden und welche Rolle den Dokumentenmodellen und Dokumenten zukommt. Daneben werden im Abschnitt 2.5 basierend auf den Möglichkeiten strukturierter Dokumente die Anforderungen

an das elektronische Publizieren im allgemeinen und an Publikationssysteme im speziellen beschrieben.

Kapitel 3 untersucht Einsatzszenarien für Publikationsanwendungen. Dazu werden zunächst zusammengehörige Anforderungen aus Abschnitt 2.5 zu Funktionsclustern zusammengefasst. Anschließend werden die Szenarien beschrieben und die Bedeutung der Funktionscluster im jeweiligen Szenario bestimmt. Als Folgerung werden funktional ähnliche Szenarien zu Klassen zusammengefasst.

Das Kapitel 3 liefert die Grundlage für die Unterstützung der Entwicklung von Publikationssystemen. Um auf eine sinnvolle Aufteilung eines Publikationssystems in Komponenten hinzuarbeiten, werden zunächst Funktionscluster herausgebildet, die ähnliche Anforderungen aus Abschnitt 2.5 in größere Einheiten gruppieren. Anschließend wird eine Spannbreite von Szenarien für den Einsatz von Publikationssystemen untersucht. Dabei wird die Bedeutung der Funktionscluster in den jeweiligen Szenarien festgestellt. Dies liefert u. a. eine Unterstützung bei der Entscheidung, welche Funktionscluster für Publikationssysteme von primärem Interesse sind und bei der Neuentwicklung eines Publikationssystems zuerst entwickelt werden sollten. Basierend auf der Bewertung der Bedeutung der Funktionscluster in den einzelnen Szenarien werden ähnliche Szenarien zu Klassen zusammengefasst. Dies liefert eine Empfehlung, in welcher Reihenfolge ein Publikationssystem um Komponenten erweitert werden sollte. Abschließend werden Defizite und Weiterentwicklungsmöglichkeiten bei den Funktionsclustern aufgezeigt.

In Kapitel 4 ist das wesentliche Ziel, die Unterstützung bei der Entwicklung von Publikationssystemen zu verbessern. Um dies zu erreichen, wird auf Basis von Methoden der Softwaretechnik ein Vorschlag für eine Software-Architektur für Publikationssysteme erarbeitet. Dieser Vorschlag ist maßgebend von den Funktionsclustern aus Abschnitt 3.1 und somit auch von den Anforderungen an Publikationssysteme aus Abschnitt 2.5 bestimmt. In diesem Zusammenhang werden einige existierende Software-Systeme mit Blick auf ihre Eignung als Publikationssystem untersucht. Daneben werden Richtlinien für die Entwicklung neuer Publikationssysteme abgeleitet.

Anschließend wird im Kapitel 5 aus den charakteristischen Bestandteilen von Szenarien für Publikationsanwendungen und dem Architektur-Vorschlag ein Meta-Modell für Publikationsanwendungen entwickelt. Das Ziel dieses Kapitels ist es, eine Modellierung von Publikationsanwendungen in einer Form zu ermöglichen, dass die Modelle von einem Publikationssystem verwendet werden können, um sich an das Einsatzszenario anzupassen. Zu diesem Zweck wird nach einer grundlegenden Vorstellung von Meta-Modellierung ein ein entsprechendes Meta-Modell entwickelt, das die Bestandteile solcher Modelle beschreibt. Das Meta-Modell wird nach seiner Vorstellung anhand eines konkreten Beispiel-Szenarios illustriert. Außerdem wird das Vorgehen beschrieben, um das Meta-Modell auf andere Szenarien anzuwenden oder zu erweitern. Abschließend wird dargelegt, wie das Meta-Modell in Publikationssystemen genutzt werden kann.

Die Kapitel 4 und 5 kombinieren dabei das bekannte Reflection Muster mit dem Einsatz eines Meta-Modells, um die Modifizierbarkeit des Publikationssystems zu verbessern.

Eine Zusammenfassung der erreichten Ergebnisse und ein Ausblick auf im Rahmen der Arbeit

neu aufgeworfene aber noch nicht weiter bearbeitete Fragestellungen bilden den Schluss in Kapitel 6.

2 Elektronisches Publizieren

The good thing about standards is that there are so many to choose from.
– ANDREW S. TANENBAUM

Im Einleitungskapitel zu dieser Arbeit wurde bereits kurz beschrieben, was unter elektronischem Publizieren zu verstehen ist. In diesem Kapitel soll dies ausführlicher geschehen. Am Ende des Kapitels sollte deutlich geworden sein, was alles zum Bereich des elektronisches Publizierens gehört, welche Gebiete der Informatik eine wichtige Rolle spielen und was der aktuelle Stand der Forschung beim elektronischen Publizieren ist. Dieses Kapitel bezieht dabei viele Ideen und Inspirationen aus [BK03, BK04].

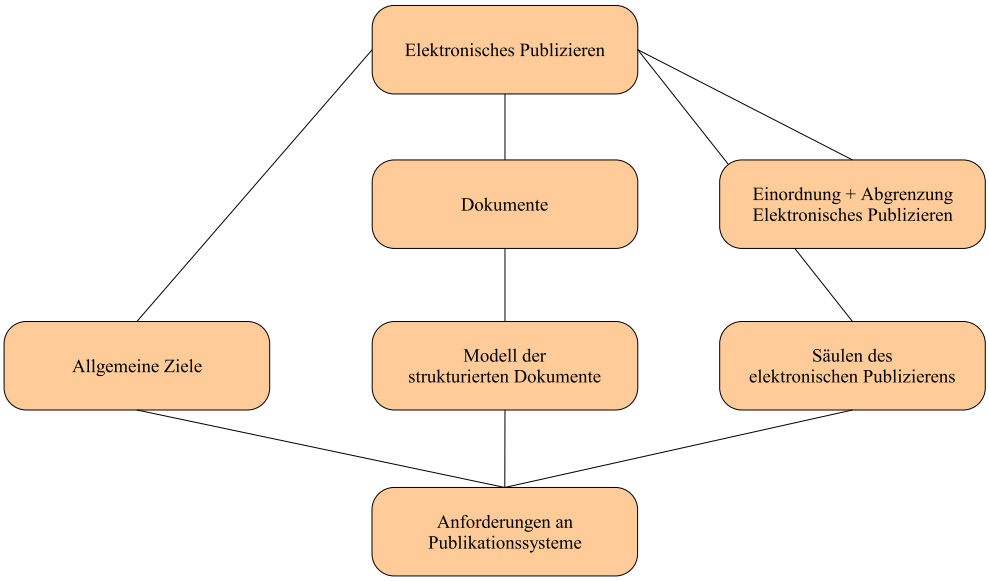

Abbildung 2.1: Kapitelübersicht

Die Grundlagen des elektronischen Publizierens werden in drei Hauptabschnitte aufgeteilt. Zunächst werden wichtige Begriffe des elektronischen Publizierens beschrieben. Der zweite Hauptteil stellt dann eine Einordnung und Abgrenzung zu anderen, verwandten Forschungsgebieten dar. Abgeschlossen wird das Kapitel durch eine Ableitung von Anforderungen an Systeme für das elektronische Publizieren. Die Abbildung 2.1 zeigt überblicksartig die logischen Zusammenhänge zwischen wichtigen Abschnitten dieses Kapitels. Die Knoten des Graphen repräsentieren die Themenbereiche, die in den Abschnitten behandelt werden. Die Kanten stehen für eine „Steht-in-Zusammenhang-mit“-Relation zwischen den Themenbereichen.

2.1 Einführung

Der Begriff „elektronisches Publizieren“ besitzt zwei verschiedene Bedeutungen. Als Erstes versteht man unter elektronischem Publizieren den Prozess, in dem Dokumente auf elektronischem Weg aus ihren Quellen zu ihrer Bereitstellung für den Endnutzer erzeugt werden. Als Quellen dienen dabei nicht ausschließlich menschliche Autoren; auch computergestützte Prozesse können Dokumente generieren. Die Dokumente können dabei nach ihrer Erzeugung eine Reihe von Zwischenstationen entlang der Publikationskette durchlaufen.

Die zweite Bedeutung des Begriffs „elektronisches Publizieren“ steht für das wissenschaftliche Fachgebiet, das Modelle, Vorgehensweisen und Werkzeuge zur Verfügung stellt, die den oben erwähnten Prozess des elektronischen Publizierens ermöglichen und unterstützen. Es beschäftigt sich mit den Grundlagen, der theoretischen Analyse und Konzeption von Systemen. In diesem Fachgebiet ist die vorliegende Arbeit anzusiedeln.

2.2 Allgemeine Ziele

Es gibt bestimmte allgemeine Ziele beim elektronischen Publizieren, die mit dem Einsatz der entsprechenden Publikationswerkzeuge, Modelle und Vorgehensweisen verfolgt werden (vgl. [Hes04], [Arb04a]). Diese Ziele sind im Folgenden zusammengefasst.

Zeitgewinn:
Die Dokumente sollen möglichst schnell bereitgestellt werden können, d. h. der Zeitaufwand zur Erstellung der Dokumente soll möglichst gering sein. Außerdem soll eine hohe Aktualität der erstellten und publizierten Inhalte erreicht werden.

Kostenreduktion:
Der Prozess des elektronischen Publizierens soll möglichst wenig Ressourcen benötigen. Dies betrifft sowohl die Kosten für Hardware, Software und Netzwerke als auch die Personalkosten. Als Hoffnungsträger für die Kostenersparnis dienen die medienneutrale Datenhaltung, die Wiederverwendbarkeit sowie die Verwendung digitaler Workflows (vgl. [Arb04a, Arb03, Die04]).

Qualitätssteigerung:
Es soll mit möglichst geringem Aufwand möglich sein, bei gegebener Qualifikation der Mitarbeiter, qualitativ hochwertige Inhalte zu erstellen. Es sollen inhaltlich und grafisch hochwertige und die Anforderungen der Nutzer möglichst gut erfüllende Publikationen erstellt werden können.

Bessere Mediendurchdringung:
Es sollen neue Nutzungsmöglichkeiten und -kanäle erschlossen werden, wie etwa Internet und mittlerweile auch mobiles Internet, also Internet auf mobilen Endgeräten. Die neuen Möglichkeiten und Kanäle sollen entsprechend der Nutzungsanforderungen und -potentiale unterstützt werden können.

Diese Ansprüche an Flexibilität und Effizienz erfordern ein auf diese Vorgaben zugeschnittenes Dokumentenmodell und dazu passende, leistungsfähige Werkzeuge zur Verarbeitung der Dokumente.

2.3 Begriffe, Konzepte und Prozesse

Das Fachgebiet des elektronischen Publizierens umfasst einige wichtige Begriffe und Konzepte, die in dieser Arbeit ebenfalls eine wichtige Rolle spielen. Die konkrete Bedeutung dieser Begriffe wird in den folgenden Abschnitten beschrieben.

2.3.1 Definition des Dokumentenbegriffs

Zunächst soll der Begriff des Dokuments festgelegt werden. Es gibt in der Literatur dafür bislang keine einheitliche Definition. Die erste Schwierigkeit besteht bereits darin, dass die entsprechende Bezeichnung im englischen Sprachraum in Form von „document“ eine weit weniger offizielle und amtliche Konnotation besitzt als das deutsche „Dokument“ (vgl. auch [BK03, BK04] zur historischen Veränderung des Begriffs „Dokument“). Hierzulande stellt man sich Dokumente meist als wichtige Unterlagen vor. Im Computerbereich werden jedoch auch z. B. Word-Dateien als Dokumente bezeichnet. Dementsprechend schwierig ist es, auf eine einheitliche Definition zu kommen.

Es gibt zusätzlich die Unterscheidung zwischen Dokumenten und elektronischen Dokumenten. In dieser Arbeit spielen jedoch nur elektronische Dokumente eine Rolle, so dass auch dann von elektronischen Dokumenten die Rede ist, wenn nur der Begriff „Dokument“ benutzt wird.

Die Definition, was unter einem Dokument zu verstehen ist, kann recht umfassend festgelegt sein. Die Grundaussage der entsprechenden Definition wäre, dass ein Dokument eine sprachlich kodierte Information transportiert (vgl. [BK03, Cla97]). Es ist jedoch das Ziel dieser Arbeit, eine umfassendere und präzisere Definition zu finden.

Ulrich Kampffmeyer definiert in [Kam03] (elektronische) Dokumente folgendermaßen:

> Elektronische Dokumente sind schwach oder unstrukturierte Informationen, die in einem in sich geschlossenen, authentischen, zeitpunktbezogen zusammenhängenden und inhaltlich originären Zusammenhang als Einheit in einem elektronischen System als Datei, Bestandteil einer Datei oder digitales Objekt vorliegen.

Diese Definition schränkt den Dokumentenbegriff unnötig stark ein. Auch ist nicht eingängig, weshalb Dokumente nur schwach und unstrukturierte und nicht auch stark strukturierte Informationen sein können.

In dieser Arbeit wird ein Mittelweg beschritten und der Begriff „Dokument“ folgendermaßen definiert:

> Unter einem Dokument ist eine endliche Datenabfolge zu verstehen, die in computerverarbeitbarer Form und effektiv zugreifbar nicht-flüchtig auf einem Datenträger vorliegt.

Dokumente sind somit nicht darauf beschränkt, einen Text (im Sinne von natürlicher Sprache) zu kodieren, sondern sie können auch Grafiken, Audio- und Videoinformationen enthalten.

Darauf aufbauend lautet die Definition für „strukturiertes Dokument“:

> Ein strukturiertes Dokument ist ein Dokument, bei dem zwischen den eigentlichen Inhalten, Strukturelementen und Präsentationsanweisungen unterschieden wird.

2.3.2 Das Modell der strukturierten Dokumente

Die Eigenschaften moderner, strukturierter Dokumente sind im sogenannten „Modell der strukturierten Dokumente“ beschrieben. Es ist ein Modell von Dokumenten, deren Inhalte mit Strukturelementen angereichert sind. Es definiert in diesem Zusammenhang auch eine bestimmte, grundlegende Begriffswelt.

Das Modell der strukturierten Dokumente stellt zunächst nur ein Gedankenmodell dar. Daher spielt es für die Verwendung im Prozess des elektronischen Publizierens eine entscheidende Rolle, dass auch eine konkrete syntaktische Realisierung des abstrakten Modells existiert. Diese syntaktische Realisierung legt fest, wie die Trennung zwischen Inhalten, Struktur und Präsentation konkret kodiert wird. Die aktuell wichtigste Realisierung stellt XML[1] dar, zusammen mit Softwaresystemen und Werkzeugen, die auf XML aufbauen.

Das Modell der strukturierten Dokumente wurde bereits in den 1980er Jahren entwickelt. Es hat sich seitdem zu einem maßgebenden Bestandteil des Fachgebiets des elektronischen Publizierens entwickelt. In den folgenden Abschnitten werden zunächst die wichtigsten Eigenschaften von strukturierten Dokumenten vorgestellt. Anschließend wird darauf eingegangen, welche Vorteile mit diesen Eigenschaften verbunden sind.

2.3.2.1 Eigenschaften strukturierter Dokumente

Das Modell der strukturierten Dokumente teilt ein Dokument in drei Teile auf: Inhalte, Struktur und Präsentation. Die Dokumente sind computerlesbar, da sie durch die zusätzlichen Strukturelemente mit den entsprechenden semantischen Daten angereichert sind, mit denen ein Computer in der Lage ist, die Bedeutung der einzelnen Inhalte zu erkennen. Sie sind auch menschenlesbar, da sie durch Präsentationsanweisungen in einer klaren Präsentation bereitgestellt werden können. Die Präsentationsanweisungen können auch getrennt vom strukturierten Dokument vorliegen. In diesem Fall werden sie als Stilvorlage bezeichnet.

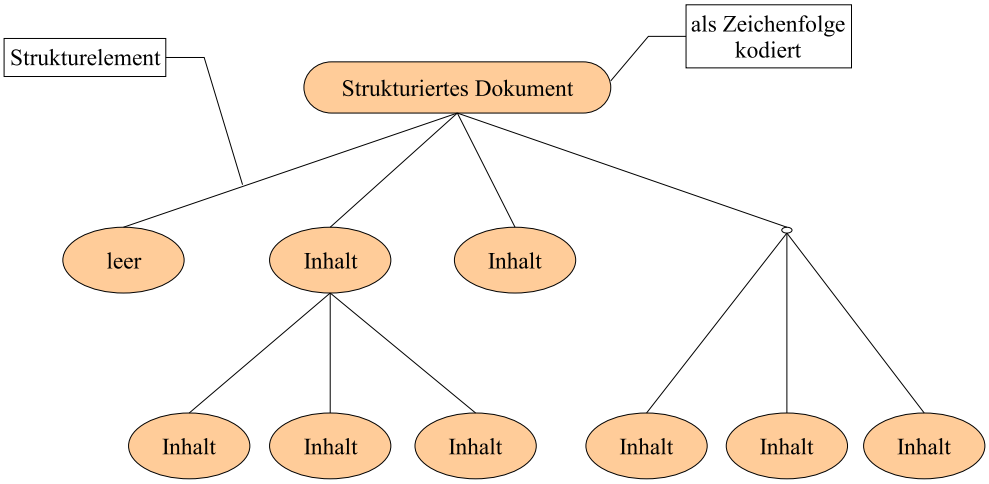

Abbildung 2.2: Aufbau strukturierter Dokumente

Inhalte und Struktur Strukturierte Dokumente enthalten nicht nur die eigentlichen Inhalte des Dokuments[2], sondern darüber hinaus auch Strukturelemente. In der Abbildung 2.2 sind die Strukturelemente als Kanten dargestellt. Diese in das Dokument eingebetteten Strukturelemente erweitern die Inhalte um eine semantische Ebene, mit deren Hilfe die Struktur des Dokuments und die Bedeutung der einzelnen Inhalte (z. B. „Definition“, „Satz“, „Beweis“) beschrieben werden kann. Strukturierte Dokumente bieten also eine semantische Selbstbeschreibung von Dokumenteninhalten. Diese Struktur eines Dokuments wird in den Dokumenten in einer Baumstruktur festgehalten, die sich aus der Beziehung der Strukturelemente zueinander ergibt.

Die Strukturelemente können leer sein, d. h. sie enthalten keine weitere Zeichenfolge. Wenn sie eine weitere Zeichenfolge enthalten, kann unterschieden werden zwischen Strukturelementen mit Unterstrukturelementen und Strukturelementen ohne Unterstrukturelemente. Die jeweils in den Strukturelementen enthaltenen Zeichenfolgen stellen die eigentlichen Inhalte des Dokuments dar. In der Abbildung 5.4 auf Seite 142 ist ein UML-Diagramm für den Aufbau von strukturierten Dokumenten angegeben.

Die Dokumente sind nicht nur rein linear vom Anfang bis zum Ende lesbar, sondern sie können über Verweise beliebig mit anderen Dokumenten verbunden sein. Diese Verweise, die für Hypertext-Dokumente entwickelt wurden, verweisen auf ein Sprungziel in einem anderen Dokument oder auch zu einer anderen Stelle im selben Dokument.

Das Modell der strukturierten Dokumente erleichtert auch den Zugriff auf die enthaltenen Daten eines Dokuments aus verschiedenen Programmiersprachen heraus. In syntaktischen Realisierungen des Modells der strukturierten Dokumente (z. B. XML) existieren verschiedene Schnittstellen, die diese Dokumentendaten abrufbar, verwendbar und editierbar machen (z. B. SAX[3] und

[1] Extensible Markup Language: `http://www.w3.org/XML/`
[2] Von jetzt an wird statt „strukturiertes Dokument“ einfach „Dokument“ geschrieben.
[3] Simple API for XML: `http://www.saxproject.org/`

DOM[4]).

Präsentation Da strukturierte Dokumente die Präsentationsvorschriften von den eigentlichen Inhalten trennen, wird die Struktur des Dokuments, das aus einzelnen Strukturelementen zusammengesetzt ist, nicht mehr durch die Präsentation vermittelt, sondern durch die explizite, deklarative Auszeichnung. Eine Stilvorlage in Form eines sogenannten „Stylesheets“ legt fest, wie ein bestimmter Inhalt grafisch dargestellt werden soll. Dabei verwendet es die eingebetteten Strukturelemente, um zu bestimmen, welche Form der Präsentation der Inhalt erhalten soll.

Die Dokumente können multimedial sein. Der Begriff „multimedial“ besitzt dabei zwei verschiedene Bedeutungen: Zum einen heißt „multimedial“, dass die Dokumente Bestandteile für mehrere Medien (Text, Grafik, Audio, Video) zu einem einzigen Dokument zusammenfassen können. Zum anderen bedeutet „multimedial“ auch, dass Dokumente:

- in verschiedenen Formaten (z. B. HTML, WML, XML, PostScript, PDF, ...) vorliegen können („Cross-Media Publishing“).
- über unterschiedliche Transportkanäle (z. B. HTTP, WAP, CD-ROM, ...) übertragen werden können („Multi-Channel Publishing“).
- auf unterschiedlichen Endgeräten (z. B. Computermonitor, Drucker, Beamer, PDA, Handy, Braille-Lesegerät, ...) genutzt werden können („Multi-Device Publishing“).

Dynamik und Interaktivität Als in den 1980er Jahren das Modell der strukturierten Dokumente eingeführt wurde, waren die Dokumente noch statisch-passiv. In der Zwischenzeit haben sie sich weiterentwickelt zu dynamisch-interaktiven Dokumenten.

Der Aspekt der Dynamik bedeutet, dass die Dokumente dynamisch ihre Inhalte oder ihre Präsentation ändern können. Dies kann z. B. abhängig vom Abfragezeitpunkt, von Inhalten einer Datenbank oder auch gesteuert durch vom Nutzer gesendete Parameter geschehen. Dynamische Dokumente sind beispielsweise sich regelmäßig aktualisierende Dokumente wie etwa Seiten mit Börsenkursen oder Wetterdaten. Die Dynamik kann zu verschiedenen Zeitpunkten und an verschiedenen Stellen stattfinden: auf der Seite des Servers, der das Dokument vor der Auslieferung eventuell aus verschiedenen Datenquellen aggregiert hat. Oder die Dynamik findet client-seitig bei den Stationen der Publikationskette statt, die der Endnutzung zuzurechnen sind. Die serverseitige Dynamik unterstützt dabei auch die Personalisierung der Dokumente, also die Anpassung an die speziellen Bedürfnisse eines bestimmten Benutzers.

Interaktive Dokumente verfügen über Bestandteile, mit denen die Nutzer interagieren können. Die Interaktion kann beispielsweise über Formulare stattfinden. Dabei können die vom Benutzer eingegebenen Formulardaten gleich auf der Client-Seite evaluiert und validiert werden. Eine Reihe von Beispielen für besondere Eigenschaften interaktiver Dokumente sind in [Pem05] formuliert. So können interaktive Dokumente den Benutzer durch auf- und zuklappbare Menüs (sogenannte „Outlines“) beim Lesen des Dokuments und beim Navigieren durch das Dokument un-

[4] W3C Document Object Model: `http://www.w3.org/DOM/`

terstützen. Dabei handelt es sich im wesentlichen um ein Auf- und Einfalten hierarchisch strukturierten Textes. Bei Klick auf unbekannte Wörter oder Begriffe oder auch ganze Sätze könnten Erklärungen oder Umformulierungen angezeigt werden. Der Nutzer kann Anmerkungen vornehmen. Interaktive Dokumente können sich an Benutzerbedürfnisse anpassen. Beispielsweise kann alternativer Text abhängig von den Interessen angezeigt werden (z. B. andere Maßeinheiten (Liter, Gramm statt pints, ounces, ...). Es könnte eine Filterung nach Randbedingungen stattfinden. Bei einem Kochbuch z. B. nach vorhandenen Zutaten oder verfügbarer Zeit. Um beim Beispiel mit dem Kochbuch zu bleiben: Es könnte eine fertige Einkaufsliste für ein ganzes Menü von Gerichten oder eine Kochanleitung für die parallele Zubereitung der Gerichte zusammengestellt werden. Es könnte Expertenwissen integriert werden, wodurch eine Bewertung dessen vorgenommen wird, was der Nutzer auswählt bzw. zusammenstellt.

2.3.2.2 Ebenen von strukturierten Dokumenten

Man kann den Aufbau eines strukturierten Dokuments als Baum darstellen (vgl. Abbildung 2.2). Jedes Element ist ein Knoten. Auch der Inhalt, der von einem Element umschlossen wird, ist ein Knoten. Wenn der Inhalt keine weiteren Elemente umfasst, dann ist er ein Blatt. Die Kanten entsprechen einer „Ist-Enthalten“-Relation.

Dieser Strukturbaum besitzt unterschiedliche Ebenen, entsprechend der Tiefe des Pfads von der Wurzel zu einem Knoten im Baum. Bei genauer Betrachtung der Elemente in den Knoten stellt man sehr oft fest, dass sich die Elemente mit kürzerem Abstand von der Wurzel in ihrer Qualität deutlich unterscheiden von den Elementen mit längerem Abstand (vgl. 2.3.2.1). Es lassen sich dabei folgende drei unterschiedliche Typen herausarbeiten[MA96]:

- Dokumentenhierarchie
  - Modellierung der Komponenten des gesamten Dokuments
  - Modellierung der Komponenten der Metainformationen
- Informationseinheiten
- Elemente der Datenebene

Der Typ von Elementen, die der Wurzel des Strukturbaums am nächsten sind, beschreibt die grobe Struktur des Dokuments (etwa Kapitel, Unterkapitel, Inhaltsverzeichnis) und enthält Metainformationen über das Dokument (etwa Autorennamen, Stichwörter). Diese Elemente geben dem Dokument auch seine charakteristische Struktur, die für alle Dokumente gilt, die zu der gemeinsamen Dokumentenklasse gehören.

Autoren haben bei der Dokumentenerstellung für gewöhnlich nur wenig Auswahl, was die Elemente der höheren Schichten betrifft. Sie haben jedoch meist die freie Gestaltungsmöglichkeit, wie tief diese grundlegenden Unterteilungselemente im gesamten Strukturbaum verschachtelt werden.

Die Ebene der Dokumentenhierarchie endet dort, wo „unspezifizierter Text“ beginnt aufzutauchen. Hier besitzen die Autoren dann großen Freiraum bei der Strukturierung und Zusammensetzung der Inhalte.

Die Ebene der Informationseinheiten umfasst demnach die Elementtypen, aus denen die Autoren nach ihrem eigenen Gutdünken auswählen können, um den „unspezifizierten Text“ auszuzeichnen, zu organisieren und zusammenzusetzen. Diese Ebene legt also eine Struktur fest, die sich etwa als Kenntlichmachung einzelner Absätze etc. manifestiert.

Eine Informationseinheit ist ein Element, das noch relativ abstrakt ist und in gewissem Maße auch für sich alleine genommen vom Leser verstanden werden kann. Ein solches Element verbindet den Inhalt zu einer geschlossenen Einheit, die bei der Informationsverarbeitung und -zusammenstellung unzerteilt bleiben muss. Informationseinheiten besitzen üblicherweise eine komplexe interne Struktur und sollten daher unter Beachtung ihrer Unterelemente modelliert werden. Zum Beispiel würden alle Unterelemente, die zu einer Liste gehören, zusammen modelliert werden.

Im Gegensatz dazu, sind Elemente der Datenebene kleine und wichtige Informationsfragmente, die aus bestimmten Gründen anders als der umgebende Text behandelt werden müssen. Diese Informationsfragmente wären für sich genommen ohne ihren Kontext ohne Bedeutung und haben so gut wie nie eine komplexe interne Struktur sondern sind meist sehr einfach aufgebaut.

Mögliche Beispiele für Elemente der Datenebene sind Markierungen zur Hervorhebung von einzelnen Wörtern oder Sätzen, etwa Dateinamen oder Produktnamen. Ferner können sie dazu dienen, Begriffe etwa mit ihrer Definition zu verknüpfen, oder bestimmte Teile des Dokuments aus diesen Datenelemente zusammenzustellen, beispielsweise Indizes.

2.3.2.3 Vorteile strukturierter Dokumente

Die Vorteile von strukturierten Dokumenten ergeben sich aus der Trennung der Präsentation vom Inhalt und der Struktur auf der einen Seite und der automatisierbaren Verarbeitbarkeit auf der anderen Seite. Im Folgenden werden einige Vorteile dieser beiden großen Bereiche genannt.

Abtrennung der Präsentation von Inhalt und Struktur Strukturierte Dokumente bieten mit ihren oben beschriebenen Eigenschaften eine Reihe von Vorteilen. Durch die Trennung von Inhalt und Struktur auf der einen Seite und der Präsentation auf der anderen Seite, müssen sich die Autoren der eigentlichen Inhalte nicht darum kümmern, wie die Inhalte präsentiert werden sollen. Das kann von spezialisierten Grafikdesignern erledigt werden, die sich ihrerseits nicht um die Erstellung des Inhalts zu kümmern brauchen. Sowohl die Autoren als auch die Layouter können sich somit auf ihr Fachgebiet konzentrieren. Bei diesem Ansatz, bei dem die unterschiedlichen Zuständigkeitsbereiche voneinander getrennt werden, handelt es sich um ein grundsätzliches Informatik-Prinzip, das sogenannte „Separation of Concerns“.

Als weiterer Vorteil kann eine beliebige Anzahl von Dokumenten mit einer einzelnen Stilvorlage in einer einheitlich Form präsentiert werden. Auch wirken sich Änderungen in der Stilvorlage automatisch auf alle Dokumente aus, ohne dass jedes Dokument geändert werden muss. Die Stilvorlage kann automatisiert auf alle relevanten Dokumente angewendet werden. Daneben kann ein einzelnes Dokument auch mit verschiedenen Stilvorlagen kombiniert werden, um es abhängig vom Verwendungszweck an unterschiedliche Benutzerbedürfnisse anzupassen. Ein Beispiel

hierfür ist die Anpassung an den Ausgabekanal (z. B. Bildschirm, Drucker, Handheld-Gerät oder auch Braille-Lesegerät). Mit einem geeigneten Dokumentenmodell kann der Benutzer zusätzlich eigene Präsentationsanweisungen festlegen, die gegenüber den Präsentationsanweisungen der Autoren Vorrang besitzen und somit die Kontrolle über die Präsentation zwischen der Autorenseite und der Nutzerseite aufteilen.

Es ist also möglich, Dokumente ohne spezielle Präsentation und medienneutral zu erstellen, zu bearbeiten, zu speichern, zu übertragen und erst unmittelbar zum erforderlichen Zeitpunkt die Stilvorlage an das Dokument zu binden. Dieses Prinzip findet sich in der Informatik wieder als sogenanntes „Late Binding“. Das Late Binding verlagert die Rechenbelastung zum Client. Dies kann in bestimmten Situationen ein Vorteil sein, wenn etwa viele Endnutzer auf einen Server zugreifen und dieser sonst einen deutlich höheren Aufwand zur Berechnung der jeweiligen Präsentation leisten müsste.

Zusammen mit der automatisierten Bearbeitbarkeit folgt auch die Möglichkeit, Dokumente zu personalisieren, also an die Bedürfnisse des jeweiligen Nutzers anzupassen. Desweiteren erleichtert es die Kombinierbarkeit von unterschiedlichen Dokumenten zu einem neuen Dokument und damit die Wiederverwendung von Inhalten.

Automatisierte Verarbeitbarkeit Durch die Kenntlichmachung der Struktur eines Dokuments wird die computergestützte und daher auch automatisierbare Verarbeitbarkeit der Inhalte ermöglicht. Dies betrifft unterschiedliche Bereiche wie Speicherung, Verwaltung, Generierung, Umwandlung, Weiterverarbeitung, Nutzung, Kombination und Interpretation der Dokumente. Die Heraustrennung der eigentlichen Präsentationsanweisungen ist dafür zwar nicht notwendig, sie erleichtert die computergestützte Bearbeitung jedoch und unterstützt die automatisierte Umwandlung von Dokumenten in unterschiedliche Formate (z. B. HTML, WML und PDF). Dadurch können Dokumente über unterschiedliche Medien publiziert werden.

Die Unterteilung des gesamten Dokuments in einzelne Struktureinheiten erleichtert auch die Übersetzung eines Dokuments in unterschiedliche Sprachen. Bei nachträglichen Änderungen im Originaldokument können die geänderten Struktureinheiten automatisiert erkannt und als erneut zu übersetzen markiert werden. Es ist sogar möglich, eine beliebige Anzahl von Sprachfassungen in einem einzigen Dokument zu speichern und zu übertragen. Durch die Kennzeichnung der semantischen Bedeutung von Inhalten könnte das Programm auf der Seite des Endnutzers genau nur die Elemente anzeigen, die zu der Sprache passen, die der Benutzer ausgewählt hat.

Dieses Verhalten kann nicht nur benutzt werden, um zwischen unterschiedlichen Sprachen umzuschalten. Auf die gleiche Art und Weise könnte z. B. ein Handbuch Text für alle Varianten einer Software enthalten und nur den gewünschten Teil anzeigen. Oder der Benutzer könnte zwischen der Anzeige eines Textes, der Anmerkungen zum Text und beidem gleichzeitig hin und her wechseln [Bos97].

Durch die semantischen, deklarativen Strukturauszeichnungen können die Rollen der verschiedenen Teile eines Dokuments klar angezeigt werden. Diese semantischen Zusatzinformationen können bei der späteren Verwendung des Dokuments vorteilhaft eingesetzt werden. Zum Beispiel können Indexierungswerkzeuge den Wörtern aus der Zusammenfassung eines Dokuments

ein höheres Gewicht beimessen [KR04]. Generell können Suchwerkzeuge bessere Ergebnisse liefern, da die Struktur eines Dokuments beschrieben ist und in die Suche einbezogen werden kann.

Strukturierte Dokumente werden in digitalen Bibliotheken, Datenbanken und Repositories abgespeichert, d. h. sie müssen für unterschiedliche Recherche- und Information-Retrieval-Methoden sowie Informationsdienste erschlossen werden. Strukturierte Dokumente unterstützen durch ihre logischen Strukturdaten die Dokumentenverwaltung wie etwa die automatische Erzeugung von Katalogeinträgen und Verzeichnissen oder das Information-Retrieval nach bestimmten logischen Kriterien.

Die Hypertextfähigkeiten ermöglichen es durch das Verknüpfen zwischen verschiedenen Dokumenten von einer linearen Lesart der Dokumente abzuweichen, bestimmte Abschnitte mit vertiefenden oder ähnlichen Dokumenten zu verbinden etc. und unterstützen so die Bildung eines Netzwerks von miteinander verknüpften Daten. Aus der Struktur dieses Netzwerks können wiederum weitere Informationen abgeleitet werden, die z. B. für die Bewertung der Relevanz eines Dokuments verwendet werden können, wenn nach bestimmten Begriffen gesucht wird (vgl. Googles PageRank Algorithmus[5]).

Abstraktion von der Speicherstruktur Die Dokumente bzw. die Bestandteile der Dokumente einer Publikationsanwendung sind irgendwo abgespeichert. Die Art und Weise, wie diese Datenhaltung realisiert ist und wie die Dokumente bzw. die Bestandteile der Dokumente zugreifbar sind, soll als Speicherstruktur bezeichnet werden. Die Möglichkeit der Abstraktion von dieser Speicherstruktur, stellt eine logische Zwischenschicht dar, die es ermöglicht, auf Dokumente bzw. Dokumentenbestandteile zuzugreifen und sie zu verwenden ohne dass die genaue Realisierung der Speicherstruktur bekannt sein muss.

Diese Abstraktion ist vergleichbar mit der logischen Datenunabhängigkeit bei Datenbanksystemen. Hier kann mit Hilfe von sogenannten Views eine Abstraktion von den definierten Tabellen vorgenommen werden. Die Daten, die in einem View zusammengefasst werden, können aus verschiedenen Tabellen stammen. Der Benutzer eines View muss sich jedoch nicht darum kümmern. Die Tabellen des View können sich ändern, ohne dass sich der View ändert.

Für den Bereich dieser Arbeit soll die Abstraktion am Beispiel des World Wide Web verdeutlicht werden: Am Anfang der Entwicklung des World Wide Web war die Speicherstruktur sehr einfach: alle Dokumente (in dem Fall HTML-Dateien) lagen in einer Verzeichnisstruktur vor. Der Web-Server konnte anhand der aufgerufenen URL erkennen, welche HTML-Datei angefordert wurde. In diesem Fall liegt somit keine Abstraktion von der Speicherstruktur vor. Dies führte v. a. bei komplexen Web-Sites (viele Dokumente, viele Verzeichnisse, viele Querverknüpfungen zwischen den Dokumenten) zu einer Reihe von Nachteilen:

- Es kann leichter zu Redundanzen in Form von mehrfach vorhandenen Dokumenten kommen.

[5] http://www.google.com/technology/

- Es ist schwierig, die Querverknüpfungen zu pflegen. Wenn ein Dokument verschoben wird, müssen alle Dokumente, die darauf verweisen, angepasst werden.
- Bei Umstrukturierungen ändern sich die Dateipfade und die Anforderung eines vor der Umstrukturierung gültigen URI funktioniert nicht mehr.

Ein Vorteil der strukturierten Dokumente liegt nun darin, dass sie es erleichtern, eine abstrahierende Zwischenschicht oberhalb der Speicherstruktur zu integrieren. Dadurch werden nicht nur die obigen Nachteile gemildert, sondern dies bietet weitere Vorteile:

- Erleichterung bei der Wieder- und Mehrfachverwendung ohne Redundanzen.
- Leichtere Austauschbarkeit von Dokumenten bzw. Dokumentenbestandteilen zwischen verschiedenen Publikationsanwendungen.

In moderner Website-Verwaltungssoftware verfügen die Dokumente daher über eine eindeutige Identifizierung.

2.3.3 Säulen des elektronischen Publizierens

Dokumente sind zentral für das elektronische Publizieren und folglich auch das Dokumentemodell, das den Aufbau der Dokumente beschreibt. Das Modell der strukturierten Dokumente teilt ein Dokument in drei wesentliche Bereiche auf: Inhalt, Struktur und Präsentation. Das elektronische Publizieren steht somit ganz wesentlich auf folgenden Säulen:

1. einer syntaktischen Realisierung des Modells der strukturierten Dokumente (=Meta-Sprache).
2. eines freien, aber formal beschreibbaren Vokabulars für die Auszeichnung der Struktur des Dokuments auf Basis der Meta-Sprache.
3. einer Möglichkeit zur Erzeugung von Präsentationen für ein Dokument.

Realisierung des Modells der strukturierten Dokumente

Die syntaktische Realisierung des Modells der strukturierten Dokumente legt fest, wie die Trennung von Inhalt, Struktur und Präsentation zu realisieren ist, z. B. beschreibt XML eine festgelegte Syntax für die Verbindung des Inhalts eines Dokuments und der dazugehörigen Struktur. Man bedient sich dabei sogenannter Elemente, die hierarchisch benannt und mit Attributen versehen sind. Dabei kann die Struktur eines konkreten Dokuments mit Hilfe weiterer Daten auf ihre Gültigkeit geprüft werden. Die Präsentationsanweisungen liegen ebenfalls in einer bestimmten Form vor – getrennt vom Inhalt und der Struktur.

Vokabular für die Strukturauszeichnung

Die Strukturauszeichnungen in einem Dokument gehören zu einem bestimmten Vokabular. Die Strukturelemente sollten zwar bei modernen Dokumenten frei wählbar und nicht auf einen begrenzten Vorrat festgelegt sein wie etwa HTML. Allerdings sollten die Strukturelemente standardisierbar sein, so dass für bestimmte Einsatzzwecke auf bekannte, bereits definierte Vokabularien zurückgegriffen werden kann. Dadurch wird auch die Austauschbarkeit und Kombinierbarkeit der Dokumente unterstützt, da alle Dokumente, die in dem gleichen Themengebiet angesiedelt sind, dieselben Kennzeichnungen für Strukturelemente verwenden können, auch wenn sie von verschiedenen Autoren erstellt wurden.

Ein Dokument muss nicht auf ein einziges Vokabular beschränkt sein. Der Begriff des Vokabulars kann durchaus umfassender sein und Kompositionsmöglichkeiten in der Form einschließen, dass innerhalb eines einzelnen Dokuments gleichzeitig verschiedene Vokabularien verwendet werden können.

Erzeugung der Präsentation

Da strukturierte Dokumente die Präsentation von der Struktur und dem Inhalt trennen, muss durch die syntaktische Realisierung des Modells der strukturierten Dokumente zum einen definiert sein, wie Präsentationvorschriften beschrieben werden können. Zum anderen muss es eine Möglichkeit geben, die Präsentationsanweisungen auf ein konkretes Dokument anzuwenden und dadurch die Präsentation des Dokuments zu erzeugen.

2.3.4 Die Publikationskette

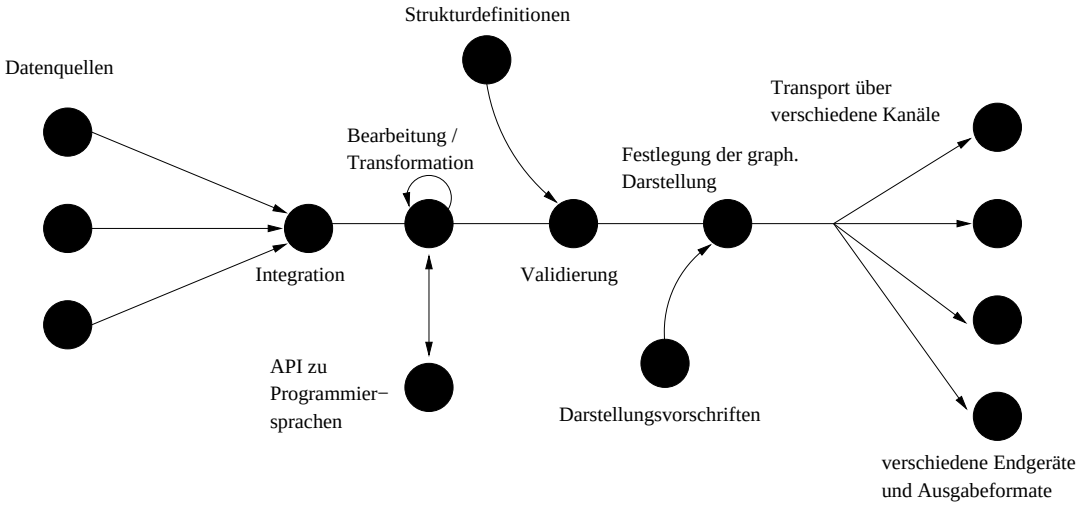

Abbildung 2.3: Beispielablauf einer Publikationskette

Dokumente bzw. die Bestandteile, aus denen sie zusammengesetzt sind, durchlaufen für ihre Verfügbarmachung eine Folge von Zwischenstationen. Diese Folge wird als „Publikationskette“ bezeichnet und ist in der Abbildung 2.3 schematisch dargestellt. In der klassischen Sicht des Publikationswesens eine streng lineare Ablauffolge. In der modernen Sicht ist diese strikte Form aufgelöst und wird sehr viel flexibler gesehen. Bei den Zwischenstationen kann es sich sowohl um computergestützte Prozesse handeln als auch um Menschen, die sich z. B. um die Begutachtung, die Aufbereitung oder die Bereitstellung der Dokumente kümmern. Als Endnutzer der Dokumente kommen wiederum sowohl Menschen als auch computergestützte Prozesse in Betracht.

In der Regel kann die Publikationskette auf unterschiedlichen Wegen durchlaufen werden: Es kann mehrere Datenquellen, eine Reihe von Bearbeitungsstationen und ebenso mehrere Ausgabeziele geben. Die Zwischenstationen müssen nicht von allen Dokumenten gleich durchlaufen werden. Dies kann abhängig von bestimmten Parametern geschehen. Diese Steuerparameter können sowohl aus den Datenquellen bzw. den Dokumente selbst als auch vom Benutzer oder dem Kontext stammen. Je komplexer diese Struktur aufgebaut ist, desto höher ist auch die Komplexität des Ablaufs und umso größer sind die Anforderungen an seine Modellierung und Strukturierung.

2.3.5 Publikationssysteme und Publikationsanwendungen

Das Modell der strukturierten Dokumente und die Publikationskette sind das konzeptuelle Fundament des elektronischen Publizierens. Von den bisherigen Realisierungen des Modells der strukturierten Dokumente hat der offene Standard XML bisher die mit Abstand größte Verbreitung gefunden und ein Ende der Durchdringung verschiedenster Bereiche ist noch nicht abzusehen.

Von diesen fundamentalen Konzepten werden vielfältige Nutzungsmöglichkeiten angeboten. Sie sind nicht von vornherein auf einen bestimmten Einsatzzweck beschränkt. Da sie noch auf einer recht hohen Abstraktionsstufe angesiedelt sind, ist für die tatsächliche Nutzung der Möglichkeiten eine Softwarebasis erforderlich, welche die Möglichkeiten praktisch realisiert und nutzbar macht. Eine solche Softwarebasis nennen wir ein „Publikationssystem“ und die Verwendung eines Publikationssystems nennen wir eine „Publikationsanwendung“.

Als Publikationssystem wird also ein Softwaresystem bezeichnet, das die Möglichkeiten der Konzepte einmal des Modells der strukturierten Dokumente und zum anderen der Publikationskette durch eine konkrete Realisierung dieser Konzepte nutzbar macht.

Als Publikationsanwendung wird der Einsatz eines Publikationssystems für einen bestimmten Anwendungszweck bezeichnet. Oder anders formuliert: Eine Publikationsanwendung ist eine spezialisierte Ausprägung der Nutzung der allgemeinen Möglichkeiten, die von einem Publikationssystem zur Verfügung gestellt werden.

Die Bandbreite der Verwendung des Publikationssystems kann dabei das gesamte Spektrum von der vergleichsweise einfachen Anpassung durch Konfiguration des Publikationssystems bis hin

zur Erweiterung des Publikationssystems um speziell programmierte Komponenten oder der Integration anderer Softwaresysteme abdecken.

2.3.6 Vergleich mit Softwareanwendungen

Ebenso wie Softwareanwendungen verwenden auch Publikationsanwendungen ein darunter liegendes Basissystem, das von bestimmten Aspekten der Systemumgebung abstrahiert. Bei Softwareanwendungen wird diese Basis Betriebssystem genannt, bei Publikationsanwendungen heißt die Basis Publikationssystem. Betriebssysteme abstrahieren von der verwendeten Hardware und verwalten Ressourcen, Benutzer und Prozesse. Publikationssysteme kümmern sich um die Ressourcen, Benutzer und Prozesse einer Publikationsanwendung.

Publikationssysteme stellen jedoch selbst eine Softwareanwendung in dem Sinne dar, als dass sie auf ein Betriebssystem aufsetzen und dessen Dienste nutzen. Publikationssysteme sind trivialerweise auf den Einsatz für das elektronische Publizieren spezialisiert und abstrahieren von spezifischen Aspekten der Publikationskette sowie des Modells der strukturierten Dokumente. Betriebssysteme sind dagegen nicht derartig eingegrenzt auf einen klar bestimmten Verwendungszweck.

Der wesentliche Unterschied zwischen Publikationsanwendungen und Softwareanwendungen besteht schließlich darin, dass Publikationsanwendungen hauptsächlich aus einer Menge von Konfigurationsparametern bestehen, die das zugrunde liegende Publikationssystem an den Einsatzzweck anpassen. Im Gegensatz dazu ist eine charakteristische Eigenschaft von Softwareanwendungen, dass sie zum größten Teil aus Rechenanweisungen bestehen. Die Rechenanweisungen führen Berechnungen durch, verändern die Werte von Speicherinhalten und beinhalten Ablaufstrukturen wie Schleifen und Fallunterscheidungen.

2.3.7 Übersicht

In der Abbildung 2.4 sind die wichtigen Konzepte, die in den vorangegangenen Abschnitten beschrieben wurden, dargestellt und zueinander in Beziehung gesetzt. Diese Beziehungen sollen hier kurz ausgeführt werden.

Im Zentrum steht das Modell der strukturierten Dokumente. Es beschreibt die Bestandteile von strukturierten Dokumenten, nämlich Inhalte und Strukturelemente. Die Strukturelemente bilden dabei eine Hierarchie über die Inhalte. Die syntaktische Realisierung legt fest, wie die strukturierten Dokumente kodiert werden. Strukturierte Dokumente sind dabei eine Spezialisierung der allgemeineren, nicht-strukturierten Dokumente. Eine Publikation fasst eine Anzahl von Dokumenten zusammen und legt gleichzeitig über die Ausgabeformate die Präsentation fest. Ein Dokument kann dabei in mehreren Präsentationen publiziert werden, was wiederum dazu führen kann, dass für unterschiedliche Präsentationen auch unterschiedliche Stilvorlagen zum Einsatz kommen. Dies gilt zumindest für den Fall, dass es sich um strukturierte Dokumente handelt. Unterschiedliche Präsentationen ziehen meist auch unterschiedliche Formate für die Kodierung

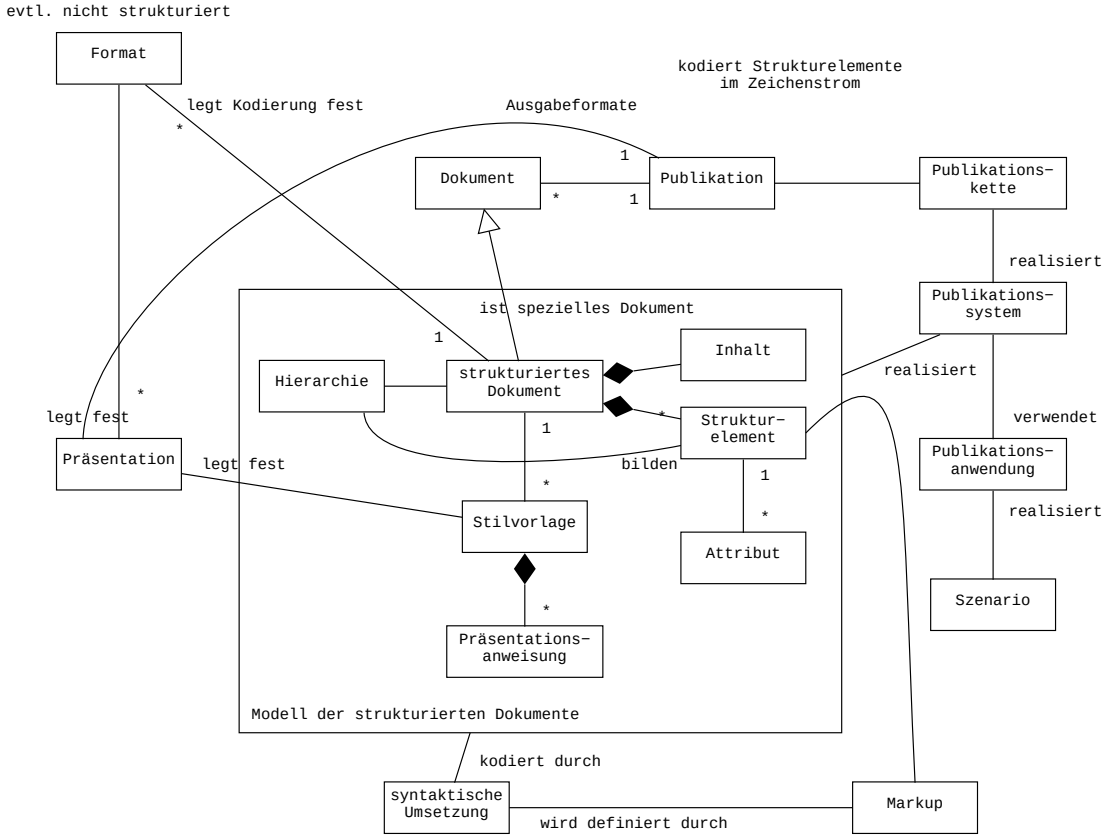

Abbildung 2.4: Konzepte des elektronischen Publizierens

nach sich. Der Unterschied zur syntaktischen Realisierung des Modells der strukturierten Dokumente besteht darin, dass strukturierte Dokumente nach der Formatierung nicht zwangsläufig noch strukturiert sind.

2.4 Einordnung und Abgrenzung

Neben dem Fachgebiet des elektronischen Publizierens existieren noch weitere Bereiche in der Forschung und der Praxis, die sich mit Dokumenten befassen. In [Kam03] wurde der Begriff der „Document Related Technologies“ eingeführt, der alle diese Bereiche zusammenfasst.

Zu diesen Bereichen gehören:

- Dokumentenmanagement
- Content Management
- Enterprise Content Management

- Elektronische Archivierung
- Knowledge Management

In dieser Arbeit wird vor allem der Begriff des elektronischen Publizierens verwendet. Damit ist jedoch nicht das gemeint, was ursprünglich im klassischen Verlagswesen damit assoziiert wurde, nämlich das einfache Verfügbarmachen von Verlagspublikationen (meistens Bücher und Zeitschriften) in elektronischer Form, womit meist die Bereitstellung auf Diskette, CD-ROM oder später auch im Internet gemeint war. Dieses inzwischen überholte Verständnis schränkt unnötig ein. Auch im Verlagswesen ist die Entwicklung weitergegangen, die nicht mehr durch den alten Begriff beschreibbar ist (vgl. das Szenario aus 3.2.8).

In dieser Arbeit geht es bei elektronischem Publizieren um Methoden, Modelle und Systeme, um Dokumente im Web verfügbar zu machen. Dabei kommt ein strukturiertes Dokumentenmodell zum Einsatz, d. h. wir beschäftigen uns mit Szenarien, die mit der Verfügbarmachung von Dokumenten im Internet zu tun haben. Softwaresysteme zur Realisierung solcher Szenarien sind in verschiedene Oberbegriffe aufgeteilt, die nachfolgend kurz erläutert werden.

2.4.1 Dokumentenmanagement

Wichtig ist, zunächst festzuhalten, dass im Dokumentenmanagement der Dokumentenbegriff wieder ähnlich ist wie im klassischen elektronischen Publizieren und dessen zentrale Eigenschaften sich folgendermaßen beschreiben lassen (vgl. [Kam03]):

> Aus Benutzersicht handelt es sich bei den Dokumenten [. . .] um eine einheitliche Einheit, die bei Bedarf lokalisiert, angezeigt oder abgespielt, editiert, gespeichert und wieder aufgefunden werden muss.

Ein Kritikpunkt hierbei ist insbesondere, dass die bei modernen strukturierten Dokumenten so wichtige Trennung von Inhalt, Struktur und Präsentation bei diesem Dokumentenbegriff nicht gegeben ist.

Dokumentenmanagement beschäftigt sich insgesamt mit der Verwaltung und Verarbeitung heterogener Dokumentensammlungen. Es ist entstanden aufgrund enorm wachsender Dokumentenbestände innerhalb von Organisationen und der damit verbundenen erschwerten Handhabung dieser Dokumentenbestände. Dokumentenmanagementsysteme sind in der Praxis auf die Bereitstellung von Dokumenten für Drittapplikationen sowie auf Auswertungsfunktionen spezialisiert und behandeln das einzelne Dokument im wesentlichen als „Black Box“.

2.4.2 Content Management

Pironet, ein Hersteller von Systemen für Content Management, definiert den Begriff „Content Management“ folgendermaßen:

> Content Management bezeichnet das professionelle Steuern aller Informationen und Prozesse im Intra-, Extra- und Internet auf der Basis einer umfassenden Redaktions- und Betriebsplattform. Durch die effektive Nutzung und Verteilung von Inhalten und Daten können Unternehmen ihre Kosten senken, Prozessabläufe verbessern, dynamische und personalisierte Web-Portale realisieren und die Kommunikation mit Mitarbeitern, Partnern und Kunden verbessern.

Diese Definition ist sehr grob gehalten und beschreibt eher die Ziele, die mit dem Einsatz eines Content Management Systems erreicht werden sollen, als die Eigenschaften von Content Management festzulegen. In [RR01] wird Content Management etwas genauer wie folgt definiert:

> [. . .] die systematische und strukturierte Beschaffung, Erzeugung, Aufbereitung, Verwaltung, Präsentation, Verarbeitung, Publikation und Wiederverwendung von Inhalten.

Beim Content Management stehen dabei weniger vollständige Dokumente im Vordergrund. Vielmehr geht es um die Erfassung und Verwaltung von Inhalten zusammen mit ihren Metadaten. Die Inhalte dienen als Bausteine, um aus ihnen ein Dokument zusammenzusetzen.

Inhalte (auch „Content“ genannt) können unterschiedlichste Formen annehmen. Der englische Begriff „Content“ steht für „Inhalt“, „Aussage“ oder „Gehalt“. Er wird seit Mitte der 1990er Jahre vor allem für mediale Inhalte im Internet und anderen Informationssystemen verwendet. Der Begriff dient vor allem zur Abgrenzung zwischen verwertbaren Informationen und Daten, die eher zu deren Verwaltung dienen [DKGS04]. Content sind Informationen, die im Internet in unterschiedlicher Form angeboten werden, z. B. Texte, Bilder, Musikstücke, Filme, Grafiken, Fotografien [KS05]. Content ist nicht anfassbar, und mittelbar oder unmittelbar von Menschen erzeugt.

Der sogenannte „Content Lifecyle“ umfasst vier Stationen: Content Creation → Content Review → Content Distribution & Presentation → Content Versioning & Archiving → wieder von vorne.

Untergebiete des Content Management sind Web Content Management, das sich spezialisiert auf die optimierte Verfügbarmachung von Inhalten im Web [VGB01] und Digital oder Media Asset Management, das sich fokussiert auf die Erfassung und Verwaltung möglichst vieler verschiedener Typen von Inhalten.

Content Management Systeme beherrschen im Unterschied zu Web Content Management Systemen neben der Verfügbarmachung der Dokumente in Formaten, die primär für die Präsentation mit Hilfe von Browsern gedacht sind, auch die Konvertierung in andere Formate, deren Hauptübertragungsweg nicht das Internet sein muss (z. B. PDF).

In der Abbildung 2.5 ist die grobe Aufteilung des Content Management in die drei verschiedenen Bereiche „Erstellen des Content“, „Content Verwaltung“ und „Präsentation“ dargestellt. Der Vorgang des „Publishing“ kann in Form einer Publikationskette wie in Abschnitt 2.3.4 modelliert werden.

Content Management Systeme lassen sich eindeutig dadurch von Dokumentenmanagement-Systemen abgrenzen, dass bei ihnen – im Gegensatz zu Letzteren – die Trennung von Inhalten

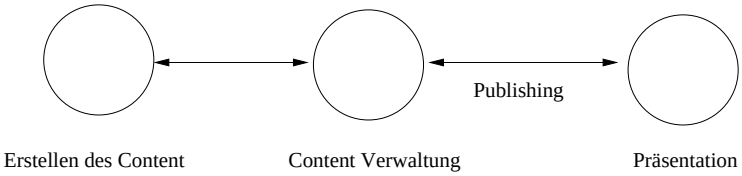

Abbildung 2.5: Bereiche des Content Management

und Präsentationsanweisungen eine zentrale Eigenschaft ist (vgl. Abschnitt 2.4.1). Dies lässt sich auf die Formel reduzieren: „Content Management = Document Management + Trennung Inhalt/Präsentation“.

Es gibt verschiedene Typen von Content Management Systemen:

Serverseitiges Content Management System:
Ein serverseitiges Content Management System verwaltet die Inhalte an einer zentralen Stelle. Der Zugriff auf die Inhalte, um diese zu speichern, zu editieren oder zu verwalten, läuft ortsunabhängig von Clientarbeitsplätzen aus. Meist reicht ein Web Browser dafür aus, so dass der Zugriff auch von überall auf der Welt erfolgen kann. Dadurch können mehrere Nutzer die Inhalte verwalten. Zur Verwaltung der Benutzer verfügen serverseitige Content Management Systeme meist über eine Möglichkeit zur Verwaltung benutzerspezifischer Berechtigungen.

Clientseitiges Content Management System:
Ein clientseitiges Content Management System wird mit Hilfe eines Programms gesteuert, das auf einem Clientarbeitsplatz installiert ist. Die Inhalte werden von diesem Arbeitsplatz auf den Server übertragen. Die Verwaltung der Inhalte geschieht also immer über diesen einen Arbeitsplatz. Dies ist dann vorteilhaft, wenn Inhalte mit großem Umfang (etwa Videos) bearbeitet werden sollen. Wenn dies mehrfach geschieht, müssten bei einem serverseitigen Content Management System die Inhalte jedes mal wieder zum Arbeitsplatz übertragen werden, falls eine Bearbeitung direkt auf dem Server nicht möglich ist.

Mischung aus beiden Typen:
Es existieren auch Content Management Systeme, die eine Mischung aus beiden anderen Typen sind. Dabei ist beispielsweise die Verwaltung der Inhalte auf dem Arbeitsplatz möglich, aber nur für Bereiche, für die man über eine Berechtigung verfügt.

Ferner lassen sich Content Management Systeme unterscheiden nach der Art der Auslieferung der erstellten Dokumente an den Nutzer:

Volldynamische Systeme:
Volldynamische Content Management Systeme berechnen angeforderte Dokumente bei jeder Anforderung erneut. Vorteile dabei sind z. B., dass die Inhalte immer dem aktuellen Stand entsprechen und eine Personalisierung der ausgelieferten Dokumente einfacher durchgeführt werden kann. Dies geht jedoch zu Lasten der Reaktionszeit des Servers.

Statifizierende Systeme:
Statifizierende Content Management Systeme berechnen Dokumente vollständig im voraus und legen sie für die zukünftige Auslieferung ab. Die Vor- und Nachteile dieses Verfahrens sind umgekehrt zu den Vor- und Nachteilen volldynamischer Content Management Systeme: die Reaktionszeit des Servers ist optimal, jedoch sind die Dokumente eventuell nicht ganz aktuell und auch eine Personalisierung ist nur eingeschränkt möglich: Bei HTML-Dokumenten z. B. können noch relativ einfach andere Stilvorlagen mit dem Dokument verknüpft werden. Eine inhaltliche oder strukturelle Umstellung des Dokuments ist dagegen nicht möglich.

Hybride Systeme:
Hybride Content Management Systeme verbinden die Vorteile sowohl der volldynamischen als auch der statifizierenden Content Management Systeme. Nur Dokumente und Inhalte, die dynamisch zum Anforderungszeitpunkt zusammengestellt bzw. erzeugt werden müssen, werden weiterhin erst auf Anforderung aus der Datenbank extrahiert. Alle anderen Dokumente und Inhalt liegen bereits statisch vor.

2.4.3 Enterprise Content Management

Das Enterprise Content Management ist aus dem Dokumentenmanagement entstanden (vgl. [Kam03]). Die Definition von Enterprise Content Management ist nach [AII06]:

> Enterprise Content Management is the technologies used to Capture, Manage, Store, Preserve, and Deliver content and documents related to organizational processes.

Der verwendete Dokumentenbegriff ist folglich derselbe wie im Dokumentenmanagement (vgl. Abschnitt 2.4.1). Die Definition ist sehr ähnlich zur Definition von Content Management von **??** im Abschnitt 2.4.2. Der Hauptunterschied zu Content Management liegt darin, dass Enterprise Content Management explizit auf die Anforderungen in Unternehmen eingeht.

2.4.4 Elektronische Archivierung

Bei der elektronischen Archivierung geht es ebenfalls um Dokumente im Sinne von Dokumentenmanagement. Das Hauptanliegen besteht dabei in der dauerhaften und gesicherten Archivierung der Dokumente.

2.4.5 Knowledge Management

Knowledge Management zielt darauf ab, das in Organisationen vorhandene Wissen bestmöglich zur Erreichung der Ziele der Organisation einzusetzen und zu entwickeln. Knowledge Management ist also inhaltlich orientiert. Es steht das Einsetzen und Entwickeln von Wissen im Vordergrund. Um dies zu ermöglichen, ist es zunächst erforderlich, Wissen festhalten und wiederfinden zu können.

Ein Bereich des Knowledge Management, der von der Informatik unterstützt wird, ist das Ableiten von neuem Wissen, d. h. das Ziehen von Schlüssen basierend auf bereits bestehendem Wissen durch logische Kalküle. Um das vorhandene Wissen einer Organisation möglichst vollständig zu erfassen, ist auch der Datenbestand interessant, der in Systemen gespeichert ist, die sich mit der Verwaltung von Content und Dokumenten befassen. Dies bedeutet, dass sich Knowledge Management und entsprechende Systeme (z. B. für das Content Management) einander annähern werden.

Für mehr Informationen zu diesem Bereich sei auf [NT95, PRR99] sowie auch auf [Weg02] verwiesen.

2.4.6 Fazit

Die Zuordnung eines Softwaresystems zu den in den vorangegangenen Abschnitte kurz beschriebenen fünf Bereichen ist nicht immer eindeutig. Viele Softwaresysteme decken in steigenden Maße nicht nur die Anforderungen eines einzigen Bereichs, sondern mehrerer Bereiche ab. Außerdem sind auch die Grenzlinien zwischen den Bereichen nicht immer eindeutig und scharf zu ziehen. Gerade Bereiche, bei denen eine rege Weiterentwicklung stattfindet, fallen darunter (z. B. Content Management und Knowledge Management).

Publikationssysteme können prinzipiell aus jedem dieser Bereiche kommen, wenn sie mit strukturierten Dokumenten umgehen und die im Folgenden noch vorgestellten Anforderungen an Publikationssysteme erfüllen. Es gibt bestimmte Merkmale, deren Erfüllung charakteristisch ist für Publikationssysteme und die zur Abgrenzung herangezogen werden können. Diese Merkmale sind:

Unterstützung der Wiederverwendung von Dokumentenbestandteilen:
Damit ist eine „Komponentenorientierung“ der Dokumente gemeint. Dokumente sind nicht monolithisch, sondern setzen sich aus Bestandteilen zusammen. Diese Bestandteile können andere Dokumente oder Teile davon sein. Eine wesentliche Eigenschaft von Publikationssystemen ist die Unterstützung der Aggregierung eines Dokuments aus Teilen anderer Dokumente.

Aggregierung mehrerer Datenquellen:
Dies bedeutet, dass für die Zusammenstellung von Dokumenten die Daten auch aus unterschiedlichen Quellen (Dateisystemen, Datenbanken, Content-Repositories) stammen können. Diese werden selektiert, kombiniert und das Ergebnis als ein Dokument publiziert. Diese Eigenschaft lässt sich mit dem Schlagwort „Drehscheibenfunktionalität“ erfassen.

2.5 Anforderungen an Publikationssysteme

Es gibt im elektronischen Publizieren unterschiedliche Ebenen. Auf der abstrakten Ebene sind das Modell der strukturierten Dokumente (vgl. Abschnitt 2.3.2) und das Konzept der Publikationskette angesiedelt. Die zweite, konkrete Ebene umfasst die Realisierungen des Dokumentenmodells und der Publikationskette durch das jeweilige Publikationssystem. Die abstrakte Ebene

definiert die Anforderungen an die konkrete Ebene. Daher lassen sich die Anforderungen an Publikationssysteme aufteilen in Anforderungen an die Realisierung des Dokumentenmodells und an die Unterstützung der Publikationskette. Diese Anforderungen müssen erfüllt sein, um das volle Potential strukturierter Dokumente nutzen zu können.

In der Fachliteratur werden die Anforderungen an solche Systeme in der Regel nach den Funktionsbereichen des elektronischen Publizierens unterschieden (Erstellen der Inhalte, Verwalten der Inhalte, das Präsentieren der Inhalte, ...). In dieser Arbeit wird eine andere Perspektive eingenommen, die auf der Überzeugung beruht, dass das Dokumentenmodell einen ganz wesentlichen Einfluss besitzt. Daher werden die Anforderungen ausgehend von der Betrachtung der Möglichkeiten des Dokumentenmodells beschrieben. Die Abbildung 2.6 ordnet die einzelnen Anforderungen den verschiedenen Stationen in der Publikationskette zu.

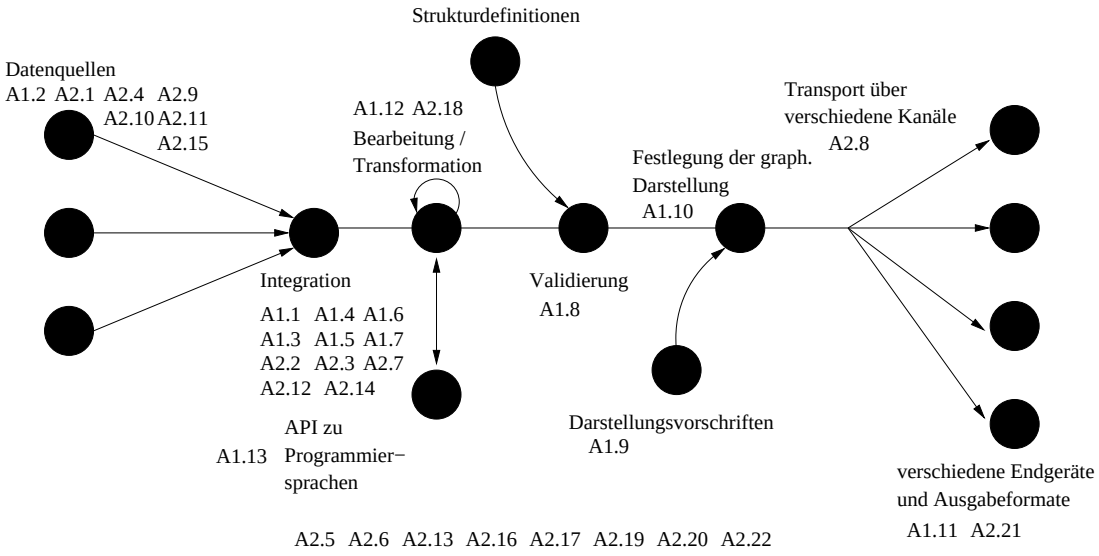

Abbildung 2.6: Einordnung der Anforderungen

2.5.1 Anforderungen an die Realisierung des Dokumentenmodells

Strukturierte Dokumente besitzen bestimmte Eigenschaften und Vorteile. Um diese Eigenschaften und Vorteile im Einklang mit den allgemeinen Zielen des elektronischen Publizierens auszunutzen, muss die Realisierung des Modells durch das Publikationssystem eine Reihe von Anforderungen erfüllen. Diese Anforderungen ergeben sich auch aus der Art und Weise, wie strukturierte Dokumente die Stationen einer Publikationskette durchlaufen. In der Abbildung 2.3 ist ein solcher Beispielablauf schematisch dargestellt.

A1.1 Dynamisches Zusammenstellen der Inhalte:

Am Anfang der Publikationskette wird das Dokument – bzw. die Daten, die es enthält – erzeugt. Dies könnte statisch geschehen, d. h. das Dokument könnte zu einem frühen

Zeitpunkt einmal zusammengestellt werden und liegt danach in dieser Form für die weiteren Bearbeitungsschritte in der Publikationskette bereit. Für jeden erneuten Durchlauf der Kette wird wieder auf diese ursprüngliche Version des Dokuments zurückgegriffen.

Mit dieser statischen Erzeugung der Inhalte lässt sich jedoch keine dynamische Anpassung der Daten an die Bedürfnisse der Benutzer erreichen. Dieses Vorgehen ist auch dann nicht sinnvoll, wenn sich die Daten häufig ändern und daher die Dokumente oft angepasst werden müssen. In einem solchen Fall ist es häufig effizienter, die Dokumente dynamisch erst zu dem Zeitpunkt zusammenzustellen, an dem sie angefordert werden.

A1.2 Unterstützung verschiedener Datenquellen:
Im vorherigen Punkt wurde beschrieben, dass ein dynamisches Zusammenstellen der Inhalte möglich sein muss. Diese Inhalte können aus unterschiedlichen Quellen stammen: Datenbanken, Dateien, den Ergebnissen von Programmaufrufen und auch anderen Dokumenten. Es muss möglich sein, Inhalte gleichzeitig aus verschiedenen Datenquellen zu einem Dokument zusammenzustellen.

A1.3 Adressierbarkeit von Dokumenten:
Manchmal sollen Dokumente vollständig wiederverwendet werden, manchmal sind nur bestimmte Teile eines bestehenden Dokuments relevant und interessant. Der Prozess des elektronischen Publizierens muss es ermöglichen, dass auch nur bestimmte, einzelne Teile eines Dokuments wiederverwendet werden können. Dazu müssen diese Teile von außen gezielt ansprechbar (adressierbar) sein.

A1.4 Verknüpfbarkeit von Dokumenten:
Es ist nicht immer sinnvoll, andere Dokumente oder Teile von anderen Dokumenten als Kopie in ein bestimmtes Dokument einzubetten. In vielen Fällen sollen die bestehenden Teile eines Dokuments mit den Daten in anderen Dokumenten nur verknüpft werden. Diese Möglichkeit zur Verknüpfbarkeit ist ein wichtiger Bestandteil des World Wide Web, dessen Verknüpfungsmöglichkeiten jedoch nur recht einfach gestaltet sind. Durch Verknüpfungen können Assoziationen und Navigationsstrukturen aufgebaut werden. Ein elektronisches Dokument kann sich damit deutlich von den gedruckten, linearen Medien unterscheiden. Solche Verknüpfungen müssen stabil sein gegenüber Veränderungen wie etwa Restrukturierungen der Dokumente.

A1.5 Unterstützung multimedialer Komponenten:
Moderne Dokumente sind aus mehreren unterschiedlichen Bestandteilen zusammengesetzt. Dabei enthalten sie nicht nur Text sondern auch multimediale Komponenten wie etwa Grafiken, Videos oder auch Programmcode, z. B. für aktive Bestandteile wie etwa Animationen. Dementsprechend muss der Prozess des elektronischen Publizierens auch Dokumente unterstützen, die multimediale Bestandteile besitzen.

A1.6 Unterstützung dynamischer Dokumente:
Dynamische Dokumente passen sich automatisch an bestimmte Rahmenbedingungen (variable Umgebungsparameter) an. Sie können dabei sowohl den Inhalt als auch ihre Präsentation ändern. Die Grundlage der Anpassung kann zum einen der Benutzer und zum anderen der Zeitpunkt sein, zu dem das Dokument abgerufen wird. Dieser Punkt ist verwandt mit der Anforderung, dass Inhalte dynamisch zusammengestellt werden können

müssen.

Die Dynamik bietet die Möglichkeit zur „Just-In-Time“-Bereitstellung von Dokumenten. Das ist zum einen hilfreich für die Personalisierung von Dokumenten, zum anderen für die möglichst zeitnahe Bereitstellung von variablen Daten (z. B. Wetter-, Börsen-, Sensordaten).

Die Dynamik kann auf unterschiedliche Weise realisiert sein. Eine Variante ist durch Java Servlets[6] geben. Hier werden nicht nur die dynamischen Bestandteile sondern auch die statischen Bestandteile durch den Programmcode ausgegeben. Im Gegensatz dazu ist der Ansatz von JavaServer Pages[7] genau umgekehrt: Statische Bestandteile werden ergänzt durch Bestandteile, die dynamisch eingesetzt werden.

A1.7 Unterstützung interaktiver Dokumente:
Dokumente können interaktiv sein. Sie können auf Eingaben des Benutzers reagieren (z. B. in Form von Formularen oder durch aufklappbare Sichten).

A1.8 Definition und Validierung eines gemeinsamen Vokabulars:
Um ein Dokument aus mehreren anderen Dokumenten zusammenzustellen und um Dokumente von verschiedenen Autoren erstellen lassen zu können, ist es sinnvoll, ein gemeinsames Vokabular für die Strukturauszeichnungen festzulegen, das von allen Dokumenten verwendet wird. Dadurch kann eine ganze Klasse von Dokumenten beschrieben werden, die eine gleichartige Struktur besitzen. Zusammen mit einer computerlesbaren Beschreibung des Vokabulars und damit des möglichen Strukturaufbaus eines Dokuments kann dessen Konformität automatisch überprüft werden.

A1.9 Unterstützung externer Präsentationsanweisungen:
Diese Anforderung leitet sich direkt aus dem Modell der strukturierten Dokumente ab. Durch die Trennung von Inhalt und Struktur auf der einen Seite und der Präsentation auf der anderen Seite, muss auch der Prozess des elektronischen Publizierens vom Dokument getrennt vorgehaltene Präsentationsanweisungen unterstützen. Dabei muss in geeigneter Weise festgehalten werden können, welche Präsentationsanweisungen mit einem bestimmten Dokument assoziiert sind. Dies kann beispielsweise durch einen Verweis im Dokument geschehen (vgl. HTML und CSS) oder auch mit Hilfe einer getrennten Zuordnungsdatenbank.

A1.10 „Late Binding“ von Präsentationsanweisungen:
Zur Personalisierbarkeit der Dokumente gehören verschiedene Aspekte. Zum einen können die enthaltenen Daten individuell auf die Bedürfnisse des Nutzers zugeschnitten sein (z. B. bei Online-Zeitungen). Zum anderen kann die Präsentation der Daten personalisiert sein. Hierfür muss es möglich sein, dass die Präsentationsanweisungen erst kurz vor der Nutzung auf das Dokument angewendet werden können.

A1.11 Unterstützung verschiedener Ausgabeformate:
Elektronische Dokumente können für die Präsentation in unterschiedliche Formate umgewandelt werden. Dies wird nicht zuletzt durch das Late Binding von Präsentationsanwei-

[6] Java Servlet Technology: `http://java.sun.com/products/servlet/`
[7] JavaServer Pages Technology: `http://java.sun.com/products/jsp/`

sungen unterstützt. Die Möglichkeit, ein einzelnes Ursprungsdokument in verschiedene Ausgabeformate zu transformieren wird auch als „Mehrfachverwendung“ oder „Multiple Usage“ oder auch „Cross-Media Publishing“ bezeichnet (vgl. [Sch01], [GHK00]).

A1.12 Unterstützung automatischer Bearbeitungsschritte:
Eine der wesentlichen Eigenschaften moderner, strukturierter Dokumente ist die Computerlesbarkeit und insbesondere die Computerverarbeitbarkeit. Die Daten, die in einem Dokument enthalten sind, können von computergestützten Prozessen gespeichert, verarbeitet, ausgewertet, erweitert, verknüpft, weitergeleitet und transformiert werden. Der Prozess des elektronischen Publizierens muss dementsprechend eine Unterstützung dieser automatischen Bearbeitungsschritte eines Dokuments leisten können.

A1.13 Schnittstellen zu Programmiersprachen:
Für die automatisierte Bearbeitung eines elektronischen Dokuments durch computergestützte Prozesse ist es unabdingbar, dass Schnittstellen für Programmiersprachen existieren, so dass aus Computerprogrammen heraus auf die Dokumentendaten zugegriffen werden kann und diese weiterverarbeitet werden können.

Dies kann auf unterschiedliche Arten realisiert sein. Bei XML ermöglichen z. B. SAX und DOM den Zugriff auf die Dokumentendaten. Ein anderes Beispiel ist XSLT, deren Syntax in XML codiert wird und das ebenfalls Zugriff auf alle Daten eines Dokuments besitzt.

Es ist damit möglich, Programmcode direkt in Dokumente zu integrieren. Das Publikationssystem führt diesen Code dann in der Bearbeitungs-/Transformationsphsae der Publikationskette aus. Diese Idee ist vergleichbar mit dem Konzept von JSP-Seiten, welche die Ausführung von Java Code in HTML-Seiten ermöglichen.

2.5.2 Anforderungen an die Unterstützung der Publikationskette

Im vorangegangenen Abschnitt wurden die Anforderungen an die konkrete Realisierung des Dokumentenmodells beschrieben. Im folgenden Abschnitt werden nun die Anforderungen betrachtet, die hauptsächlich im Fachgebiet des elektronischen Publizierens mit den Publikationssystemen und der Unterstützung der Publikationskette zu tun haben, also mit den Modellen, Vorgehensweisen und Werkzeugen, die in Publikationssystemen zum Einsatz kommen.

A2.1 Verwaltung aller Ressourcen einer Publikationsanwendung:
Es muss eine Möglichkeit geben, die Ressourcen, die von einer bestimmten Publikationsanwendung verwendet werden, in einer geeigneten Art und Weise zu verwalten. Daten in den Zuständigkeitsbereichen von unabhängigen dritten Parteien, die über Verknüpfungen angesprochen werden, gehören zwar nicht mehr in den Verwaltungsbereich der Publikationsanwendung. Jedoch müssen die Verknüpfungen verwaltet und gepflegt werden können. Wie im World Wide Web können sich auch hier die Verknüpfungsziele ändern. Auf diese Änderungen muss entsprechend reagiert werden können und die Publikationsanwendungen müssen daraufhin anpassbar sein.

A2.2 Management verschiedener Ressourcen und Datentypen:
Moderne Dokumente können multimedial sein. Sie bestehen nicht nur aus reinen Text-

elementen sondern auch aus Grafiken, Animationen, Audioelementen, Videoelementen usw. Entsprechend müssen unterschiedliche Datentypen und Ressourcen verwaltet und integriert werden können. Die Dokumenteninhalte müssen in geeigneten Datenbanken gespeichert werden.

A2.3 Dynamisches Generieren der angeforderten Ressourcen:
Dynamische Dokumente erfordern, dass die angeforderten Ressourcen ebenfalls dynamisch zum Zeitpunkt der Anforderung durch den Nutzer generiert werden. Es sind geeignete Modelle und Verfahren erforderlich, die dieses Verhalten unterstützen.

Diese Anforderung lässt sich in weitere Unteraspekte zerlegen:

Datenquellen integrieren und kombinieren:
Ein Dokument kann aus mehreren Teilen bestehen, die aus unterschiedlichen Quellen stammen. Diese Datenquellen müssen integriert werden. Die so erhaltenen Daten müssen zu einem vollständigen Dokument kombiniert werden.

Dynamische Bestandteile ausfüllen:
Die dynamischen Bestandteile eines Dokuments müssen mit Inhalten, z. B. aus einer Datenbank, ausgefüllt werden.

In Präsentationsformate überführen und ausliefern:
Zum Schluss muss das Dokument, das verschiedene Daten enthält, in ein Format überführt werden, das für die Präsentation geeignet ist.

A2.4 Unterstützung der Erstellung der Dokumente:
Es sollte eine gute Unterstützung der Autoren vorhanden sein, damit sichergestellt ist, dass alle Möglichkeiten des Publikationssystems und des Dokumentenmodells genutzt werden können.

A2.5 Mehrbenutzerfähigkeit:
Ein Publikationssystem wird normalerweise mehrere Benutzer haben. Zum einen werden mehrere Nutzer lesend auf die Dokumente zugreifen, zum anderen werden mehrere Dokumente gleichzeitig bearbeitet werden, teilweise auch einzelne Dokumenten von mehreren Nutzern gleichzeitig. Eine Mehrbenutzerfähigkeit stellt sicher, dass es hierbei nicht zu Datenverlust oder unbekannten Systemzuständen kommt.

A2.6 Navigationsunterstützung:
Strukturierte Dokumente bieten die Möglichkeit zur Verknüpfung mit anderen Dokumenten oder Teilen anderer Dokumente. Der Benutzer muss vom System bei der Navigation durch die Verknüpfungen unterstützt werden, so dass er den größtmöglichen Nutzen erzielen kann und nicht im Netz der Verknüpfungen den Überblick verliert.

A2.7 Unterstützung von Wiederverwendung:
Teile von vorhandenen Dokumenten können in anderen Dokumenten wiederverwendet werden. Dazu müssen diese wiederverwendbaren Teile in einer modularen und medienneutralen Form gespeichert werden, so dass sie aus anderen Dokumenten referenziert werden können. Hierfür ist ein geeignetes Verwaltungsmodell zur Speicherung der Dokumente erforderlich.

A2.8 Unterstützung verschiedener Medienkanäle:
Ein elektronisches Dokument kann über unterschiedliche Medienkanäle transportiert werden. Beispiele für Medienkanäle sind HTTP, WAP oder auch CD-ROM.

A2.9 Versionierung und Change-Management:
Dokumente ändern sich. Sie werden an neue Erkenntnisse, Wünsche und Anforderungen angepasst. Wenn ein Teil eines wiederverwendeten Dokuments geändert wird, kann dies eine ganze Reihe von weiteren Änderungen in anderen Dokumenten nach sich ziehen. Die alten Versionen der Dokumente sollten weiter im Publikationssystem verwaltbar sein, da man diese Daten eventuell wieder einsetzen können möchte oder man aus anderen (z. B. rechtlichen) Gründen in der Lage sein will, den Zustand zu einem bestimmten Zeitpunkt rekonstruieren zu können.

A2.10 Wiederherstellung eine bestimmten Standes:
Es sollte außerdem möglich sein, einen bestimmten Stand aus der Vergangenheit der Publikationsanwendung wiederherzustellen. Dies ist nötig für den Fall von Gerichtsverfahren, die sich um einen bestimmten Inhalt der Publikation drehen. Zum anderen kann es allgemein hilfreich sein, wenn ein alter Stand noch einmal betrachtet werden kann.

A2.11 Vergleich unterschiedlicher Versionen:
Ein Publikationssystem sollte es ermöglichen, dass zwei unterschiedliche Versionen eines Dokuments miteinander verglichen werden können. Dadurch kann der Benutzer die Hauptunterschiede leichter erkennen. Dies ist besonders hilfreich im redaktionellen Prozess, wenn Änderungen an einem Dokument freigegeben werden müssen.

A2.12 Rechte-Verwaltung:
Nicht nur im Zusammenhang mit dem Change-Management ist eine wichtige Frage, welche Berechtigungen für welche Daten vergeben werden und wie fein-granular diese Rechteverwaltung sein soll: sollen Berechtigungen für Mengen von Dokumenten, einzelne Dokumente oder sogar Teile von Dokumenten von der Rechte-Verwaltung unterstützt werden.

A2.13 Änderungsprotokolle:
Es sollten Protokolle über die Ausführung aller relevanten Aktionen und speziell von sicherheitskritischen Aktionen geführt werden. Diese Protokolle sollten vom Publikationssystem automatisch erzeugt und vom restlichen Datenbestand getrennt gespeichert werden.

A2.14 Beachtung von Urheberrechten:
Es kann sein, dass Dokumente vom Publikationssystem bereitgehalten werden, deren Ersteller besondere Verwertungsrechte geltend machen. Meistens geht es dabei um die Vergütung entsprechend der Dokumentennutzung[8]. Sie dient nicht dazu, den Zugriff auf Dokumente im Sinne der Rechte-Verwaltung (vgl. Anforderung A2.12) einzuschränken. Die Beachtung der Urheberrechte erfordert zum einen, feststellen zu können, wer ein Dokument wie oft abruft. Zum anderen muss es eine Verbindung zwischen den Dokumenten und den sie betreffenden Regelungen geben.

[8] In Deutschland vertritt die VG Wort (`http://www.vgwort.de`) die Interessen von Rechteinhabern.

A2.15 Unterstützung von Recherche und Information-Retrieval:
Strukturierte Dokumente können andere Dokumente auf verschiedene Arten wiederverwenden. Sie können Teile der Dokumente einbinden oder sie können sich über Verknüpfungen auf andere Dokumente beziehen. Zur Unterstützung dieser Fähigkeiten muss der Autor in der Lage sein, andere Dokumente zu finden, deren Inhalte er wiederverwenden oder auf die er verweisen kann. Daher muss die Recherche und das Information-Retrieval unterstützt werden.

A2.16 Unterstützung von Suchwerkzeugen:
Neben den Autoren haben auch andere Nutzer ein Interesse daran, den Dokumentenbestand nach bestimmten Begriffen oder Kriterien durchsuchen zu können. Die Suchwerkzeuge, die bislang im Elektronischen Publizieren eingesetzt werden, haben jedoch Probleme bei der Erfassung von Dokumenten, wenn sie dynamisch generiert werden oder nur nach Anmeldung des Nutzers zugänglich sind (z. B. im Rahmen eines kostenpflichtigen Angebots).

A2.17 Modellierung und Steuerung des Publikationsprozesses:
Die Publikationskette besteht aus einer Reihe von Stationen, die miteinander verbunden sind. Durch die Dynamik und Interaktivität der Dokumente werden diese Stationen abhängig vom Benutzer unterschiedlich durchlaufen. Diese Abläufe einer Publikationsanwendung müssen auf geeignete Art modelliert und gesteuert werden können. Die Modellierung einer Publikationsanwendung beeinhaltet als Anforderung an das Fachgebiet auch die Unterstützung bei der Anforderungsdefinition einer konkreten Publikationsanwendung.

A2.18 Verarbeitungsfunktionen:
Moderne, strukturierte Dokumente enthalten Daten in einer computerlesbaren Form. Die Semantik der Inhalte, aus denen das Dokument besteht, ist durch Strukturelemente kenntlich gemacht. Daher kann ein Dokument automatisiert von computergestützten Prozessen verarbeitet werden. Diese Eigenschaft ist vom Fachgebiet des elektronischen Publizierens durch passende Modelle, Vorgehensweisen und Werkzeuge zu unterstützen. Ein Beispiel wurde bereits bei den Vorteilen des Modells für strukturierte Dokumente angesprochen: Indexierungswerkzeuge können Wörter abhängig vom Element, in dem sie enthalten sind, unterschiedlich gewichten. So sind Begriffe in der Zusammenfassung eines Dokuments wahrscheinlich wichtiger als die gleichen Begriffe in der Bibliografie. Wie dieser Ansatz konkret umgesetzt werden kann, ist eine Frage des Fachgebiets.

A2.19 Personalisierungsfunktionen:
Die verschiedenen Benutzer in den Zielgruppen haben unterschiedliche Ansprüche und Anforderungen an die Dokumente, den darin enthaltenen Daten, an die Präsentation und die Nutzung der Daten. Personalisierung ist die Anpassung der Dokumente an die individuellen Anforderungen in der Zielgruppe. Eine Voraussetzung hierfür ist, dass der Endnutzer in die Lage versetzt wird, seine individuellen Vorstellungen in den Prozess des elektronischen Publizierens einzubringen. Publikationssysteme müssen geeignete Mechanismen bereitstellen, um die verschiedenen Benutzer zu erkennen und zu verwalten. Dazu gehören Personalisierungsfunktionen, die Unterstützung der Authentifizierung und die

Modellierung von Nutzern. Die Dokumente müssen, basierend auf diesen Daten, an die Nutzungsansprüche in der Zielgruppe angepasst werden können.

Für die Personalisierung von Dokumenten muss ein Publikationssystem folgenden Anforderungen genügen:

- Es muss eine Benutzerverwaltung vorhanden sein, die es ermöglicht, Benutzer von einander zu unterscheiden.
- Es muss einen Rückkanal geben, über welchen Daten vom/über den Benutzer vom Ende der Publikationskette in Richtung des Anfangs der Publikationskette übertragen werden können.
- Es muss Late Binding unterstützt werden, um Präsentationsinformationen erst zu einem späteren Zeitpunkt in der Publikationskette auf das Dokument anwenden zu können.

A2.20 Unterstützung von Mehrsprachigkeit:

Zu den spezifischen Anforderungen von Benutzern gehört auch, dass die Dokumente in einer Sprache angeboten werden, die der Benutzer versteht. Daher sind die sprachlichen Elemente eines Dokuments in den passenden Sprachen vorzuhalten. Dies stellt zwei Anforderungen an das Fachgebiet:

- es muss der Übersetzungsvorgang und die Speicherung der übersetzten Elemente möglich sein.
- die bereits übersetzten Elemente müssen bei Änderungen am Hauptelement – also dem Element, das als Vorlage für die Übersetzung dient – an die Änderungen angepasst und erneut übersetzt werden. Dies muss zum einen erkannt und zum anderen entsprechend unterstützt werden.

A2.21 Flexibilität in Bezug auf Online/Offline Nutzung:

Das elektronische Publizieren muss flexibel möglich sein. Es müssen verschiedene Medien unterstützt werden können und es sollte ein flexibler Einsatz der Dokumente möglich sein, egal ob die Nutzung der Dokumente mit einer aktiven Verbindung zum Internet oder ohne eine solche Verbindung geschieht (Online/Offline).

A2.22 Unterstützung eines redaktionellen Workflow:

Ein Publikationssystem verwaltet die verwendeten Ressourcen einer Publikationsanwendung. Damit muss es also auch die Aufgaben eines Content Management Systems erfüllen. Die verwalteten Inhalte müssen von Autoren erstellt, gespeichert, geprüft und zusammengestellt werden, um von der Publikationsanwendung benutzt werden zu können. Die an diesen Arbeitsschritten beteiligten Parteien müssen in ihren Tätigkeiten entsprechend unterstützt und koordiniert werden. Das Ziel ist dabei, die Qualität der redaktionellen Arbeit zu verbessern. Dies erfolgt durch Aufgabenverteilung und eine Qualitätskontrolle durch gestaffelte Reviewzyklen. Damit geht die Anforderungen an das Berechtigungskonzept einher, ein entsprechendes Rollenmodell zu unterstützen.

2.5.3 Weitere Anforderungen

Neben den spezifischen Anforderungen an Publikationssysteme, die in den beiden vorangegangenen Abschnitten beschrieben wurden, gibt es auch allgemeine Anforderungen, denen sich auch viele andere interaktive Softwaresysteme gegenüber sehen (vgl. [Sch01]):

Plattformunabhängigkeit:
Eine Softwarelösung, die das elektronische Publizieren ermöglicht und unterstützt, sollte idealerweise auf offenen Standard-Technologien aufbauen, um unabhängig von der Plattform zu sein und um auf eine große Auswahl an Softwarekomponenten und Dokumenten im verwendeten Dokumentenmodell zurückgreifen zu können.

Gute Benutzerschnittstelle:
Es existieren verschiedene Richtlinien und Normen, wie Benutzerschnittstellen gut – d. h. effizient und ergonomisch – gestaltet werden können (z. B. ISO-Norm-9241-10).

Datenschutz und Datensicherheit:
Geeignete Mechanismen der durchgängigen Zugriffskontrolle stellen zum einen einen konfliktfreien Mehrbenutzerbetrieb sicher. Zum anderen gewährleisten sie Datenschutz und Datensicherheit. Gerade in Mehrbenutzersystemen, in denen benutzerspezifische Daten gespeichert sind, ist es wichtig, dass keine Unbefugten Zugriff auf diese Daten erhalten. Es ist Aufgabe der Datensicherung durchzusetzen, dass Daten nicht unberechtigt eingesehen oder vernichtet werden.

Skalierbarkeit:
für die Anpassung an verschiedenste Performanzanforderungen, z. B. durch eine verteilbare Mehr-Schichten-Architektur. Dadurch ist es möglich, flexibel auf steigende Nutzerzahlen zu reagieren.

Anpassbarkeit:
an das konkrete Einsatzgebiet des Werkzeugs. Gerade wenn ein Softwaresystem ein allgemeines Problem löst, muss es für den konkreten Einsatz häufig noch an unterschiedliche Randbedingungen angepasst werden. Dieser Punkt ist verwandt mit dem Punkt „Integration“.

Integration:
in die vorhandene Werkzeugumgebung. Ein Softwaresystem muss häufig in ein bestehendes Umfeld aus bereits eingesetzten Softwaresystemen integriert werden. Dies ist speziell wichtig bei größeren Unternehmen, die in der Regel eine Software zur Planung und Steuerung der Abläufe einsetzen. Beispielsweise könnte die Benutzerverwaltung auf einen bereits im Unternehmen eingesetzten Verzeichnisdienst (z. B. über LDAP) zugreifen (vgl. [Byr04]).

Erweiterbarkeit:
Dieser Punkt ist verwandt mit der Anpassbarkeit. Die Erweiterbarkeit geht über die Anpassbarkeit hinaus. Die Möglichkeit der Erweiterbarkeit legt fest, wie gut ein Publikationssystem um zusätzliche Funktionalität erweitert werden kann, wenn die Anpassung der vorhandenen Funktionalität nicht ausreicht. Die Erweiterung kann beispielsweise in Form von Komponenten ermöglicht werden.

2.6 Ein Format für strukturierte Dokumente

Bislang wurde nur das theoretische Modell der strukturierten Dokumente beschrieben. Wie bereits in 2.3.2 erwähnt wurde, ist jedoch für den Einsatz des Modells auch eine konkrete syntaktische Realisierung erforderlich. Im folgenden Abschnitt wird daher XML als das wichtigste Beispiel für eine solche Realisierung vorgestellt.

2.6.1 XML

Die Extensible Markup Language (XML) definiert eine bestimmte syntaktische Realisierung für die Unterteilung von Dokumenten in Strukturelemente und die eigentlichen Inhalte. Die Strukturelemente werden in XML auch „Markup“ genannt und sind im gleichen Zeichenstrom enthalten wie die eigentlichen Inhalte des Dokuments. Mit einer speziellen Kodierung ist es möglich, zwischen Markup und Inhalten zu unterscheiden. Ein einzelnes Strukturelement besitzt jeweils Kodierungen für den Anfang und das Ende des Elements. Diese Kodierungen heißen auch „Tags“. Alles, was zwischen den beiden Tags steht (sowohl Inhalte als auch weitere Strukturelemente) wird logisch diesem Strukturelement untergeordnet. Auf diese Weise entsteht eine Baumstruktur von Strukturelementen. Im Starttag können für das Strukturelement noch weitere Daten, sogenannte Attribute, angegeben werden.

XML ist eine sogenannte „Metasprache für Markupsprachen“. Dies bedeutet, dass sie eine bestimmte Syntax für die Anreicherung von Inhalten mit Strukturelementen definiert ohne gleichzeitig einen konkreten Vorrat an Strukturelementnamen festzulegen. Dieser Sprachschatz von Strukturelementen kann über eine sogenannte „Document Type Definition“ (DTD) festgelegt werden. Die in der XML-Syntax geschriebenen Dokumente können dann programmgesteuert auf ihre Konformität mit der DTD überprüft werden.

XML basiert auf der Standard Generalized Markup Language (SGML [Con95, SMB94]). SGML wurde bereits 1986 als ISO Standard definiert[9]. Das bekannteste Anwendungsbeispiel von SGML ist die Webseitensprache HTML. SGML ist syntaktisch sehr flexibel und somit sehr mächtig, was dazu führt, dass Anwendungen, die SGML interpretieren, entsprechend komplex und aufwändig zu programmieren sind (vgl. [Kay03a]). In der Folge wurde nur wenig Software verfügbar, die den Einsatz von SGML unterstützt hat, und die verfügbare Software war sehr teuer. Daher wurde als verwandte, aber vereinfachte Version XML entwickelt, welche 1998 vom W3 Konsortium in der Version 1.0 verabschiedet wurde.

Einige syntaktische Konstruktionen, die bei SGML erlaubt sind, sind bei XML nicht möglich. Seit der Veröffentlichung des Standards haben sich bereits eine ganze Reihe von Satellitenstandards entwickelt.

Diese Satellitenstandards sind aus den Anforderungen entstanden, die an XML bestanden und auch weiter bestehen. XML selbst liefert Sprachmittel zur Behandlung folgender Punkte der Anforderungsliste an den Prozess des elektronischen Publizierens:

[9] `http://www.iso.org/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=16387`

- Dynamisches Zusammenstellen der Inhalte und multimediale Komponenten.
- Mit der Hilfe von DTDs und XML Schema kann ein gemeinsames Vokabular definiert werden.
- Die XML Query Language unterstützt verschiedene Datenquellen, dynamische Dokumente und die Personalisierung von Dokumenten.
- XML Forms unterstützen interaktive Dokumente.
- XPointer und XLink unterstützen die Adressierbarkeit und Verknüpfbarkeit von Dokumenten.
- XSL, das aus den drei Unterstandards XSLT, XSL-FO und XPath besteht, unterstützt die Adressierbarkeit von Dokumententeilen, die Ausgabe des Dokuments in unterschiedlichen Formaten, dynamische Dokumente, externe Präsentationsanweisungen, Personalisierung von Dokumenten und das Late Binding von Stilvorlagen (vgl. [Kay03b]).
- Durch die Schnittstellen SAX und DOM kann aus anderen Programmiersprachen heraus auf Dokumente zugegriffen werden, um diese automatisiert zu bearbeiten.

2.6.2 Vorteile von XML

XML ist sehr gut für den Einsatz im elektronischen Publizieren geeignet. Eine ganze Reihe von Merkmalen tragen dazu bei:

XML ist ein vollständig offener Standard:
XML basiert auf einer festen Syntax aber unbegrenzt erweiterbarer Semantik. Dies bedeutet, dass jeder, der XML einsetzt, gewisse Grundregeln befolgen muss. Dadurch ist die Austauschbarkeit sichergestellt. Man muss sich jedoch nicht „verbiegen“, um sich an ein bestehendes Datenmodell anzupassen. Verschiedene Gruppen von Dokumenten besitzen normalerweise auch eine einzigartige Struktur – oder Semantik. XML besitzt Möglichkeiten zur Strukturdefinition von Dokumenten, die diesem Umstand Rechnung trägt.

XML ist ein selbstbeschreibendes Format:
Die Syntax ist festgelegt, aber der Rest kann beliebig definiert werden. D. h. es müssen nur der Standard und die Werkzeuge unterstützt werden und es können beliebige Daten gespeichert, verwaltet, bearbeitet und bereitgestellt werden. Dies ist ideal für Archive, die lange Zeit lesbar sein müssen oder für Archive, die aus unterschiedlichen Datenformaten gespeist werden.

XML ermöglicht universellen Datenaustausch:
Verschiedene Publikationssysteme und Unternehmen können Inhalte mit Hilfe von XML austauschen. Dabei müssen sie weder interne Datenmodelle offenlegen noch in komplexe Integrationsprozesse investieren. Daher eignet sich XML besonders für „Daten in Bewegung“. Unternehmen versuchen, einen höheren Wert aus ihren Inhalten zu erzielen, genau dadurch, dass sie die Daten in Bewegung setzen [Byr04]. Damit ist der Fall gemeint, dass Dokumente in unterschiedlichen Formaten publiziert werden.

Kontrolle und Anpassbarkeit durch Erweiterbarkeit:
Elektronisches Publizieren verfolgt zwei ganz wesentliche Ziele: Zum einen wird die Präsentation von den eigentlichen Inhalten getrennt (d. h. gleiche Inhalte können mit verschiedenen Präsentationen verbunden werden). Zum anderen sollen die Inhalte unabhängig sein von dem Ort, an dem sie genutzt werden (z. B. in der Site Map). Wenn diese beiden Ziele erfüllt sind, ist es möglich, Inhalte mehrfach zu verwenden und auch sie gleichzeitig in unterschiedlichen Dokumenten, Formaten und Präsentationen verfügbar zu machen. Durch die Erweiterbarkeit, die einen ganz wesentlichen Aspekt von XML (Extensible Markup Language) darstellt, wird genau das erreicht: Mit XML können die Inhalte beschrieben werden, ohne festzulegen, wie sie präsentiert werden müssen oder wo sie abgespeichert sind.

Ermöglichung besserer Suchergebnisse:
Neben dem generellen Vorteil von strukturierten Dokumenten, dass Suchwerkzeuge die Strukturelemente für die Verbesserung der Suchergebnisse verwenden können, ermöglicht XML es auch, die einzelnen Strukturelemente flexibel zu benennen. Dies kann Suchwerkzeugen, die damit umgehen können, dazu dienen, die Ergebnisse weiter zu verbessern.

Quasi-Standard für den Datenaustausch:
XML hat sich zum Quasi-Standard für den Austausch und die Aggregierung von Inhalten entwickelt [Byr04]. Sie erleichtert das Zusammenstellen von Inhalten aus unterschiedlichen Quellen zu einem Dokument deutlich. Inhalte mit unterschiedlichen Strukturen können automatisiert und regelbasiert in eine gemeinsame Struktur überführt werden.

2.6.3 Studie zur XML-Nutzung in Verlagen

XML wird nachgesagt, das Potential zu besitzen, um einige wichtige Problemstellungen des elektronischen Publizierens zu erleichtern. Dazu zählen z. B. das Lösen von Schwierigkeiten bei der Informationsverarbeitung, das Erreichen von Prozessvereinfachungen, sowie die Senkung von Kosten.

Um zu überprüfen, inwieweit diese erhofften Vorteile von XML auch in der Realität zutreffen, wurde vom Arbeitskreis Elektronisches Publizieren des Börsenvereins des Deutschen Buchhandels im Jahr 2004 eine empirische Untersuchung durchgeführt ([Arb04b, Hes04]). Die Studie sollte die Anwendungsgebiete, die Wirtschaftlichkeit und die Erfolgsfaktoren von XML beleuchten. In dieser Untersuchung wurden die Mitgliedsverlage in drei Phasen von Februar bis August 2004 zur Nutzung von XML befragt. Die wichtigsten Ergebnisse sind im folgenden zusammengefasst.

In der ersten Phase der Studie war das Ziel, die Verbreitung der XML-Technologie innerhalb der Verlage zu ermitteln. Ein Ergebnis der Befragung war, dass 29 % der Verlage bereits XML einsetzen. Dabei war der Anteil der Verlage, die XML einsetzen, umso höher, je größer die Verlage waren. Auch wenn sich mit knapp einem Drittel noch nicht viele Verlage für den Schritt zu XML entschlossen haben, können bereits ein paar Rückschlüsse gezogen werden: XML ist vor allem in den neueren Medienbereichen von DVD, CD-ROM und Internet relevant, weniger im

klassischen Buch- und Zeitschriftenbereich. Wirklich bedeutend wird XML jedoch dann, wenn viele Medienkanäle bedient werden sollen (vgl. auch [Kay03a]). Hier werden dann die Vorteile eines strukturierten Dokumentenformats und der weiteren Möglichkeiten, die XML bietet, von den dort tätigen Verlagen als so bedeutend gesehen, dass die höhere Komplexität gegenüber einfacheren Formaten keinen Nachteil mehr darstellt. Man kann soweit gehen, dass ab einer bestimmten Komplexität des Publikationsprozesses das daran angepasste Dokumentenformat zwangsläufig ebenfalls eine nicht-triviale Komplexität besitzt und sich die Prozesse mit den einfacheren Formaten nicht mehr zufriedenstellend handhaben ließen. Bei den Verlagen, die XML bislang nicht einsetzen, geben 43 % Prozent als Grund dafür an, dass sie nicht über ausreichendes Know-How verfügen.

In der zweiten Phase der Studie sollten die bevorzugten Anwendungsgebiete, die Wirtschaftlichkeit und die Erfolgsfaktoren von XML in den Verlagen untersucht werden. Dort ist der am häufigsten genutzte Einsatzzweck von XML-Technologien die Formatintegration und -transformation. Daneben sind die Möglichkeiten zur medienneutralen Datenhaltung in Datenbanken und der inner- und zwischenbetriebliche Datenaustausch hervorzuheben.

Der Nutzen aus XML wird insgesamt als höher eingeschätzt als die mit XML einhergehenden Kosten. Anfangskosten werden hauptsächlich für die Strukturierung von Inhalten, aber auch für Mitarbeiterschulungen und Prozessänderungen erwartet. Durch den Einsatz von XML nehmen hauptsächlich die Konvertierungskosten für Formatumwandlungen leicht ab. Alle anderen erhobenen einmaligen sowie laufenden Kostenarten nehmen zumindest kurzfristig leicht zu (z. B. Erfassungkosten, Strukturdefinitionskosten). Dabei sind die verlagsspezifischen Herstellungskosten umso geringer, je strukturierter die Inhalte sind, die der Verlag verarbeitet und publiziert.

Der bedeutendste Erfolgsfaktor für den Einsatz von XML-Technologien in Verlagen ist somit die Unterstützung von strukturierten Daten. Daneben spielt auch die Möglichkeit, Abläufe und Bearbeitungsschritte automatisieren zu können, eine wichtige Rolle. Die Unterstützung eines standardisierten Formats ist ebenfalls nicht unbedeutend. Die Plattformunabhängigkeit von XML ist ebenfalls ein wichtiger Faktor. Dieses Ergebnis der zweiten Phase bestätigt XML als eine Implementierung des Modells für strukturierte Dokumente.

In der dritten Phase der Studie wurden einige Praxisbeispiele für den Einsatz von XML in Verlagen anhand von konkreten Fallstudien untersucht. Laut der Studie lässt sich zusammenfassend sagen, dass XML sein volles Potential dann entfaltet, wenn Inhalte, Stilvorlagen zur Präsentation und Strukturelemente physisch zentral, logisch jedoch getrennt voneinander gespeichert werden. Zudem erhöht eine systematische Auszeichnung mit Metadaten die Möglichkeit der effizienten Wiederverwendung von Inhalten.

Es müssen dabei jedoch hohe Anfangsinvestitionen für die Strukturierung von Inhalten, aber auch für Mitarbeiterschulungen und Prozessänderungen in Kauf genommen werden. Dennoch wird XML insgesamt häufig von den Verlagen als wirtschaftlich sinnvoll eingeschätzt. Die reine Kostenbetrachtung reicht unter Umständen nicht aus, denn der Einsatz von XML-Technologien kann Geschäftsprozesse effizienter und flexibler gestalten. Die Etablierung von XML als Datenaustauschformat kann neue Geschäftsfelder eröffnen.

2.7 Weitere Forschungsbereiche

In diesem Kapitel wurde bisher das elektronische Publizieren sowohl als Prozess als auch als gesamtes Fachgebiet näher beleuchtet. Ausgehend vom Begriff des Dokuments wurden die strukturierten Dokumente mit ihren Eigenschaften und Vorteilen vorgestellt. Anschließend wurden daraus die Anforderungen sowohl an den Prozess des elektronischen Publizierens als auch an das Fachgebiet abgeleitet und beschrieben. In diesem Teil des Kapitels soll nun kurz auf ausgewählte weitere Forschungsbereiche des Gebiets des elektronischen Publizierens eingegangen werden.

Diese Forschungsbereiche des elektronischen Publizierens stehen teilweise in Zusammenhang mit dem Modell für strukturierte Dokumente und haben auch teilweise Bezug zur Liste der Anforderungen, die weiter oben vorgestellt wurden.

2.7.1 Ontologien

Ein wichtiges Prinzip strukturierter Dokumente ist die Trennung von Präsentation auf der einen Seite und Inhalt und Struktur auf der anderen Seite. D. h. der eigentliche Inhalt und die Strukturelemente stehen in einer gemeinsamen Datei und der Autor des Inhalts muss den Text, den er erstellt, mit dem passenden Markup anreichern.

Ein Problem dabei ist die Frage, wie die entsprechenden Tags für die Strukturauszeichnung festgelegt werden. Es ist in vielen Fällen sinnvoll, ein gemeinsames Vokabular festzulegen. Eine Ontologie ist eine Hierarchie von Begriffen aus einem Vokabular, beschrieben in einer speziellen Ontologien-Beschreibungssprache [Pid03].

Diese Sprache besitzt eine Grammatik, mit der festgelegt werden kann, wie Ausdrücke des Vokabulars verwendet werden, um etwas Aussagekräftiges auszudrücken. Die Grammatik umfasst formale Regeln. z. B. legt sie fest, wie wohlgeformte Ausdrücke aussehen müssen und wie Begriffe des Vokabulars miteinander kombiniert werden können.

Es gibt einen eigenen Forschungsbereich, der sich mit Fragen der Ontologien beschäftigt. Es sollen dort u. a. standardisierte Vokabulare für bestimmte Bereiche festgelegt werden, die sich in Form von DTDs als Strukturvorgaben verwenden lassen. Bei Dokumenten, die vergleichbare Strukturen besitzen, aber nicht dasselbe Vokabular verwenden, können die Vokabulare mit Hilfe von Mappings angeglichen werden.

Eine offene Frage bei der Festlegung der Vokabulare ist z. B. wie tief das Gebiet modelliert werden soll und wie weit diese Struktur eine absolute Gültigkeit besitzt.

2.7.2 Mediendurchdringung

Das Internet breitet sich auf immer mehr Geräte aus. Waren anfangs nur wenige Rechner mit dem Internet verbunden, ist dies heute schon die Regel. Kaum ein Unternehmen kommt noch ohne

Internetzugang aus. Darüber hinaus hat sich das Internet auch auf andere Geräte ausgedehnt, die nicht von Anfang an dafür gedacht waren. So ist das mobile Internet, das den Zugang über mobile Endgeräte ermöglicht, eine relativ neue Entwicklung. Inzwischen können auch PDAs (Personal Digital Assistants) benutzt werden, um Daten aus dem Internet anzuzeigen und in das Internet zu übertragen.

Diese neuen Entwicklungen bieten neue Chancen. Sie stellen jedoch auch neue Herausforderungen an das elektronische Publizieren. So ist beispielsweise aufgrund der technischen Besonderheiten wie eine schmalbandige Datenverbindung und ein kleines Display eine Konvertierung der zu publizierenden Daten in entsprechende Formate erforderlich. Zu den neuen Chancen gehört etwa die Nutzung von Positionsinformationen, die von mobilen Endgeräten zur Verfügung gestellt werden, um die Daten abhängig vom dem Ort, an dem sich der Endnutzer befindet, anzupassen.

2.7.3 Personalisierung

Neben der reinen Anpassung an die Bedürfnisse des Benutzers kommt der Personalisierung noch eine weitere, wichtige Funktion zu: der Reduzierung des sogenannten „Information Overload“ des Lesers. Es gibt bereits eine große Menge an Daten. Gleichzeitig kommen ständig neue Daten hinzu, die auch in zunehmendem Maße miteinander verknüpft sind, so dass sich die Komplexität, die Informationen zu erfassen, immer weiter erhöht. Der Leser steht vor der Aufgabe, die für ihn relevanten Daten herauszufiltern, gleichzeitig muss er auch mit sich widersprechenden Informationen umgehen. Auf der Seite der Autoren besteht das Problem, die Daten möglichst schnell zu veröffentlichen und darüber hinaus die Daten auch genau der interessierten Zielgruppe zugänglich zu machen.

Zum Problembereich der Personalisierung gehören eine Reihe von Gebieten:

- Identity Management.
- individuelle Benutzerunterstützung während der Suche nach Dokumenten.
- Kurzzeit- und Langzeitpersonalisierungstechniken.
- Informationsfilterung in Publikationssystemen, insbesondere Collaborative Filtering.

Durch Personalisierung kann der Nutzen, die Benutzerzufriedenheit und die Benutzertreue verbessert werden, indem der Leser mit auf seine speziellen Bedürfnisse maßgeschneiderten Dienste versorgt wird. Gleichzeitig wird damit die Qualität des Informationstransfers vom Autor zum Nutzer verbessert (vgl. [Miz02]).

2.7.4 Modellierung und Realisierung des Publikationsprozesses

Ein wesentlicher Bereich, der momentan noch nicht befriedigend bearbeitet ist, betrifft die Modellierung und die Realisierung des Publikationsprozesses. Dieser Punkt umfasst alle Aspekte, die mit der Modellierung zu tun haben.

Das Modell der strukturierten Dokumente beschreibt zunächst einmal die Dokumente als statisch-passiv. In der momentanen Praxis können Dokumente aber durchaus auch dynamisch-interaktiv sein, z. B. in Form von eingebettetem Programmcode (vgl. Abbildung 5.4 auf Seite 142). Hier ist die Realität der Modellierung also voraus.

Jede einzelne Station in der Publikationskette muss in einem geeigneten Modell erfasst werden können: von der Integration verschiedener Datenquellen wie Datenbanken, Programme, anderer Dokumente, über die Wiederverwendung auch von Teilen von Dokumenten mit den Aspekten der Rechteverwaltung, Versionierung und dem Change-Management, bis hin zu automatisierten Bearbeitungsschritten.

2.8 Fazit

Im letzten Unterkapitel wurden weitere Forschungsgebiete im elektronischen Publizieren vorgestellt. Auch – oder gerade – nach der Entwicklung des Modells der strukturierten Dokumente ist die Entwicklung nicht zum Stillstand gekommen. Es gibt eine Reihe von Gebieten, in denen es noch offene Fragen gibt, die in der Zukunft in Forschungsarbeiten beantwortet werden können. Diese Arbeit fokussiert dabei auf die Betrachtung des Bereichs des Software Engineering und der Unterstützung einer geeigneten Modellierung der Abläufe in der Publikationskette.

3 Publikationsanwendungen

In diesem Kapitel werden Szenarien untersucht, in denen Publikationssysteme eingesetzt werden können. Dazu werden zunächst ähnliche Anforderungen aus Abschnitt 2.5 zu größeren Einheiten zusammengefasst. Anschließend werden die Szenarien beschrieben und die Signifikanz der Funktionscluster im jeweiligen Szenario untersucht. Anhand dieser Ergebnisse werden zwei Gruppen von Funktionsclustern mit allgemein hoher bzw. niedriger Bedeutung für die Szenarien identifiziert. Als weitere Folgerung werden Klassen von Szenarien mit vergleichbaren Ergebnissen bei den Signifikanzen der Funktionscluster gebildet. Abschließend werden Weiterentwicklungsmöglichkeiten aufgezeigt.

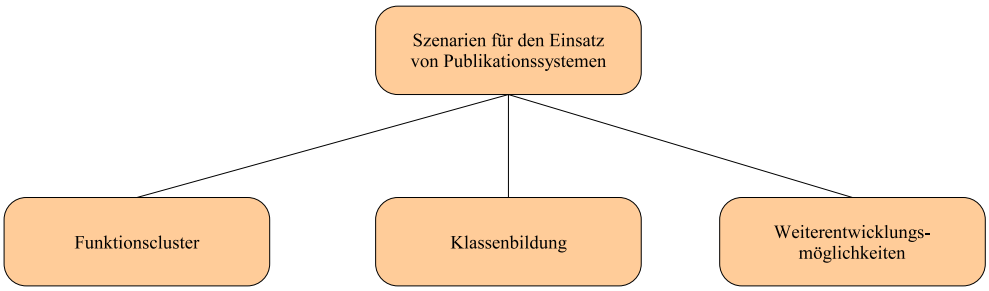

Abbildung 3.1: Kapitelübersicht

3.1 Funktionscluster

In Abschnitt 2.5 wurden Anforderungen an die Funktionalität von Publikationssystemen abgeleitet. In diesem Abschnitt werden Anforderungen, die einander ähnlich sind, zu größeren Einheiten zusammengefasst. Diese Einheiten werden als Funktionscluster bezeichnet.

Datenintegration:
: die Daten, aus denen die Dokumente zusammengestellt werden, können aus verschiedenen Quellen stammen. Dieser Punkt fasst die Anforderungen A1.1, A1.2, A1.3 und A2.7 zusammen.

Navigation:
: umfasst die Hypermedia-Aspekte von Dokumenten und Publikationssystemen, d. h. die Verknüpfbarkeit und Navigationsunterstützung. Grundsätzlich ist es das Hauptziel einer Navigationsunterstützung, dem Benutzer jederzeit den Kontext zu bereitzustellen, wo er

sich gerade befindet und wohin er von dort aus gelangen kann. So soll verhindert werden, dass er die Orientierung verliert. (A1.4, A2.6)

Multimedia-Bestandteile:
die Fähigkeit von Dokumenten, neben Text auch multimediale Bestandteile wie Audio und Video zu enthalten. Nachrichtenportale können z. B. Bewegtbilder oder Audiokommentare anbieten. (A1.5)

Interaktivität:
beschreibt, ob Dokumente und Publikationssysteme eher passiv genutzt werden (z. B. wie TV ohne Rückkanal), oder ob Nutzer interagieren können bzw. diese Möglichkeit sogar einen wesentlichen Aspekt des Szenarios darstellt. (A1.7)

Validierung:
die Möglichkeit zur Prüfung der Dokumentenstruktur auf die Einhaltung von Strukturierungsvorgaben. (A1.8)

Trennung Präsentation:
die Präsentationsanweisungen sind vom Dokument getrennt und insbesondere austauschbar. (A1.9, A1.10)

Formatumwandlung + Medienkanäle:
bezeichnet die Möglichkeit, Dokumente in verschiedenen Formaten und über verschiedene Kanäle bereitzustellen. (A1.11, A2.8, A2.21)

Automatische Bearbeitung:
bezeichnet die Möglichkeit, programmgestützt und automatisch eine Verarbeitung (z. B. Transformation) der Dokumente durchführen zu können. (A1.12, A1.13, A2.18)

Dynamik:
Dokumente werden erst zum Zeitpunkt der Anforderung zusammengestellt. Der wesentliche Aspekt hierbei ist die Fähigkeit, das Dokument abhängig von Parametern unterschiedlich zu generieren. Dieser Cluster bezeichnet die Dynamik, die unabhängig von einer eventuellen Personalisierung (s. u.) ist. (A1.6, A2.3)

Autorenunterstützung:
In der Publikationskette können verschiedene Datenquellen zum Einsatz kommen (vgl. 2.3.4): die Daten der Datenquelle können sowohl aus automatischen Prozessen als auch von menschlichen Autoren stammen. Der Funktionscluster der Autorenunterstützung beschreibt die Integration einer Unterstützung menschlicher Autoren bei der Erstellung dieser Daten in das Publikationssystem. Falls in einem Szenario menschliche Autoren keine Rolle spielen, ist folglich der Funktionscluster „Autorenunterstützung“ in einem solchen Szenario unwichtig. (A2.4)

Mehrbenutzer:
beschreibt den Fall, dass mehrere Benutzer gleichzeitig Daten bearbeiten. (A2.5)

Versionierung:
umfasst die Fähigkeit, jederzeit auf einen beliebigen Datenstand in der Vergangenheit zugreifen zu können. (A2.9, A2.10, A2.11)

Rechteverwaltung:
die Fähigkeit des Publikationssystems, Berechtigungen zu verwalten und durchzusetzen, die den Zugriff von Benutzern auf Dokumente regeln. (A2.12, A2.13)

Suchfunktion:
die Fähigkeit, in einem Publikationssystem nach bestimmten Daten und Dokumenten zu suchen. (A2.15, A2.16)

Personalisierung:
umfasst alle Möglichkeiten, Dokumente angepasst an die individuellen Bedürfnisse von Benutzern bereitzustellen. Personalisierung lässt sich in zwei Bereiche unterscheiden: Personalization und Customization. Der Unterschied liegt in der Quelle, woher die Daten für die Anpassung an die Benutzerbedürfnisse stammen. Bei der Personalization werden die Daten automatisch erzeugt (z. B. durch Beobachtung des Nutzungsverhaltens). Bei der Customization werden diese Daten direkt vom Benutzer erfragt. (A2.19)

Mehrsprachigkeit:
beschreibt die Fähigkeit, Dokumente in verschiedenen Sprachversionen bereitzustellen. (A2.20)

Workflow:
beschreibt die Möglichkeit, Abläufe (=Folgen von Arbeitschritten) bei der Erstellung der Dokumente festzulegen und die Einhaltung sicherzustellen. (A2.22)

3.2 Szenarien für den Einsatz von Publikationssystemen

In diesem Abschnitt werden Szenarien untersucht, in denen Publikationssysteme zum Einsatz kommen können. Diese Aufstellung deckt zwar schon ein möglichst breites Spektrum ab, sie erhebt aber dennoch nicht den Anspruch der Vollständigkeit. Weitere Szenarien, die in Zukunft noch aufgefunden werden, lassen sich jedoch nachträglich leicht mit Hilfe des verwendeten Vorgehens in die Aufstellung integrieren. In der Aufstellung wird jedes Szenario eingehend beschrieben und seine wesentlichen Merkmale werden gezeigt. Bei dieser Untersuchung ist vor allem interessant, zu klären, welche Funktionalität ein Publikationssystem erbringen muss, damit es für ein bestimmtes Szenario eingesetzt werden kann. Hierfür wird für jedes Szenario betrachtet, welche Signifikanzen die Funktionscluster besitzen.

Die Signifikanz eines Funktionsclusters für ein Szenario wird anhand der folgenden vier Stufen ausgedrückt:

- ++: sehr wichtig
- +: wichtig
- o: neutral bis „nice-to-have“
- –: unwichtig

Die Signifikanzen der einzelnen Funktionscluster in den jeweiligen Szenarien liefern die Basis für die spätere Klassenbildung der Szenarien.

Die Aufstellung der Signifikanzen der Funktionscluster innerhalb eines Szenarios liefert zugleich die Grundlage für eine Evaluierung der Eignung von Softwaresystemen für ein bestimmtes Szenario anhand der sogenannten „Qualitativen Gewichtung und Summierung“ (QGS-Verfahren), die in [BHMH03] genauer beschrieben ist.

Das QGS-Verfahren wird dem häufig eingesetzten Evaluierungsverfahren der „Numerischen Gewichtung und Summierung“ (NGS-Verfahren) vorgezogen. Beim NGS-Verfahren wird für jeden Evaluenden ein numerisches Ergebnis bestimmt, das zum Vergleich mit den anderen Evaluenden dient. Dabei wird jedes Kriterium numerisch gewichtet und mit dem Grad der Erfüllung multipliziert. Die Summe dieser Produkte ergibt das Gesamtergebnis für einen Evaluenden. Das NGS-Verfahren besitzt einige entscheidende Nachteile, die vom QGS-Verfahren umgangen werden. Dazu gehört etwa, dass durch eine große Zahl vergleichsweise unwichtiger Kriterien die Ergebnisse bei wenigen wichtigen Kriterien verwässert werden können. Dieser Effekt kann auch durch die Gewichtung nicht vollständig verhindert werden. Daneben bildet das NGS-Verfahren alle Ergebnisse auf eine einzige lineare Skala ab. Das ist die entscheidende Kritik und das innerhalb des NGS-Verfahrens unlösbare Hauptproblem: Die einzelnen Kriterien liegen eben nicht auf einer linearen Skala.

3.2.1 Blogs

3.2.1.1 Beschreibung

Der Einsatzbereich von Blogs[1] ist sehr breit und nicht scharf abgegrenzt. Blogs sind vergleichbar mit Newslettern oder Kolumnen, jedoch persönlicher - sie selektieren und kommentieren oft einseitig und werden deswegen auch mit Pamphleten des 18. und 19. Jahrhunderts verglichen. Blogs sind demnach keine Alternative zu (Online-)Zeitungen, sondern eine Ergänzung. Im Idealfall reagieren Blogs schneller auf Trends oder bieten weiterführende Informationen bzw. Links zu bestimmten Themen. Die meisten Blogs haben eine Kommentarfunktion, die es den Lesern ermöglicht, einen Eintrag zu kommentieren und so mit dem Autor oder anderen Lesern zu diskutieren.

Anfangs wurden Blogs besonders als eine Art persönliches Logbuch über die während des Surfens im Internet gefundenen Websites geführt. Die Websites wurden verlinkt und kommentiert. Inzwischen kommentieren bzw. berichten Blogs auch über aktuelle Ereignisse, Erfahrungen und persönliche Erlebnisse bis hin zur Beschreibung von Einzelheiten des privaten Lebens. Es gibt Blogs von Mitarbeitern von Firmen wie Google, Apple oder Microsoft, die über neue Entwicklungen berichten. Blogs werden von Firmen wie Mattel zur Kundenwerbung und -bindung eingesetzt. In weiteren Typen von Blogs werden Fachinformationen über ein bestimmtes Gebiet veröffentlicht. Viele Einträge in Blogs bestehen aus Einträgen anderer Blogs oder beziehen sich auf diese, so dass Blogs untereinander stark vernetzt sind. Die Gesamtheit aller Blogs bildet die sogenannte „Blogosphäre“.

[1] Auch als „Weblogs“ bezeichnet. Zur Vermeidung von Verwechslungen mit Zugriffsprotokolldateien von Web-Servern wird aber eher die Bezeichnung„Blogs“ verwendet.

Der Einstieg in die Nutzung ist sehr einfach. Es gibt einige Provider im Internet, die Blogs als Dienst zur Verfügung stellen. Oft ist die Nutzung sogar kostenlos. Normalerweise kann schon unmittelbar nach der einmaligen Anmeldung und dem Auswahl einer grafischen Präsentation („Template“) mit dem Einstellen von Einträgen begonnen werden. Auch dafür ist normalerweise nur sehr wenig Vorwissen erforderlich. Dies ist eine wesentliche Eigenschaft von Publikationssystemen für Blogs: Die Verwendung durch den Artikelautor soll möglichst einfach und mit wenig Einarbeitung möglich sein.

Ein typisches Kennzeichen von Blogs ist eine vergleichsweise einfache und flache Struktur der Website. Alle Einträge im Blog sind umgekehrt chronologisch sortiert, neue Einträge erscheinen oben in der Auflistung. Damit soll ein wesentliches Ziel von Blogs unterstrichen werden, nämlich die hohe Aktualität der Einträge. Dies hängt natürlich sehr davon ab, welchen Zeitaufwand der sogenannte „Blogger“ bereit ist zu investieren. Denn üblicherweise steht nur eine einzelne Person hinter dem Blog, die die Artikel in der Freizeit schreibt. Aktualität heißt dabei nicht nur, dass zeitnah auf einzelne Ereignisse reagiert wird, sondern dass periodisch immer wieder neue Artikel im Blog veröffentlicht werden.

Typisch für Blogs ist auch eine relativ hohe Linkdichte. Dies gilt vor allem für den Blogtyp, bei dem andere Websites kommentiert werden.

üblich ist, dass Besucher des Blog die Möglichkeit haben, zu den veröffentlichten Artikeln Kommentare zu schreiben, die dann dem Artikel zugeordnet sind.

Blogs besitzen eine vergleichsweise starke Verknüpfung untereinander. Dies hat dazu geführt, dass zur Unterstützung dieser Verknüpfung ein Verfahren namens „TrackBack“ entwickelt wurde. Wenn ein Autor in seinem Blog einen Artikel schreibt, der sich auf einen Artikel in einem anderen Blog bezieht, so kann sein Blog-System dem System des Bezugsartikels diese Tatsache mitteilen. Man kann dies als extern geschriebenen und gespeicherten Kommentar auffassen und jeder, der den Bezugsartikel liest, kann eine Liste der Artikel abrufen, die sich auf diesen Artikel beziehen. Diese Benachrichtigung wird auch „Ping“ genannt.

Für TrackBack ist eine permanente URL erforderlich, die den referenzierten Artikel eindeutig identifiziert. Anfangs musste diese URL manuell abgefragt und bei der Erstellung des neuen Artikels angegeben werden. Inzwischen wurden Blog-Systeme derart erweitert, dass mit Hilfe von RDF und den Dublin Core Metadaten diese URL in die Ursprungsseite eingebettet ist und automatisch extrahiert werden kann. Dieses Verfahren wird als „TrackBack Auto Discovery“ bezeichnet.

über das TrackBack-Verfahren werden auch zentrale Blog-Verzeichnisserver über neue Artikel benachrichtigt, so dass diese, im Unterschied zu normalen Suchmaschinen mit dem herkömmlichen Pull-Prinzip der Indizierungsverfahren, schon kurze Zeit später die neuen Artikel auffinden.

Artikel lassen sich oft bestimmten Kategorien zuordnen. Hierbei wird ein Artikel genau einer Kategorie zugeordnet. Als Weiterentwicklung verbreiten sich inzwischen sogenannte „Tags“ als im Vergleich zu Kategorien flexiblere Variante. Einem Artikel können mehrere Tags zugeordnet werden, was bei der Einteilung in Kategorien nicht der Fall ist.

Hoster bieten meist weitere Schnittstellen zur Einstellung von Artikeln, z. B. über XML-RPC, SMS, E-Mail.

Viele Blog-Systeme bieten auch die Möglichkeit, die neuen Einträge über RSS-Reader zu lesen.

Andere Blogs, die Photoblogs (kurz auch Phlog), veröffentlichen hauptsächlich Fotografien oder Handy-Kamerabilder (Moblogs). Schnellere Datenübertragungen und neue Download-Technologien (z. B. BitTorrent) erlauben es auch größere Video-Sequenzen - angekündigt als „Internet-TV“ - zugänglich zu machen. Diese Form eines Blogs wird Video-Blog oder kurz Vlog genannt.

Blogs haben mit Problemen wie Linkspamming in Kommentaren und TrackBack- oder Pingbackspamming (Blogspam) zu kämpfen. Um Linkspamming in Kommentaren zu verhindern, werden sogenannte Captchas (Completely Automated Public Turing test to tell Computers and Humans Apart) eingesetzt. z. B. Grafiken mit verzerrten Ziffern und Buchstaben.

3.2.1.2 Bewertung und Einsatzmöglichkeiten von XML

XML wird bereits für Ping-Nachrichten und das TrackBack Auto Discovery eingesetzt. Letzteres verwendet RDF und die Dublin Core Metadaten. RSS Feeds verwenden ebenfalls ein Datenformat, das in einer XML-Sprache beschrieben ist.

Die Vernetzung der Blogs untereinander und mit Verzeichnisservern ist offensichtlich ein wichtiger Aspekt. Dazu lässt sich XML sehr gut einsetzen, es ist prädestiniert für den Datenaustausch. Auch wenn sich unterschiedliche Softwarehersteller nicht auf einen gemeinsamen Standard für das Format einigen können sollten, so kann die Formatkonvertierung automatisiert mit Hilfe von XSLT-Stylesheets erledigt werden. Die dazu notwendigen Softwarekomponenten wie XML-Parser und XSLT-Prozessoren sind bereits zahlreich vorhanden. Sie müssen nur eingebunden werden.

Ideen für Forschungsbereiche: Verbesserung der Kategorisierung der Blogs. Momentan ist die Recherchemöglichkeit beschränkt auf die Suche nach Schlagwörtern. Vereinzelt gibt es Verzeichnisse, die Blogs zu bestimmten Themenkatalogen zuordnen. Katalog von Themen und Einordnung von Blogs in diese Bereiche. Erkennung von „A-List-Blogs“, also den bedeutendsten Blogs, innerhalb der Verlinkungsstruktur von Blogs.

3.2.2 Wikis

3.2.2.1 Beschreibung

Unter Wikis versteht man eine offene Sammlung von Webseiten, die normalerweise von jedem Besucher der Website online über ein einfaches Formular bearbeitet werden können. Die Inhalte werden durch einen kooperativen und offenen Prozess erstellt. Das bekannteste Beispiel für Wikis ist die Internet-Enzyklopädie Wikipedia[2].

Wikis besitzen einige Gemeinsamkeiten mit Blogs. So gibt es bereits Softwaresysteme, die versuchen, die wesentlichen Eigenschaften von Wikis und Blogs miteinander verschmelzen.

[2] Wikipedia: `http://www.wikipedia.org`

Funktionscluster	Signifikanz
Datenintegration	o
Navigation	+
Multimedia-Bestandteile	+
Interaktivität	++
Validierung	–
Trennung Präsentation	+
Formatumwandlung + Medienkanäle	o
automatische Bearbeitung (RSS)	+
Dynamik	–
Autorenunterstützung	++
Mehrbenutzer	o
Versionierung	o
Rechteverwaltung	o
Suchfunktion	+
Personalisierung	–
Mehrsprachigkeit	o
Workflow	–
Besondere Funktionalität	
Chronologische Darstellung der Einträge	
Archivierung mit kalendarischer Zugriffsmöglichkeit	
Kommentarmöglichkeit zu Einträgen	
Vernetzung mit anderen Blogs	
aktive Benachrichtigung von zentralen Verzeichnisservern	
weitere Schnittstellen zur Erstellung von Einträgen (=Datenintegration)	
automatische RSS-Feed Erstellung (=Automatische Verarbeitung)	

Tabelle 3.1: Funktionalität des Szenarios „Blogs“

Wikis beschäftigen sich ebenfalls im Kern mit der Verwaltung von Inhalten, sind also im weitesten Sinne Content Management Systeme. Wie Blogs haben sie das Ziel, die Erstellung von Inhalten im World Wide Web möglichst einfach zu gestalten (vgl. [LC01]).

Um dieses Ziel zu erreichen, wurde für Wikis eine eigene Syntax eingeführt (die sogenannte „Wiki Syntax“), mit der Inhalte geschrieben werden und die einfacher aufgebaut ist als HTML. Die Wiki Syntax verwendet festgelegte Zeichenkombinationen zur Kennzeichnung besonderer Bereiche des Textes. Diese Zeichenkombinationen sind so gestaltet, dass der Text beim Editieren noch möglichst gut lesbar bleibt. Diese Wiki Syntax unterscheidet sich in jeder Wiki-Software nur minimal.

Beispiele für die Auszeichnung sind:

- ==Überschrift Ebene 1==
- ”kursiver Text”

- "'fetter Text"'
- * Liste mit normalen Aufzählungszeichen
- # nummerierte Liste

Da Wikis offene System sind, in denen jeder beliebige Inhalte einstellen und verändern kann, können unerwünschte Eingriffe ein Problem darstellen. Eine automatische Erkennung, ob ein Bearbeitungsvorgang erwünscht ist oder nicht, ist nicht möglich. Daher wird von den meisten Wiki-Systemen eine oder mehrere der folgende Möglichkeiten implementiert:

- RecentChanges-Seite: zeigt eine chronologische Liste aller zuletzt gemachten Änderungen
- Versionskontrolle: speichert alle Versionen einer Wiki-Seite
- Diff-Funktion: zeigt die Änderungen zwischen zwei Versionen einer Wiki-Seite

Wikis können in folgenden Szenarien eingesetzt werden:

- Technische Dokumentation: Installationsanleitungen, Handbücher, FAQs, Change-Logs, etc.
- Projektmanagement: Projektbeschreibungen, Meetingprotokolle, Gesprächsprotokolle, Zeitpläne, Testergebnisse, etc.
- Sonstige: Link-Sammlung, Notiz-Block, ToDo-Listen, etc.

Der Nachteil von Wikis ist, dass sie keine Struktur besitzen. Alle existierenden Webseiten sind prinzipiell gleichberechtigt. Webseiten werden daher normalerweise nur über eine Volltextsuche mit passenden Stichwörtern gefunden. Bei Online-Enzyklopädien enthalten die Artikel typischerweise relativ viele Verknüpfungen mit anderen Artikeln. Außerdem gibt es zu bestimmten Themengebieten sogenannte Portal-Seiten, die eine Übersicht über das Gebiet geben und als Einstiegsseite mit sehr vielen Verknüpfungen gedacht sind.

Die Artikeln in Wiki-Systemen lassen sich in zwei grundsätzlich verschiedene Kategorien einteilen: Auf der einen Seite sind das alle Artikel, die (kooperativ) von Nutzern erstellt wurden. Auf der anderen Seite sind das alle Artikel, die vom Wiki-System generiert werden. Die Artikel dieser Kategorie werden auch als „Spezialseiten“ bezeichnet. Diese Spezialseiten befinden sich im Namensraum „Special:“. Namensräume sind ein Konzept, um unterschiedliche Bereiche von einander zu trennen und Konflikte auszuschließen. So befinden sich die Hilfeseiten, die Seiten von Benutzern, Portalseiten und Kategorieseiten in verschiedenen Namensräumen. Portalseiten dienen als Einstieg in einen bestimmten Themenbereich. Kategorieseiten fassen Artikel, die zur gleichen Kategorie gehören, zu einer Übersicht zusammen.

Die Unterstützung von Verknüpfungen zwischen zwei Wiki-Systemen ist ebenfalls vorhanden. Dazu muss beim Bearbeiten eines Artikels in einem Wiki vor den Verknüpfungsnamen ein passendes Präfix (z. B. „WikiPedia:“) gesetzt werden.

Das wichtigste Wiki ist wie angesprochen Wikipedia. Die große Bedeutung hatte auch starken Einfluss auf die Entwicklung der Wiki-Systeme. Das anfangs für Wikipedia eingesetzte System erwies sich schon bald als zu beschränkt. Der Kölner Biologie-Student Magnus Manske

entwickelte daraufhin mit Hilfe der Skriptsprache PHP und der Datenbank MySQL eine neue Software, die viele Probleme beseitigte. Diskussionsseiten und Seiten über Wikipedia wurden von Artikelseiten streng getrennt. Das Benutzerinterface wurde in viele Sprachen übersetzt. Unterstützung für das komfortable Hochladen von Bilddateien wurde hinzugefügt. Neue Schutzmechanismen kamen hinzu, Administratoren wurden ernannt, die Seiten löschen und Benutzer wegen Vandalismus verbannen konnten.

Eine wesentliche neue Funktion, die Wikipedia anderen Wikis voraus hat, sind sogenannte Beobachtungslisten: Alle in diese Liste eingetragenen Artikel lassen sich gesondert von den „Recent Changes“ über einen längeren Zeitraum überwachen. Wer also viel Zeit in einen umfangreichen Artikel investiert hat, muss nicht ständig auf die Liste aller Änderungen schielen - ein regelmäßiger Blick auf die Beobachtungsliste genügt, und unerwünschte oder unpassende Änderungen durch Dritte können verbessert, gelöscht oder integriert werden.

Seitens der Administration wurde stets auf eine möglichst transparente Vorgehensweise geachtet. Soll eine Seite beispielsweise gelöscht werden, muss sie - sofern es sich nicht um eine offensichtliche Nonsens-Seite handelt - auf einer Spezialseite namens „Votes for deletion“ (auf der deutschen Wikipedia: „Seiten, die gelöscht werden sollen“) eingetragen werden. Dort bleibt sie eine Weile stehen, und sofern niemand Einspruch erhebt, darf ein Benutzer mit besonderen Rechten sie entfernen. Gibt es dagegen Widerspruch, gilt das Konsens-Prinzip: Entscheidungen müssen wenn irgend möglich einvernehmlich gefällt werden. Kontroverse Inhalte werden deshalb eher verschoben, umbenannt oder nachbearbeitet als gelöscht.

3.2.2.2 Bewertung und Einsatzmöglichkeiten von XML

Die Wiki Syntax zur Erstellung der Inhalte ist zwar bereits recht einfach. Dennoch können Inhalte bei der Bearbeitung im Quelltext noch recht unübersichtlich wirken. Hier würde ein WYSIWYG-Editor eine Verbesserung bedeuten. Damit könnten auch Laien, die gar nichts von der speziellen Wiki Syntax wissen, einfach Inhalte bearbeiten. XML könnte hier sehr gut als Datenaustauschformat zwischen Editor und Speicherungsschicht dienen.

3.2.3 Digitale Zeitungen

3.2.3.1 Beschreibung

Dieses Szenario wird beschrieben anhand des Beispiels der Süddeutschen Zeitung E-Paper Ausgabe[3].

Beim E-Paper handelt es sich um eine mit der gedruckten Ausgaben der Zeitung identischen Version. Der Zugang ist nur kostenpflichtig möglich.

Die E-Paper Ausgabe kann seitenweise durchblättert werden. Dabei wird jede Seite in einer verkleinerten Größe vollständig angezeigt. Außer Überschriften sind keine Texte lesbar. Wenn die

[3] Süddeutsche E-Paper: `http://epaper.sueddeutsche.de/`

Funktionscluster	Signifikanz
Datenintegration	o
Navigation	++
Multimedia-Bestandteile	+
Interaktivität	+
Validierung	o
Trennung Präsentation	++
Formatumwandlung + Medienkanäle	+
automatische Bearbeitung	o
Dynamik	–
Autorenunterstützung	++
Mehrbenutzer	+
Versionierung	++
Rechteverwaltung	o
Suchfunktion	++
Personalisierung	–
Mehrsprachigkeit	o
Workflow	o
Besondere Funktionalität	
Automatische Zielfindung von Links über einfache Namen teilweise über Wiki-Grenzen hinweg.	

Tabelle 3.2: Funktionalität des Szenarios „Wikis“

Maus über einen Artikel bewegt wird, wird dieser hervorgehoben. Außerdem wird eine kleines Informationsfenster angezeigt, das die Überschrift und die Unterüberschrift des Artikels anzeigt.

Beim Klick auf einen Artikel wird dieser in einer größeren Version geladen und angezeigt. Dabei gibt es die Möglichkeit zwischen der „Original-Darstellung“ (als Grafikdatei) und der Textansicht (als HTML) umzuschalten. Im Artikel enthaltene Grafiken werden jedoch in keinem der beiden Fälle mit angezeigt. Dazu muss die Grafik explizit in der Seitenansicht angeklickt werden. Dann wird aber auch nur die Grafik angezeigt. Das Ausdrucken des Artikels wird über die Druckfunktion des Browsers unterstützt. Eine spezielle Druckversion existiert nicht. Jede Seite wird auch komplett als PDF-Datei angeboten, die dann gespeichert und ausgedruckt werden kann.

In der Artikelansicht gibt es die Möglichkeit, linear durch die Artikel der Seite zu navigieren. Dabei ist nicht vorhersehbar, welcher Artikel angezeigt werden wird.

Die E-Paper Ausgabe der Süddeutschen Zeitung verwendet als Software MSH-Web:digiPaper. Diese Software verwendet auf Serverseite Java Servlets. Die Aktivierung von JavaScript im Browser des Client ist zwingend erforderlich.

Die Software ist gedacht zur automatisierten Übernahme von sogenannten „Print-Publikationen“ aus Produktionssystemen zur Realisierung von e-Paper-Ausgaben. Sie verfügt über Schnittstel-

len und Adapter zu Planungs-, Redaktions- und Anzeigen-Systemen. Als Basis für den Import dient eine XML-Schnittstelle.

Der Verwendungszweck ist eindeutig begrenzt auf möglichst originalgetreue elektronische Ausgaben von Zeitungen und Zeitschriften.

Funktionscluster	Signifikanz
Datenintegration	+
Navigation	+
Multimedia-Bestandteile	++
Interaktivität	–
Validierung	–
Trennung Präsentation	o
Formatumwandlung + Medienkanäle	++
automatische Bearbeitung	–
Dynamik	–
Autorenunterstützung	–
Mehrbenutzer	–
Versionierung	–
Rechteverwaltung	++
Suchfunktion	+
Personalisierung	+
Mehrsprachigkeit	o
Workflow	–
Besondere Funktionalität	
Integration mit Redaktionssystemen	

Tabelle 3.3: Funktionalität des Szenarios „Digitale Zeitungen“

Defizite bei: Navigation, Personalisierung

3.2.4 Nachrichtenportale

Digitale Zeitungen aus Abschnitt 3.2.3 orientieren sich in der Darstellung sehr stark an der gedruckten Ausgabe. Im Gegensatz sind Nachrichtenportale sehr viel modularer aufgebaut. Es gibt keine Verknüpfung mehr zwischen einem Artikel und einer Zeitungsseite.

Nachrichtenportale bieten auf der Startseite typischerweise eine Übersicht über die aktuellen Schlagzeilen. Jede Schlagzeile ist mit dem dazu gehörigen Artikel verknüpft. Meist gibt es neben der Schlagzeile einen kurzen Textanriss des Artikels, so dass etwas klarer ist, um was es sich dreht.

Viele Nachrichtenportale – insbesondere größere wie die von Tageszeitungen im Internet – verteilen die Nachrichtenartikel auf verschiedenen Ressorts wie z. B. Politik, Wirtschaft, Kultur und

Sport. Oft werden auf der Startseite zu jedem Ressort die jeweils neuesten Artikel angezeigt. Daneben gibt es so gut wie immer eine von Ressorts unabhängige Übersicht der wichtigsten oder neuesten Artikel. Als Ergänzung gibt es oft noch Verknüpfungen zu ständigen Rubriken, die nichts mit den eigentlichen Nachrichten zu tun haben, aber als Mehrwert angeboten werden. Bei der Süddeutschen Zeitung ist das z. B. die SZ-Cinemathek.

In diesen Bereich fallen auch Aktionen wie Gewinnspiele, die immer wieder durchgeführt werden. Bei vielen Angeboten gibt es daher die Möglichkeit, sich als Benutzer zu registrieren, um an den Aktionen teilnehmen zu können.

Gerade bei den Internetauftritten großer Tageszeitungen ist der Nachrichtenbereich nur ein Teil des Angebots. Andere Hauptbereiche sind oft Immobilien-, Wohnungs-, Stellen- und KFZ-Märkte. Diese Bereiche besitzen eigene Anforderungen an die Funktionalität, die sich nicht mit den Anforderungen des Nachrichtenportals decken.

Die im Nachrichtenbereich verfügbaren Artikel sind in der Regel in unterschiedlichen Formaten verfügbar. Eine Druckversion, die Randelemente wie Werbebanner und Verknüpfungssammlungen weglässt, ist Standard. Oft hat der Benutzer auch noch die Wahl, ob er den Artikel vollständig auf einer Seite oder aufgeteilt auf mehrere Seiten haben möchte. Multimediale Bestandteile wie Grafiken und Bilder zur Illustration der Artikel sind üblich.

Die Recherche im Archiv ist ein wichtiger Bereich. Eine Volltextsuche gehört daher zum Standard. Die Möglichkeiten zur Beschreibung und Eingrenzung der Suche unterscheiden sich je nach System.

3.2.4.1 Verbesserungsmöglichkeiten

Bei der Suche nach Artikeln wäre interessant, eine Übersicht zu bestimmten Themen (etwa Wahlkampf, bestimmte, zusammengehörige regionale und überregionale Ereignisse, etc.) abfragen zu können, in der die wichtigsten Artikel zu einem Thema aufgelistet werden.

Verbesserungsmöglichkeiten gibt es auch im Bereich der Informationsverteilung an den Leser. Beispielsweise automatische Newsfeeds (RSS, Atom) oder eine automatische Benachrichtigung, wenn Artikel mit bestimmten Kriterien oder Schlüsselwörtern veröffentlicht werden. Etwas ähnliches bietet die Suchmaschine Google an. Hier kann man sich registrieren und erhält eine E-Mail, wenn sich die Ergebnisse der gespeicherten Suche ändern.

Bei der Personalisierung ist es denkbar, den Nutzer für ihn uninteressante Ressorts ausblenden zu lassen. Gleichzeitig könnte man die Artikel durch die Leser bewertbar machen, auch in verschiedenen Dimensionen wie Qualität, Tiefe, Relevanz, allgemeine Wichtigkeit, etc. Dies würde eine kooperative Wissensbewertung und Qualitätssicherung darstellen. Die Ergebnisse können in der Suchfunktion und der automatischen Benachrichtigung eingesetzt werden.

3.2.5 Digitale Bibliotheken wissenschaftlicher Fachgesellschaften

Digitale Bibliotheken von wissenschaftlichen Fachgesellschaften (z. B. ACM: `http://portal.acm.org/dl.cfm` oder Springer: `http://www.springerlink.com`) stellen wissen-

Funktionscluster	Signifikanz
Datenintegration	+
Navigation	++
Multimedia-Bestandteile	++
Interaktivität	o
Validierung	–
Trennung Präsentation	++
Formatumwandlung + Medienkanäle	+
automatische Bearbeitung	o
Dynamik	+
Autorenunterstützung	+
Mehrbenutzer	+
Versionierung	–
Rechteverwaltung	+
Suchfunktion	++
Personalisierung	o
Mehrsprachigkeit	o
Workflow	+
Besondere Funktionalität	
Integration mit Redaktionssystemen	

Tabelle 3.4: Funktionalität des Szenarios „Nachrichtenportale“

schaftliche Publikationen zur Verfügung. Die Publikationen sind Fachartikel und stammen aus dem verlagseigenen Fundus von Tagungsbänden, Journalen, Magazinen, Newslettern, Special Interest Groups etc. Die Publikationen sind normalerweise nicht kostenlos für alle zugänglich, sondern man muss sich registrieren und bekommt dann Zugriff auf die Artikel. Hochschulen haben meist einen Rahmenvertrag abgeschlossen, so dass hier alle Angehörigen der Hochschule Zugriff erhalten.

Um eine Publikation zu finden, gibt es meist zwei Möglichkeiten: Zum einen durch eine Suchfunktion über den gesamten Bestand, zum anderen durch das gezielte Heraussuchen einer bestimmten Ausgabe einer bestimmten Sammlung von Publikationen (das sind die oben erwähnten Tagungsbände, Journale etc.). Die gesamte Sammlung ist als Baumstruktur organisiert und der Einstieg wird meist schon auf der Hauptseite der Digitalen Bibliothek angeboten. Das Angebot der ACM bietet auch die Möglichkeit, während des Abstiegs im Strukturbaum eine Suche zu starten, die auf genau diesen Unterbaum begrenzt ist.

Auf der vorletzten Ebene des Strukturbaums wird dann eine Übersichtsseite mit einer Liste aller Artikel einer Publikation angezeigt. Nach der Auswahl eines bestimmten Artikels wird eine Seite mit genaueren Informationen über den Artikel angezeigt. Diese Informationen können Meta-Informationen, eine Kurzfassung, eine Liste von Referenzen (teilweise mit Verknüpfung zu anderen Artikeln oder Büchern), eine Liste von Schlüsselwörtern und eine Klassifizierung in

Kategorien sein. Von dieser Seite aus ist auch die PDF-Version des Artikels ansprechbar, die den vollständigen Artikel enthält.

Um den Benutzer bei der Suche nach für ihn relevanten Artikeln zu unterstützen, wird eine Personalisierung in verschiedenen Formen angeboten. Jeder angemeldete Benutzer kann normalerweise eine Reihe von Suchanfragen abspeichern. Teilweise können die Ergebnisse dieser Anfragen mit Kollegen geteilt oder zum Aufbau einer Bibliographie verwendet werden. Eine zweite Unterstützung des Benutzers kann durch Benachrichtigungen erfolgen, wenn neue Artikel bzw. Artikelsammlungen veröffentlicht werden, die ein Schlüsselwort enthalten bzw. es wird das Inhaltsverzeichnis der Artikelsammlung mitgeschickt.

3.2.5.1 Verbesserungsmöglichkeiten

Die Lieferung der Dokumente durch die Autoren im XML Format statt als PDF würde deutliche Vorteile bringen. Die Bibliothek kann Schemata zur Verfügung stellen, mit denen das Dokument bereits im Vorfeld auf Validität geprüft werden kann. Eine automatische Integration in die Datenbank wäre ebenso leichter möglich, wie der einfachere und schnellere Aufbau von Stichwörterverzeichnissen, Referenzen, Volltextindex etc. Eine einheitliche Umwandlung in PDF ist ebenso möglich, wie die Unterstützung anderer Formate.

Zertifizierte Dokumentationsserver: Veröffentlichungen werden der Wissenschaftsgemeinde unveränderbar, lesbar und langfristig zur Verfügung gestellt [Die04].

Größere Verlage, wie z. B. Springer, halten ihre Dokumente intern schon lange in der seit 1986 existierenden „Standard Generalized Markup Language“ (SGML), die alle Voraussetzungen erfüllt, die für Recherche und Retrieval, Präsentation in verschiedenen Medien und Archivierung von Dokumenten geeignet ist, weil sie auf international abgesprochenen Normen basiert und sicherstellt, dass sie auch noch im nächsten Jahrhundert lesbar bleiben, was für proprietäre Formate nicht gilt, deren „Rückwärts-Kompatibilität“ sich selten über mehr als ein halbes Jahrzehnt erstreckt. [Die04]

Defizite seitens der Dokumentenserver: Die rasche Verbreitung des Internet hat zu einem Wildwuchs an digitalen Veröffentlichungen geführt. Jedes Institut, jeder Wissenschaftler, jede Privatperson kann ohne besondere Mühe und ohne Kontrolle Dokumente „ins Netz stellen“, die dann auch von den großen Suchmaschinen gefunden und angezeigt werden. Oft handelt es sich dabei um Arbeiten, die über die Zeit verändert werden, deren Adressen (URLs) wechseln oder die wieder gelöscht werden. Manchmal ist ihre Autorenschaft unklar, Urheberrechte ungeklärt oder ihre Qualität fragwürdig. Diese Unsicherheit, einschließlich der Durchsetzung des Copyright, hält viele Kollegen davon ab, ihre Arbeiten im Internet zur Verfügung zu stellen. In dieser Situation ist die Qualitätssichtung und -sicherung unabdingbar. Die Deutsche Initiative für Netzwerkinformation (DINI), eine Vereinigung der Hochschulbibliotheken, Hochschulrechenzentren und der Medienzentren in Kooperation mit wissenschaftlichen Fachgesellschaften, hat das Konzept eines „zertifizierten Dokumentenservers“ erarbeitet und vergibt entsprechende Zertifikate an Hochschulserver, die diese Bedingungen erfüllen. Es geht dabei um abgeschlossene wissenschaftliche Arbeiten, deren Qualität von Fachexperten begutachtet wurde und die eine

Universität auf Dauer im Internet verfügbar hält. Zu den Anforderungen eines solchen Zertifikats gehören unter anderem:

- Autorenbetreuung: nach außen sichtbares Beratungsangebot, Unterstützung des gesamten Publikationsprozesses (einschließlich rechtlicher und technischer Problemstellungen)
- rechtliche Aspekte: nicht ausschließliches Recht der Universität zur elektronischen Speicherung und zum Verfügbarmachen für die Öffentlichkeit, Weitergabe an Archivierungsinstitutionen wie Die Deutsche Bibliothek (DDB), Definition von Zugriffsrechten für verschiedene Nutzer
- Sicherung von Authentizität und Integrität: Backupsystem, Sicherung des Servers, Datenaufnahme geschieht kontrolliert und dokumentiert; persistent identifier, Verwendung kryptografischer Verfahren
- Sacherschließung: definierte Policy zur Sacherschließung, mindestens verbalen Sacherschließung durch drei Schlagwörter oder klassifikatorische Erschließung
- Metadatenexport: frei zugängliches Angebot der Metadaten, Mindest-Standard: Dublin Core Simple
- Schnittstellen: Webserverschnittstelle für Nutzer, OAI-PMH 2.0
- Zugriffsstatistik: Führung einer konsistenten Zugriffsstatistik nach den rechtlichen Vorgaben
- Langzeitverfügbarkeit: dauerhafte Verbindung der Metadaten mit den Dokumenten, Mindestzeit der Verfügbarkeit 5 Jahre

[Die04]

3.2.6 Persönliche digitale Bibliotheken

Es gibt mittlerweile im Internet eine ganze Reihe von Digitalen Bibliotheken. Literaturrecherchen lassen sich immer öfter ohne den Gang in eine „reale“ Bibliothek durchführen. Dennoch sind die digitalen Angebote oftmals noch verbesserungsfähig. Aufgrund der Vielzahl der Angebote, deren Heterogenität sowie der fehlenden Integration sind Recherchen sehr aufwändig und fehleranfällig[FGK01].

Im Rahmen des von der DFG geförderten Projektes Daffodil wird - aufbauend auf den Ergebnissen der (abgeschlossenen) Projektgruppe PIANO - ein Informatik-Portal entwickelt, das eine effektive Nutzung digitaler Bibliotheken ermöglicht. Die derzeitige Version des Systems integriert zwölf Informatik-relevante digitale Bibliotheken, die dem Benutzer als eine einzige virtuelle digitale Bibliothek erscheinen. Zur strategischen Unterstützung der Suche bietet das System höhere Suchfunktionen an (Strategien, Strategeme, Taktiken), die den Funktionsumfang der einzelnen digitalen Bibliotheken erweitern. Zum Beispiel werden Zitationssuchen, Frageerweiterung/-einengung und das Browsen in Klassifikationsschemata und Inhaltsverzeichnissen unterstützt sowie automatisch nach Volltexten zu einer gegebenen Referenz gesucht.

Funktionscluster	Signifikanz
Datenintegration	++
Navigation	+
Multimedia-Bestandteile	o
Interaktivität	–
Validierung	–
Trennung Präsentation	o
Formatumwandlung + Medienkanäle	o
automatische Bearbeitung	++
Dynamik	–
Autorenunterstützung	–
Mehrbenutzer	–
Versionierung	–
Rechteverwaltung	+
Suchfunktion	++
Personalisierung	+
Mehrsprachigkeit	o
Workflow	–
Besondere Funktionalität	
Abrechnungsinfrastruktur für kostenpflichtige Artikel	

Tabelle 3.5: Funktionalität des Szenarios „Digitale Bibliothek“

Im Rahmen der Projektgruppe soll das System um die Persönliche Adaptive Digitale Bibliothek (PADDLE) erweitert werden. Diese soll neben der Berücksichtigung persönlicher Präferenzen der einzelnen Benutzer insbesondere auch neue personalisierte Dienste bereitstellen. Dabei spielt die Adaptivität des Systems eine wesentliche Rolle.

Persönliche Präferenzen des Benutzers können funktionaler oder inhaltlicher Art sein:

- Funktionale Präferenzen beziehen sich auf Systemeinstellungen, wie z. B. den Grad der Benutzerführung, oder auf Optionen bezüglich der Präsentation von Ergebnissen.
- Inhaltliche Präferenzen beziehen sich auf thematische Vorlieben des Benutzers. Im einfachsten Fall führt das System Buch über die Dokumente, die der Benutzer schon einmal gesehen hat, und markiert diese Dokumente bei neuen Recherchen entsprechend. Ferner können die generellen Interessengebiete eines Benutzers festgehalten werden, um konkrete Anfragen jeweils entsprechend einzuschränken.

Durch die Beobachtung des Benutzers kann das System viele dieser Präferenzen automatisch erkennen und sich adaptiv verhalten. Die persönliche digitale Bibliothek erweitert die Funktionalität des Systems, um den Nutzen für die Anwender zu steigern:

- Informationsfilterung basiert auf der (expliziten oder impliziten) Formulierung eines Interessenprofils durch den Benutzer. In regelmäßigen Abständen werden alle digitalen Biblio-

theken nach neuen Dokumenten zu den im Profil beschriebenen Themen befragt und die Ergebnisse an den Benutzer weitergeleitet. Ein Spezialfall hiervon ist der Zeitschriften-Benachrichtigungsdienst, der über den Inhalt neu erschienener Hefte bestimmter Zeitschriften informiert.

- Eine persönliche Handbibliothek verwaltet diejenigen Dokumente, mit denen ein Benutzer aktiv arbeitet. Hierzu werden sowohl Metadaten als auch Volltexte der Dokumente lokal gespeichert. Der Benutzer kann Bewertungen (Ratings), Kommentare und Verweise zu den Dokumenten ablegen und insbesondere auch Annotationen innerhalb des Volltextes anbringen. Bei einer umfangreicheren Handbibliothek können die Dokumente gemäß inhaltlicher Kriterien klassifiziert werden.

- Kollaboratives Filtern ist eine Erweiterung der beiden vorgenannten Dienste zur Unterstützung von Benutzergruppen. Dabei können Benutzer ihre Suchergebnisse, Bewertungen und Anmerkungen mit den anderen Mitgliedern der Gruppe teilen. Sie können direkt in den für sie zugänglichen Daten der Gruppe recherchieren, um z.B. das Klassifikationsschema der Gruppe zu durchsuchen oder nach Dokumenten mit positiver Bewertung zu suchen. Bei der normalen Recherche werden sie darauf hingewiesen, wenn ein gefundenes Dokument bereits von anderen Gruppenmitgliedern annotiert wurde. Frühere Qualitätsbeurteilungen können z.B. in das Ranking der Antwortdokumente einfließen.

Funktionscluster	Signifikanz
Datenintegration	++
Navigation	+
Multimedia-Bestandteile	o
Interaktivität	–
Validierung	–
Trennung Präsentation	o
Formatumwandlung + Medienkanäle	o
automatische Bearbeitung	++
Dynamik	–
Autorenunterstützung	–
Mehrbenutzer	–
Versionierung	–
Rechteverwaltung	o
Suchfunktion	++
Personalisierung	++
Mehrsprachigkeit	o
Workflow	–
Besondere Funktionalität	
Unterstützung des Benutzers durch Personalisierung	

Tabelle 3.6: Funktionalität des Szenarios „Digitale Bibliothek“

3.2.7 Autonome Such- und Profildienste für wissenschaftliche Information

Bei den autonomen Such- und Profildiensten[4] steht die Datenintegration im Vordergrund. Diese Dienste bieten unter einer einheitlichen Oberfläche Zugang zu umfangreichen Beständen wissenschaftlicher Publikationen. Eine Hauptfunktionalität ist das Durchsuchen dieser Bestände nach Schlüsselwörtern, die andere Hauptfunktionalität ist die Benachrichtigung des Benutzers, sobald neue Publikationen gefunden werden, die seinen gespeicherten Kriterien entsprechen. Insofern ist dieses Szenario vergleichbar mit dem Szenario der Digitalen Bibliotheken wissenschaftlicher Fachgesellschaften aus Abschnitt 3.2.5. Der Unterschied besteht darin, dass Such- und Profildienste die Publikationen nicht selbst vorhalten, sondern nur auf sie verweisen.

Funktionscluster	Signifikanz
Datenintegration	++
Navigation	+
Multimedia-Bestandteile	o
Interaktivität	–
Validierung	–
Trennung Präsentation	o
Formatumwandlung + Medienkanäle	o
automatische Bearbeitung	++
Dynamik	–
Autorenunterstützung	–
Mehrbenutzer	–
Versionierung	–
Rechteverwaltung	–
Suchfunktion	++
Personalisierung	++
Mehrsprachigkeit	o
Workflow	–
Besondere Funktionalität	
Unterstützung des Benutzers durch Personalisierung	

Tabelle 3.7: Funktionalität des Szenarios „Such- und Profildienste“

3.2.8 Redaktionssysteme für Fachverlage

In Fachverlagen besteht oft das Problem, dass externe Autoren ihre Manuskripte in sehr unterschiedlichen Formaten liefern. Sie benutzen dabei meistens das Textverarbeitungsprogramm

[4] z. B. CiteSeer: `http://citeseer.ist.psu.edu/`

Microsoft Word. Die Publikation stellt sich für den Verlag jedoch sehr aufwändig und problematisch dar, denn die Voraussetzung für eine Veröffentlichung war, die Inhalte zunächst in entsprechende Druckvorlagenformate zu konvertieren. Die Übernahme von Formatierungen aus Textverarbeitungsprogrammen wie Microsoft Word ist jedoch prinzipiell mit vielen Unwägbarkeiten verbunden. Das gängige „Copy & Paste“-Verfahren ist äußerst fehleranfällig und zeitraubend. So ist eine exakte Übernahme von autorenseitigen Formatierungen nicht gewährleistet. Der Verlag muss eine Lösung finden, mit der sich Dokumente externer Autoren möglichst schnell und fehlerfrei unter Wahrung des Qualitätsstandards in ein druckfähiges Format bringen lassen.

Zu diesem Zweck kann ein Redaktionssystem eingesetzt werden, das die Eingabe der Inhalte von jedem beliebigen Ort aus ohne Schulungsaufwand mittels Browser ermöglicht. Das System legt alle Inhalte in einer medienneutralen Form ab. Das ist die Grundvoraussetzung für eine automatische Dokumentenaufarbeitung zur anschließenden Verfügbarmachung. Autoren erstellen ihre Dokumente online in der Redaktionsoberfläche des Systems. Dies geschieht über einen Editor, der den Autoren Möglichkeiten wie Querverweise, Fußnote, Stichwörter etc. anbietet. Bei entsprechender Softwareunterstützung können auch mehrere Autoren gleichzeitig an einem Dokument arbeiten. Ein festgelegter Workflow leitet die Dokumente anschließend automatisch in das Lektorat. Dort erfolgen das Korrekturlesen, die Freigabe zur weiteren Verarbeitung oder die Rückgabe an den Autor mit entsprechenden Änderungswünschen.

Alle Inhalte sind redundanzfrei zentral in einer Datenbank gespeichert. Autoren und Kontrollinstanzen haben jederzeit Zugriff auf das Dokument.

3.2.8.1 Einsatzmöglichkeiten von XML

Der wichtigste Aspekt dieses Szenarios ist die medienneutrale Datenhaltung der Inhalte. Hier ist XML sehr gut für den Einsatz als Datenformat geeignet. Es unterstützt gleichzeitig auch noch die Ausgabe der Inhalte in unterschiedlichen Formaten.

3.2.9 Portale zu Gerichtsentscheidungen

Ein Portal zu Gerichtsentscheidungen[5] sammelt Dokumente von Urteilsverkündigungen von Gerichten und stellt sie unter einer einheitlichen Oberfläche zur Verfügung. Welche Gerichte dabei abgedeckt werden, ist vom Portal abhängig.

Die wichtigste Funktionalität ist die Möglichkeit, den Bestand zu durchsuchen. Dabei werden unterschiedlichen Möglichkeiten angeboten. Meist gibt es zumindest die Möglichkeit, den Bestand im Volltext zu durchsuchen. Daneben wird häufig auch eine Übersicht zur kalendarischen Suche angeboten. Eine weitere Möglichkeit ist die Bereitstellung einer Übersicht neuer Urteile.

[5] z. B. `http://www.caselaw.de/`

Funktionscluster	Signifikanz
Datenintegration	+
Navigation	o
Multimedia-Bestandteile	o
Interaktivität	o
Validierung	+
Trennung Präsentation	++
Formatumwandlung + Medienkanäle	++
automatische Bearbeitung	o
Dynamik	–
Autorenunterstützung	++
Mehrbenutzer	++
Versionierung	++
Rechteverwaltung	+
Suchfunktion	o
Personalisierung	o
Mehrsprachigkeit	o
Workflow	++
Besondere Funktionalität	
Medienneutrale Datenhaltung	

Tabelle 3.8: Funktionalität des Szenarios „Redaktionssysteme für Fachverlage“

3.2.10 eLearning-Plattformen

eLearning-Plattformen stellen eine Plattform für die Entwicklung von eLearning-Angeboten zur Verfügung. Das erstellte eLearning-Angebot soll die Lernenden didaktisch möglichst optimal unterstützen. Dabei gibt es eine Reihe von Herausforderungen im Bereich Lernumgebungen [SFB99]:

- Lernende variieren in Vorwissen und Interessen. Daher ist eine anpassbare Navigation und Präsentation des gegebenen Lerninhaltes und der Struktur nötig.
- Autoren müssen existierendes Material wiederfinden und angepasst an neue Zielgruppen zu neuem Lernmaterial kombinieren können.

In [SFB99] wird ein Meta-Modell für die konzeptuelle und navigationale Beschreibung von Kursmaterial vorgestellt. Das Meta-Modell dient als Basis für eine XML Markup Sprache („Learning Material Markup Language“). XSL Stylesheets werden dabei eingesetzt, um XML Repräsentationen des Lernmaterials anhand von Metadaten neu zu strukturieren.

Funktionscluster	Signifikanz
Datenintegration	++
Navigation	+
Multimedia-Bestandteile	o
Interaktivität	–
Validierung	–
Trennung Präsentation	o
Formatumwandlung + Medienkanäle	o
automatische Bearbeitung	+
Dynamik	–
Autorenunterstützung	–
Mehrbenutzer	–
Versionierung	–
Rechteverwaltung	–
Suchfunktion	++
Personalisierung	–
Mehrsprachigkeit	–
Workflow	–
Besondere Funktionalität	
Medienneutrale Datenhaltung	

Tabelle 3.9: Funktionalität des Szenarios „Portale zu Gerichtsentscheidungen“

3.2.11 Webbasierte Ticketingsysteme

Webbasierte Ticketingsysteme[6] ermöglichen den einfachen Verkauf von Eintrittskarten für unterschiedliche Veranstaltungen über das Internet. Die Zielgruppen für den Einsatz dieser Ticketingsysteme sind dabei sowohl Veranstalter als auch Vorverkaufsstellen sowie Kunden. Ein einzelnes Ticketingsystem kann dabei mehrere Veranstalter gleichzeitig getrennt voneinander bedienen.

Tickets können über einen individuellen Saalplan platzgenau reserviert und direkt vor Ort ausgedruckt werden. Der Saalplan kann individuell für jede Veranstaltung erstellt werden. Einige Ticketingsysteme bieten dem Kunden zudem die Möglichkeit, sich die Sichtperspektive vom gewünschten Platz zur Bühne anzeigen zu lassen.

3.2.12 Newsticker

Ein Newsticker berichtet über die neuesten Meldungen in einem Themengebiet. Wobei das Themengebiet auch sehr breit gefasst werden kann. Beispiele für sehr bekannte Newsticker sind

[6] z. B. ReserviX: `http://www.reservix.de`

Funktionscluster	Signifikanz
Datenintegration	o
Navigation	++
Multimedia-Bestandteile	++
Interaktivität	++
Validierung	–
Trennung Präsentation	+
Formatumwandlung + Medienkanäle	o
automatische Bearbeitung	o
Dynamik	–
Autorenunterstützung	–
Mehrbenutzer	–
Versionierung	–
Rechteverwaltung	–
Suchfunktion	o
Personalisierung	++
Mehrsprachigkeit	o
Workflow	–

Tabelle 3.10: Funktionalität des Szenarios „eLearning-Plattformen“

„heise online – 7-Tage-News“[7] und „Schlagzeilen – Spiegel Online“[8]. Das erste Beispiel ist ein reiner Newsticker, das zweite Beispiel stellt mehr eine andere Sicht auf das Nachrichtenportal von Spiegel Online dar. Es ist aber trotzdem hier aufgeführt, da es im Grundcharakter keinen Unterschied zu reinen Newstickern besitzt.

Ein Newsticker führt genau wie ein Blog die Artikel in einer umgekehrt chronologischen Reihenfolge auf, d. h. neueste Meldungen stehen auf der Seite ganz oben. Dabei kann neben der reinen Schlagzeile auch ein kurzer Textausschnitt angezeigt werden. Wesentlich ist, dass von der Schlagzeile eine Verknüpfung auf den vollständigen Artikel existiert. Oft kann eine Meldung vom Leser kommentiert werden. Dabei werden üblicherweise Antworten auf schon vorhandene Kommentare entsprechend gekennzeichnet dargestellt, so dass der Diskussionsverlauf erkennbar bleibt.

So gut wie alle Newsticker bieten die Möglichkeit, einen RSS-Stream zu abonnieren, so dass neue Meldungen verfügbar sind, ohne dass der Leser die gesamte Seite neu laden muss.

XML eignet sich besonders gut für dieses Szenario, da es wegen seiner Medienneutralität sowohl die Umwandlung in konkrete Präsentationen als auch die Bereitstellung als RSS erlaubt. Daneben unterstützt es die automatische Generierung der Newstickerübersichtsseiten aus den Daten des Artikels (Extraktion der Überschrift und eines Textanrisses), ohne dass die Daten redundant gespeichert sein müssen, was leicht zu Inkonsistenzen führen kann.

[7] `http://www.heise.de/newsticker/`
[8] `http://www.spiegel.de/schlagzeilen/`

Funktionscluster	Signifikanz
Datenintegration	o
Navigation	+
Multimedia-Bestandteile	++
Interaktivität	++
Validierung	–
Trennung Präsentation	++
Formatumwandlung + Medienkanäle	o
automatische Bearbeitung	+
Dynamik	++
Autorenunterstützung	++
Mehrbenutzer	++
Versionierung	o
Rechteverwaltung	++
Suchfunktion	o
Personalisierung	o
Mehrsprachigkeit	o
Workflow	o
Besondere Funktionalität	
Integration von Zahlungsverfahren Selbstdruck der Tickets	

Tabelle 3.11: Funktionalität des Szenarios „Webbasierte Ticketingsysteme“

3.2.13 Foren

Forensysteme unterstützen die Diskussion und den Wissensaustausch von Nutzern, indem sie einen Informationsraum anbieten, in dem sich die Nutzer austauschen können. Ein Forum beschäftigt sich normalerweise mit einem beschränkten Themengebiet. Sie gibt es beispielsweise Sportforen, Foren zu bestimmten Softwareprodukten, Foren zu Linux-Distributionen etc. Innerhalb eines Forums wird das Themengebiet in unterschiedliche Teilaspekte zerlegt, so dass eine thematische Zuordnung der Diskussionen zu Aspekten erfolgt. Oft beginnt eine Diskussion mit einer Frage oder Problemstellung. Andere Nutzer können daraufhin eigene Beiträge schreiben, die in der Regel linear geordnet unterhalb des ersten Beitrags erscheinen. Üblicherweise können alle Internetnutzer die Diskussionen verfolgen, jedoch können nur angemeldete Benutzer einen Antwortbeitrag schreiben.

Beispielsoftware: phpBB[9].

[9] phpBB.com: `http://www.phpbb.com/`

Funktionscluster	Signifikanz
Datenintegration	+
Navigation	++
Multimedia-Bestandteile	+
Interaktivität	+
Validierung	o
Trennung Präsentation	++
Formatumwandlung + Medienkanäle	+
automatische Bearbeitung	+
Dynamik	–
Autorenunterstützung	o
Mehrbenutzer	o
Versionierung	o
Rechteverwaltung	o
Suchfunktion	+
Personalisierung	o
Mehrsprachigkeit	o
Workflow	o
Besondere Funktionalität	
Oft Zusatzangebot von Nachrichtenportalen automatische RSS-Feed Erstellung (=Automatische Verarbeitung)	

Tabelle 3.12: Funktionalität des Szenarios „Newsticker“

3.2.14 Freizeitcommunities

Freizeitcommunities[10] ermöglichen es den Nutzern, über Sport- und Freizeitinteressen Gleichgesinnte aus der näheren Umgebung zu finden. Dazu kann jeder Benutzer aus einer Liste von Interessen sein individuelles Profil erstellen und über eine Suchfunktion nach anderen Nutzern mit diesen Interessen suchen und über ein schwarzes Brett aktiv nach Interessierten zu suchen oder selbst auf dortige Einträge zu antworten. Die Kommunikation erfolgt in der Regel über ein internes Nachrichtensystem, so dass keine persönlichen Daten oder Adressinformationen mitgeteilt werden. Sinnvoll ist eine Eingrenzung des örtlichen Suchbereichs, so dass nur Benutzer bzw. Einträge von Benutzer angezeigt werden, die in einem bestimmten Umkreis wohnen.

Eine Verbesserungsmöglichkeit vieler bestehender Systeme liegt darin, automatische Benachrichtigungen beim Veröffentlichen relevanter Einträge am schwarzen Brett zu versenden, wenn bestimmte Kriterien erfüllt sind.

[10] z. B. new-in-town: `http://www.new-in-town.de`

Funktionscluster	Signifikanz
Datenintegration	o
Navigation	+
Multimedia-Bestandteile	o
Interaktivität	++
Validierung	–
Trennung Präsentation	++
Formatumwandlung + Medienkanäle	o
automatische Bearbeitung	o
Dynamik	–
Autorenunterstützung	++
Mehrbenutzer	++
Versionierung	–
Rechteverwaltung	++
Suchfunktion	++
Personalisierung	+
Mehrsprachigkeit	+
Workflow	o
Besondere Funktionalität	
Oft Zusatzangebot von Nachrichtenportalen automatische RSS-Feed Erstellung (=Automatische Verarbeitung)	

Tabelle 3.13: Funktionalität des Szenarios „Foren“

3.2.15 Online-Shops

In diesem Szenario geht es darum, Waren und Dienstleistungen über das Internet anzubieten und zu verkaufen. Eines der bekanntesten Beispiele hierfür ist der Online-Shop von Amazon. Händler möchten ihre Produkte möglichst ansprechend, übersichtlich und auffindbar anbieten, d. h. potentielle Kunden sollen alle für sie interessanten Informationen zu einem Produkt leicht finden können und der Kauf soll auf möglichst einfache Art und Weise abgewickelt werden können.

3.2.16 Technische Dokumentation

Unter dem Begriff Technische Dokumentation versteht man nicht nur die reine Gebrauchsanleitung für Produkte, sondern sämtliche Informationen, die für ein Produkt in seinem Lebenszyklus von Bedeutung sind. Das betrifft alle internen prozess- und workflow-begleitenden Informationen und das daraus resultierende Wissen für die fachgerechte, effiziente und einfache Herstellung, Wartung und Anwendung der Produkte. Hierunter fallen zum Beispiel Entwicklungsdokumentationen, Fertigungsunterlagen, Reparaturanleitungen, Stücklisten, Datenblätter,

Funktionscluster	Signifikanz
Datenintegration	o
Navigation	+
Multimedia-Bestandteile	o
Interaktivität	++
Validierung	–
Trennung Präsentation	+
Formatumwandlung + Medienkanäle	–
automatische Bearbeitung	o
Dynamik	–
Autorenunterstützung	++
Mehrbenutzer	++
Versionierung	–
Rechteverwaltung	+
Suchfunktion	++
Personalisierung	++
Mehrsprachigkeit	o
Workflow	–
Besondere Funktionalität	
Internes Nachrichtensystem	

Tabelle 3.14: Funktionalität des Szenarios „Freizeitcommunities“

Ersatzteilkataloge, Betriebsanleitungen, Supportunterlagen, Kundenliteratur, Diagnosehandbücher und umfangreiche Hilfesysteme.

3.2.17 Internet Banking

Beim Internet Banking steht der Zugriff auf Banking-Funktionalitäten (Girokonten, Depots, sonstige Geldanlagen) im Mittelpunkt. Ein zentraler Aspekt dabei ist die Bereitstellung von Kontoauszügen, Orderberichten und sonstigen Unterlagen über das Internet. Neben den reinen Bankgeschäften werden üblicherweise auch Verwaltungsmöglichkeiten angeboten.

3.2.18 Internetauktionen

Bei Internetauktionen geht es wie bei Online-Shops um den Verkauf von Waren. Bei Auktionen können jedoch auch Privatpersonen nicht mehr benötigte Waren zum Verkauf anbieten. D. h. üblicherweise steht nicht eine einzelne Firma hinter allen angebotenen Waren, sondern dabei handelt es sich um viele einzelne Personen oder Firmen, die jeweils selbst ihren Artikel beschreiben und zum Verkauf stellen. Daher ist die Interaktivität in diesem Szenario sehr wichtig.

Funktionscluster	Signifikanz
Datenintegration	+
Navigation	++
Multimedia-Bestandteile	++
Interaktivität	o
Validierung	–
Trennung Präsentation	+
Formatumwandlung + Medienkanäle	o
automatische Bearbeitung	+
Dynamik	–
Autorenunterstützung	–
Mehrbenutzer	–
Versionierung	–
Rechteverwaltung	–
Suchfunktion	++
Personalisierung	+
Mehrsprachigkeit	o
Workflow	–
Besondere Funktionalität	
Produktempfehlungen basierend auf Heuristiken	

Tabelle 3.15: Funktionalität des Szenarios „Online-Shops“

3.2.19 Investor-Relations-Sites

Investor-Relations-Sites[11] bieten für Investoren an einer zentralen Stelle gesammelt Informationen über ein Unternehmen an. Es geht dabei hauptsächlich um Nachrichten über das Unternehmen, Geschäftsberichte, Termine für Veranstaltungen und Informationen über die Börsenentwicklung bei börsennotierten Unternehmen.

üblicherweise können die Anforderungen an die Funktionalität mehr als vollständig von Content-Management-Systemen abgedeckt werden. IR-Sites können ihren Nutzern jedoch durch den Transfer von Funktionalität aus anderen Szenarien einen höheren Nutzen bringen. So wäre eine Benachrichtigung, sobald neue Nachrichten, Berichte etc. bereitstehen, ein erster Schritt. Auch das Anbieten eines RSS-Feeds ist denkbar, so dass RSS-Clientprogramme automatisch neue Nachrichten empfangen und den Benutzer benachrichtigen.

[11] z. B. Volkswagen Investor Relations: `http://www.volkswagen-ir.de/`

Funktionscluster	Signifikanz
Datenintegration	+
Navigation	++
Multimedia-Bestandteile	++
Interaktivität	o
Validierung	–
Trennung Präsentation	+
Formatumwandlung + Medienkanäle	+
automatische Bearbeitung	+
Dynamik	–
Autorenunterstützung	o
Mehrbenutzer	+
Versionierung	++
Rechteverwaltung	+
Suchfunktion	++
Personalisierung	o
Mehrsprachigkeit	+
Workflow	+

Tabelle 3.16: Funktionalität des Szenarios „Technische Dokumentation“

3.2.20 Filmdatenbanken

Filmdatenbanken[12] stellen eine Datenbank für Kinofilme, Fernsehfilme, TV-Serien, Schauspieler, Regisseure etc. bereit. üblicherweise gibt es für die Nutzer eine Möglichkeit, Kommentare und Bewertungen abzugeben, aus denen mit bestimmten Algorithmen Ranglisten berechnet werden.

Ansonsten bieten Filmdatenbanken bislang nur geringe Möglichkeiten zur Interaktion. Eine Möglichkeit zur Weiterentwicklung wäre beispielsweise eine automatische Benachrichtigung des Nutzers, sobald neue Filme aus bestimmten Genres ins Kino kommen oder sich Änderungen in den Ranglisten ergeben oder Filme bestimmte Bewertungskriterien erfüllen etc.

3.2.21 TV-Zeitschriften im Internet

Online TV-Zeitschriften bieten das Fernsehprogramm im Internet an. Dabei gibt es verschiedene Möglichkeiten nach Programmen zu suchen: gezielt nach Uhrzeit und Tag, was aktuell oder in kürze laufen wird und auch Suche nach Programmtitel.

Eine Personalisierung wird normalerweise nicht unterstützt. Dabei gäbe es gerade hier interessante Möglichkeiten. Etwa Ausblenden der vom Benutzer als uninteressant eingestuften Sendungen, automatische Benachrichtigungen bei Programmänderungen, Benachrichtigungen vor

[12] z. B. The Internet Movie Database: `http://www.imdb.org/`

Funktionscluster	Signifikanz
Datenintegration	+
Navigation	++
Multimedia-Bestandteile	+
Interaktivität	++
Validierung	–
Trennung Präsentation	+
Formatumwandlung + Medienkanäle	++
automatische Bearbeitung	+
Dynamik	+
Autorenunterstützung	–
Mehrbenutzer	++
Versionierung	–
Rechteverwaltung	++
Suchfunktion	o
Personalisierung	++
Mehrsprachigkeit	o
Workflow	–

Tabelle 3.17: Funktionalität des Szenarios „Technische Dokumentation“

Beginn einer Sendung oder wenn neue Sendungen ins Programm aufgenommen werden, die bestimmten Kriterien genügen etc.

3.2.22 Marktplätze

Dieses Szenario beschäftigt sich im wesentlichen mit Marktplätzen für Arbeitsstellen, Automobile, Immobilien etc. Oft sind diese Marktplätze als Zusatzangebote in andere Szenarien wie Nachrichtenportale integriert. Das Szenario besitzt eine Ähnlichkeit mit den Szenarien der Online-Shops und der Internetauktionen. Der Unterschied besteht zum einen in der Integration in weitere Szenarien, zum anderen in der üblicherweise vorhandenen Spezialisierung auf eine einzige Produktkategorie.

3.3 Ergebnisse

Bei der Betrachtung der Signifikanz der Funktionscluster in den einzelnen Szenarien zeigen sich deutliche Unterschiede in Bezug auf die Signifikanz für die Gesamtmenge der untersuchten Szenarien. In der Abbildung 3.2 auf Seite 81 ist die szenarienspezifische Signifikanz der Funktionscluster übersichtsartig zusammengefasst. Die Funktionscluster sind von links nach rechts absteigend nach ihren Signifikanzen für alle Szenarien sortiert.

Funktionscluster	Signifikanz
Datenintegration	++
Navigation	++
Multimedia-Bestandteile	++
Interaktivität	++
Validierung	–
Trennung Präsentation	+
Formatumwandlung + Medienkanäle	o
automatische Bearbeitung	+
Dynamik	++
Autorenunterstützung	++
Mehrbenutzer	++
Versionierung	–
Rechteverwaltung	++
Suchfunktion	++
Personalisierung	++
Mehrsprachigkeit	o
Workflow	–

Tabelle 3.18: Funktionalität des Szenarios „Internetauktionen“

3.3.1 Funktionscluster mit hoher Signifikanz

3.3.1.1 Suchmöglichkeiten

Die Möglichkeit zur Suche in den vorhandenen Dokumenten ist mit Abstand bei den meisten Szenarien als sehr wichtig einzustufen. Wenn der Dokumentenbestand einen gewissen Umfang und Komplexität erreicht, ist eine direkte Suche in den Dokumenteninhalten oft die schnellste Methode, um bestimmte Informationen aufzufinden. Dies kann man auch an der Bedeutung von Suchmaschinen für das Internet erkennen. Viele Szenarien, die mit großen Dokumentenbeständen arbeiten, bieten eine Suche über den kompletten Dokumentenbestand an. Beim Szenario der Autonomen Such- und Profildienste ist dies ein essentieller Bestandteil.

3.3.1.2 Navigation

Das World Wide Web wäre ohne Verknüpfungen, die es von den Hypertext-Systemen geerbt hat, undenkbar. Auf die einzelnen Vorteile von Hypertext-Systemen soll an dieser Stelle nicht weiter eingegangen werden, es soll nur die bedeutendste Auswirkung genannt werden: Es ist einfach möglich, von einer linearen Lesart abzuweichen bzw. der Nutzer kann über sein Voranschreiten selbst und mit Unterstützung durch das System nach seinen Wünschen und Bedürfnissen entscheiden.

Funktionscluster	Signifikanz
Datenintegration	+
Navigation	++
Multimedia-Bestandteile	+
Interaktivität	–
Validierung	–
Trennung Präsentation	+
Formatumwandlung + Medienkanäle	+
automatische Bearbeitung	–
Dynamik	–
Autorenunterstützung	–
Mehrbenutzer	–
Versionierung	–
Rechteverwaltung	–
Suchfunktion	+
Personalisierung	–
Mehrsprachigkeit	o
Workflow	–

Tabelle 3.19: Funktionalität des Szenarios „Investor-Relations-Sites“

Viele Szenarien nutzen folglich intensiv die Möglichkeiten zur Verknüpfung von Dokumenten. Dabei kann zum einen die Verknüpfungsstruktur direkt ausgewertet werden (z. B. durch Suchmaschinen), zum anderen muss der Benutzer geeignet unterstützt werden, um sich nicht in der Menge der Verknüpfungen zu verlieren.

Navigationsunterstützungsverfahren lassen sich in zwei Hauptkategorien unterteilen[RM02]:

1. Eingebettete Navigationsunterstützung
2. Ergänzende Navigationsunterstützung

Die eingebettete Navigationsunterstützung ist direkt in die Dokumente integriert und lässt sich wiederum unterscheiden in globale, lokale und kontextuelle Navigationsunterstützung.

Globale Navigationsunterstützung ist per definitionem in allen Dokumenten vorhanden. Sie erlauben den direkten Zugang zu Schlüsselbereichen und -funktionen.

Lokale Navigationsunterstützung ermöglichen es dem Benutzer, die nähere Umgebung des Dokuments zu überblicken. Es gibt meist neben dem globalen noch ein bis zwei lokale Navigationsunterstützungen.

Kontextuelle Navigationsunterstützung kümmert sich um die Verknüpfungen, die nicht sauber in die strukturierten Kategorien der globalen und lokalen Navigationsunterstützungen passen. Diese Verknüpfungen sind spezifisch für eine bestimmte Seite, ein bestimmtes Dokument oder

Funktionscluster	Signifikanz
Datenintegration	o
Navigation	++
Multimedia-Bestandteile	++
Interaktivität	+
Validierung	–
Trennung Präsentation	+
Formatumwandlung + Medienkanäle	o
automatische Bearbeitung	o
Dynamik	–
Autorenunterstützung	–
Mehrbenutzer	–
Versionierung	–
Rechteverwaltung	–
Suchfunktion	++
Personalisierung	o
Mehrsprachigkeit	o
Workflow	–

Tabelle 3.20: Funktionalität des Szenarios „Filmdatenbanken“

ein bestimmtes Produkt. Mit Hilfe der kontextuellen Navigationselemente ist es möglich, ein Netz zu flechten, das verschiedene Dokumente und Bereiche der Site miteinander verbindet.

Die ergänzenden Navigationsunterstützungen sind von den eigentlichen Dokumenten unabhängige Bestandteile eines Publikationssystems. Es handelt sich hierbei um Sitemaps, Indizes, Guides und Suchfunktionen.

Sitemaps spiegeln die organisationelle Struktur wider. Sie ermöglichen den Nutzern den direkten Zugriff auf die Dokumente der Site. Aufgrund ihres hierarchischen Aufbaus machen sie nur Sinn, wenn die Struktur der Site ebenfalls hierarchisch aufgebaut ist.

Indizes präsentieren Schlüsselwörter und Ausdrücke in einer relativ flachen Liste, die üblicherweise nur ein oder zwei Stufen tief ist.

Guides sind geführte Touren, Tutorials oder Mikro-Portale, die auf eine bestimmte Nutzergruppe, ein bestimmtes Thema oder eine bestimmte Aufgabe beschränkt sind. Guides ergänzen die bestehenden Möglichkeiten, den Dokumentenbestand einer Site zu überblicken und sich durch ihn zu bewegen.

3.3.1.3 Multimediale Bestandteile

Multimediale Bestandteile dienen zur inhaltlichen Aufwertung von Dokumenten. Neben Texten können dann auch Grafiken, Video- und Audiobestandteile dargestellt werden. Die Verbreitung

Funktionscluster	Signifikanz
Datenintegration	+
Navigation	+
Multimedia-Bestandteile	+
Interaktivität	o
Validierung	–
Trennung Präsentation	+
Formatumwandlung + Medienkanäle	o
automatische Bearbeitung	–
Dynamik	+
Autorenunterstützung	–
Mehrbenutzer	–
Versionierung	–
Rechteverwaltung	–
Suchfunktion	++
Personalisierung	o
Mehrsprachigkeit	o
Workflow	–

Tabelle 3.21: Funktionalität des Szenarios „TV-Zeitschriften“

multimedialer Bestandteile hat zusammen mit der weiteren Verbreitung von Breitbandinternetzugängen deutlich zugenommen. Die Verbreitung wird auch noch weiter steigen. Im Internet existieren bereits seit längerem Musik-Angebote und auch komplette Filme werden zunehmend bereitgestellt. Die gleichzeitige Nutzung mehrerer Wahrnehmungssinne ist v. a. für das Szenario eLearning interessant.

3.3.1.4 Interaktivität

Die Interaktivität, also die Möglichkeit des Benutzers, Dokumente nicht nur passiv zu konsumieren, sondern auch aktiv mit ihnen zu interagieren, spielt in einer Reihe von Szenarien eine wichtige Rolle. Bei manchen Szenarien geht es hauptsächlich darum, dass der Benutzer die Dokumente ergänzen kann (z. B. durch Kommentare bei Blogs). Andere Szenarien wie etwa Online-Auktionen funktionieren nur mit Interaktivität durch den Benutzer.

3.3.1.5 Trennung der Präsentation

Eine wichtige Eigenschaft strukturierter Dokumente, nämlich die Trennung der Präsentation vom Inhalt und der Struktur, hat sich durch die Untersuchung der Szenarien bestätigt. Sie hat sich zwar nicht in allen Szenarien als wichtig erwiesen, aber doch in weitaus den meisten.

Funktionscluster	Signifikanz
Datenintegration	+
Navigation	+
Multimedia-Bestandteile	++
Interaktivität	++
Validierung	–
Trennung Präsentation	++
Formatumwandlung + Medienkanäle	o
automatische Bearbeitung	o
Dynamik	–
Autorenunterstützung	++
Mehrbenutzer	++
Versionierung	–
Rechteverwaltung	++
Suchfunktion	++
Personalisierung	+
Mehrsprachigkeit	o
Workflow	o

Tabelle 3.22: Funktionalität des Szenarios „Marktplätze“

3.3.1.6 Personalisierung

Der Funktionscluster der Personalisierung spielt aus zwei Hauptgründen in vielen Szenarien eine bedeutende Rolle. Zum einen ermöglicht es dem Benutzer, das Informationsangebot auf für ihn interessante Bereiche zu begrenzen bzw. von vornherein uninteressante Dokumente auszuschließen, was dem Information Overload entgegenwirkt.

Zum anderen führt eine Personalisierung durch den Benutzer auch dazu, dass dieser stärker an den Dienst gebunden ist, da er bei einem Wechsel zu einem anderen Dienst diese Personalisierung normalerweise erneut leisten muss.

3.3.2 Funktionscluster mit niedriger Signifikanz

3.3.2.1 Validierung

Der Funktionscluster „Validierung“ ist nur in einem einzigen Szenario „Redaktionssysteme für Fachverlage“ von wichtiger Bedeutung. In allen anderen Szenarien ist diese Funktionalität von keiner bis geringer Bedeutung. Dieses Ergebnis mag auf den ersten Blick überraschend erscheinen, ist doch die Möglichkeit zur Validierung der Struktur eines Dokuments ein häufig genannter Vorteil von strukturierten Dokumenten. Mit dieser Möglichkeit verhält es sich jedoch so ähnlich wie mit einigen Bereichen der IT-Security. Sie verfolgt normalerweise keinen Selbstzweck.

++: sehr wichtig +: wichtig o: neutral -: unwichtig	Suchmöglichkeit	Navigation	Multimediateile	Interaktivität	Autorenunterstützung	Trennung Präsentation	Mehrbenutzer	Personalisierung	Rechte Management	Datenintegration	autom. Bearbeitung	Formatunwandlung/Medienkänale	Versionierung	Dynamik	Workflow	Mehrsprachigkeit	Validierung	Sonstiges
Blog	+	+	+	++	++	+	o	-	o	o	+	o	o	-	-	o	-	autom. Kategorisierung?
Wiki	++	++	+	+	++	++	+	-	o	o	o	+	++	-	o	o	o	WYSIWYG Editor, XML Datenformat
Digitale Zeitung	+	+	++	-	-	o	-	+	++	+	-	++	-	-	-	o	-	Personalisierung, Navigation verbessern
Digitale Bibliothek	++	+	o	-	-	o	-	+	+	++	++	o	-	-	-	o	-	XML-Datenlieferung, zert. Dok.server
Personliche Digitale Bibliothek	++	+	o	-	-	o	-	++	o	++	++	o	-	-	-	o	-	
Red.systeme von Fachverlag.	o	o	o	o	++	++	++	o	+	+	o	++	++	-	++	o	+	medienneutrale Datenhaltung
Autonome Such-+Profildienste	++	+	o	-	-	o	-	++	-	++	++	o	-	-	-	o	-	
Portale zu Gerichtsentsch.	++	+	o	-	-	o	-	-	-	++	+	o	-	-	-	-	-	
eLearning	o	++	++	++	-	+	-	++	-	o	o	o	-	-	-	o	-	
Online-Shops	++	++	++	o	-	+	-	+	-	+	+	o	-	-	-	o	-	Prod.empfehlungen basierend auf Profil
Technische Dokumentation	++	++	++	o	o	+	+	o	+	+	+	+	++	-	+	+	-	
Banking	o	++	+	++	-	+	++	++	++	+	+	++	-	+	-	o	-	
Online-Auktionen	++	++	++	++	++	+	++	++	++	++	+	o	-	++	-	o	-	
Foren	++	+	o	++	++	++	++	+	++	o	o	o	-	-	o	+	-	
Investor-Relations-Sites	+	+	+	-	-	+	-	-	-	+	-	+	-	-	-	o	-	
Freizeitcommunities	++	+	o	++	++	+	++	++	+	o	o	-	-	-	-	o	-	autom. Benachrichtigung bei interess.
Ticketsysteme	o	+	++	++	++	++	++	o	++	o	+	o	o	++	o	o	-	
Newsticker	+	++	+	+	o	++	o	o	o	+	+	+	o	-	o	o	o	
TV-Zeitschriften	++	+	+	o	-	+	-	o	-	+	-	o	-	+	-	o	-	
Filmdatenbanken	++	++	++	+	-	+	-	o	-	o	o	o	-	-	-	o	-	
Nachrichtenportale	++	++	++	o	+	++	+	o	+	+	o	+	-	+	+	o	-	Suche, Infoverteilung, Personalisierung
Marktplätze (Job,Kfz,Immo)	++	+	++	++	++	++	++	+	++	+	o	o	-	-	o	o	-	

Abbildung 3.2: Szenarienspezifische Signifikanz der Funktionscluster

So wie ein sicheres System gegenüber einem unsicheren keinen unmittelbaren Nutzungsvorteil bietet, werden auch Dokumentenstrukturen nicht nur deshalb validiert, weil sie technisch validierbar sind. Nur wenn bestimmte Anforderungen gestellt werden, wie dies beim Szenario „Redaktionssysteme für Fachverlage“ der Fall ist, spielt die Validierung der Struktur eine wichtige Rolle. Zu diesen Anforderungen gehört die Integration von Dokumenten von verschiedenen Autoren bzw. Quellen in einen Gesamtbestand bei gleichzeitiger Sicherstellung der Einhaltung bestimmter Strukturvorgaben, die für die einheitliche Formatierung, Nutzung und Weiterverarbeitung erforderlich sind.

3.3.2.2 Dynamik

Ebenfalls eher überraschend ist, dass Dynamik in den meisten Szenarien keine bedeutende Rolle spielt. Sie ist nur dann sehr wichtig, wenn das Szenario eine Interaktivität mit dem Nutzer hoch gewichtet und die Entscheidungen der Nutzer bei dieser Interaktivität maßgeblich von dem aktuellen Datenbestand des Systems abhängt – welcher selbst wiederum durch die Nutzerinteraktivität verändert wird. Dies ist beispielsweise in den Szenarien für Ticketsysteme und Online-Auktionen der Fall.

3.3.2.3 Mehrsprachigkeit

Ebenso wie der Funktionscluster der Validierung stellt die Unterstützung der Mehrsprachigkeit keinen Selbstzweck dar. Interessanterweise spielt die Mehrsprachigkeit trotz der immer wieder betonten und offensichtlichen Globalität des Internet in den meisten Szenarien für Publikationsanwendungen keine wichtige Rolle. Vorteile einer Unterstützung sind zwar nicht von der Hand zu weisen, kommen aber nur selten zum Tragen. Dies liegt häufig daran, dass solche Publikationsanwendungen – obwohl sie weltweit für jede Person zugänglich sind – nur für bestimmte Personengruppen relevant sind, die dann normalerweise dieselbe Sprache beherrschen. Darunter fallen regionale Angebote wie Freizeitcommunities oder auch regionale Nachrichtenportale.

3.3.2.4 Workflow

Die Unterstützung einer Workflow-Funktionalität ist nur dann relevant, wenn mehrere Personen an der Erstellung eines Dokuments beteiligt sind oder wenn erstellte Dokumente vor der Verfügbarmachung einer gesonderten Überprüfung unterzogen werden. In den meisten der beschriebenen Szenarien ist dies nicht der Fall. Hier kommt in der Regel der optimistischere Ansatz zum tragen, dass Dokumente nachträglich bearbeitet werden, wenn der Bedarf hierfür bekannt wird.

3.3.2.5 Versionierung

Hierbei handelt es sich ebenfalls um einen Funktionscluster, der keinen Selbstzweck verfolgt. Eine Ausnahme gibt es nur für diejenige Szenarien, in denen es wichtig ist, Änderungen an Dokumenten zurückverfolgen und verschiedene Versionen desselben Dokuments abruf- und vergleichbar zu haben. Beispielsweise ist dies für das Szenario der Wikis sehr wichtig, da hier in der Regel jeder Internetnutzer an den Dokumenten Veränderungen vornehmen darf.

3.4 Induktive Klassenbildung

In Abschnitt 3.2 wurden Szenarien für den Einsatz von Publikationssystemen untersucht. Anhand der Signifikanzen der Funktionscluster für jedes Szenario lassen sich mehrere Szenarien zu unterschiedlichen Klassen gruppieren. Dabei werden paarweise jeweils zwei Szenarien A und B aus der Menge der untersuchten Szenarien miteinander verglichen. Als Ähnlichkeitsmaß dient die Anzahl der Funktionscluster, die in A und in B dieselbe Signifikanz aufweisen. Je größer diese Anzahl ist, desto ähnlicher sind sich die beiden Szenarien. Jede gefundene Klasse von Szenarien besitzt also sehr ähnliche Signifikanzen der Funktionscluster.

Abbildung 3.3 zeigt für jedes Szenario die Anzahl der Übereinstimmungen mit den anderen Szenarien bei der Signifikanz der Funktionscluster. Erwähnt werden soll, dass es keine zwei Szenarien gibt, die in allen Signifikanzen übereinstimmen. Es wurden daher alle Szenarien betrachtet, die in mindestens elf (also rund zwei Drittel) der insgesamt 17 Signifikanzen übereinstimmen.

	Marktplätze	Nachrichtenportale	Filmdatenbanken	TV-Zeitschriften	Newsticker	Ticketsysteme	Freizeitcommunities	IR-Sites	Foren	Online-Auktionen	Banking	Technische Dokumentation	Online-Shops	eLearning	Portale zu Gerichtsentsch.	Autonome Such-+Profildienste	Red.systeme von Fachverlag.	Persönliche Digitale Bibliothek	Digitale Bibliothek	Digitale Zeitung	Wiki
Blog	7	2	7	7	8	9	9	9	7	8	7	4	7	8	7	6	3	7	6	6	7
Wiki	7	7	7	3	10	5	6	5	7	4	3	6	4	5	3	3	6	4	3	2	
Digitale Zeitung	9	5	8	9	4	5	6	12	6	6	8	4	10	8	9	10	4	10	11		
Digitale Bibliothek	8	5	9	9	2	4	9	9	8	7	5	4	10	8	13	15	4	15			
Persönliche Digitale Bibliothek	7	4	9	9	3	4	9	9	7	8	6	3	9	9	13	16	3				
Red.systeme von Fachverlag.	7	7	4	4	5	6	7	3	6	3	5	6	4	4	2	3					
Autonome Such-+Profildienste	7	4	10	10	2	4	9	10	7	8	6	3	10	10	14						
Portale zu Gerichtsentsch.	6	3	9	9	2	4	7	10	7	7	5	4	10	8							
eLearning	8	6	14	9	3	7	10	9	7	10	10	5	12								
Online-Shops	9	8	13	12	5	5	7	10	6	10	9	9									
Technische Dokumentation	5	11	7	6	7	4	5	5	4	6	5										
Banking	7	6	7	9	5	7	8	8	5	11											
Online-Auktionen	10	6	9	7	3	10	10	5	8												
Foren	14	5	7	5	3	10	11	4													
Investor-Relations-Sites	6	5	9	12	6	3	7														
Freizeitcommunities	10	6	9	7	2	7															
Ticketsysteme	11	5	6	5	6																
Newsticker	5	6	5	4																	
TV-Zeitschriften	7	8	11																		
Filmdatenbanken	8	8																			
Nachrichtenportale	8																				

Abbildung 3.3: Szenarienvergleich

In der Abbildung 3.4 auf Seite 84 ist ein Graph dargestellt, der die wichtigsten Szenarien als Knoten zeigt. Als Kantengewichte sind die jeweiligen Ähnlichkeitswerte zwischen den von der Kante verbundenen Szenarien angegeben. Es lassen sich folgende Szenarien zu jeweils einer Klasse zusammenfassen:

- Digitale Bibliotheken, Portale zu Gerichtsentscheidungen, Autonome Such- und Profildienste, Persönliche Bibliotheken

  Zu den wichtigsten Funktionsclustern in dieser Klasse von Szenarien gehören:

  - Datenintegration
  - Navigation
  - automatisches Bearbeiten
  - Suchfunktion
  - Personalisierung (nur bei Portalen zu Gerichtsentscheidungen weniger wichtig)

- Banking, Internetauktionen

  Zu den wichtigsten Funktionsclustern in dieser Klasse von Szenarien gehören:

  - Datenintegration

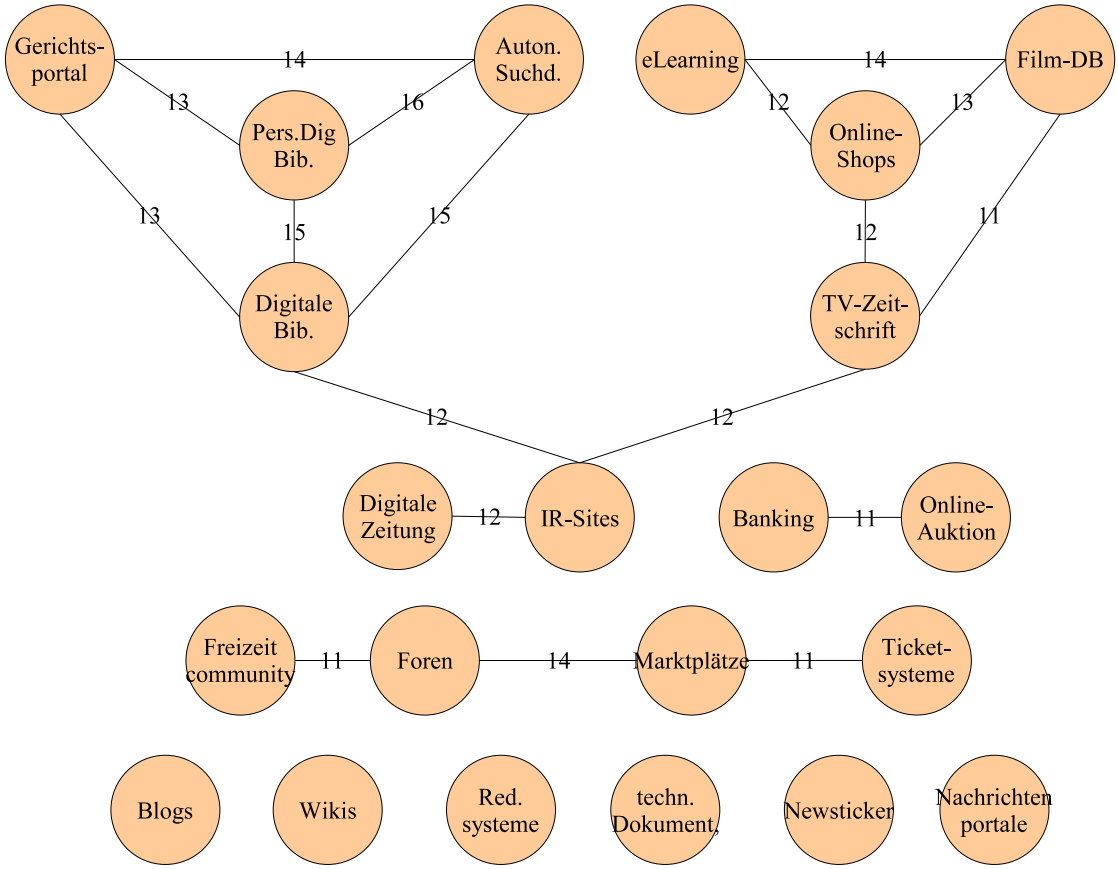

Abbildung 3.4: Ähnlichkeitswerte der Szenarien

  - Navigation
  - Interaktivität
  - Mehrbenutzer
  - Rechteverwaltung
  - Personalisierung
  - Dynamik

- Foren, Marktplätze, Freizeitcommunities, Ticketsysteme

  Zu den wichtigsten Funktionsclustern in dieser Klasse von Szenarien gehören:

  - Navigation
  - Interaktivität
  - Trennung Präsentation
  - Autorenunterstützung
  - Mehrbenutzer
  - Rechteverwaltung

  - Suchfunktion

- eLearning, Filmdatenbanken, Online-Shops

  Zu den wichtigsten Funktionsclustern in dieser Klasse von Szenarien gehören:

  - Navigation
  - Multimediale Bestandteile
  - Interaktivität
  - Suchfunktion

- Digitale Zeitung, Investor-Relations Sites, TV-Zeitschriften

  Zu den wichtigsten Funktionsclustern in dieser Klasse von Szenarien gehören:

  - Datenintegration
  - Navigation
  - Multimediale Bestandteile
  - Suchfunktion

Ohne weitere Zusammenfassung mit anderen Szenarien bleiben aus der ursprünglichen Liste folgende Szenarien bestehen:

- Blogs
- Wikis
- Redaktionssysteme für Fachverlage
- Technische Dokumentation
- Newsticker
- Nachrichtenportale

Überraschend an diesen Ergebnissen ist auf den ersten Blick zum einen, dass eLearning, Filmdatenbanken und Online-Shops eine Klasse bilden. Bei näherer Betrachtung sind sich diese drei Szenarien jedoch sehr ähnlich, was die wichtigen Funktionscluster wie Navigation, Multimediateile, Trennung des Layout und Suchmöglichkeiten betrifft. Einige Funktionscluster spielen dagegen in keinem der drei Szenarien eine erwähnenswerte Rolle. Darunter fallen etwa Validierung, Formatumwandlung, Dynamik, Autorenunterstützung, Mehrbenutzer, Versionierung, Rechtemanagement und Workflow.

Zum anderen hätte man erwarten können, dass sich die Szenarien Blogs, Newsticker und Nachrichtenportale weniger stark unterscheiden als sie es im Ergebnis tun. Blogs und Newsticker sind sich dabei mit einem Wert von 8 noch vergleichsweise ähnlich. Die Unterschiede liegen v. a. in der Bedeutung der Datenintegration, Navigation, Interaktivität, Autorenunterstützung und Personalisierung. Die deutliche Abgrenzung zwischen Blogs und Nachrichtenportalen erklärt sich durch zusätzliche Unterschiede bei der Bedeutung von Multimedia-Bestandteilen, Dynamik, Versionierung, Rechtemanagement, Suchmöglichkeiten und Workflow.

3.5 Oberklassen von Szenarien

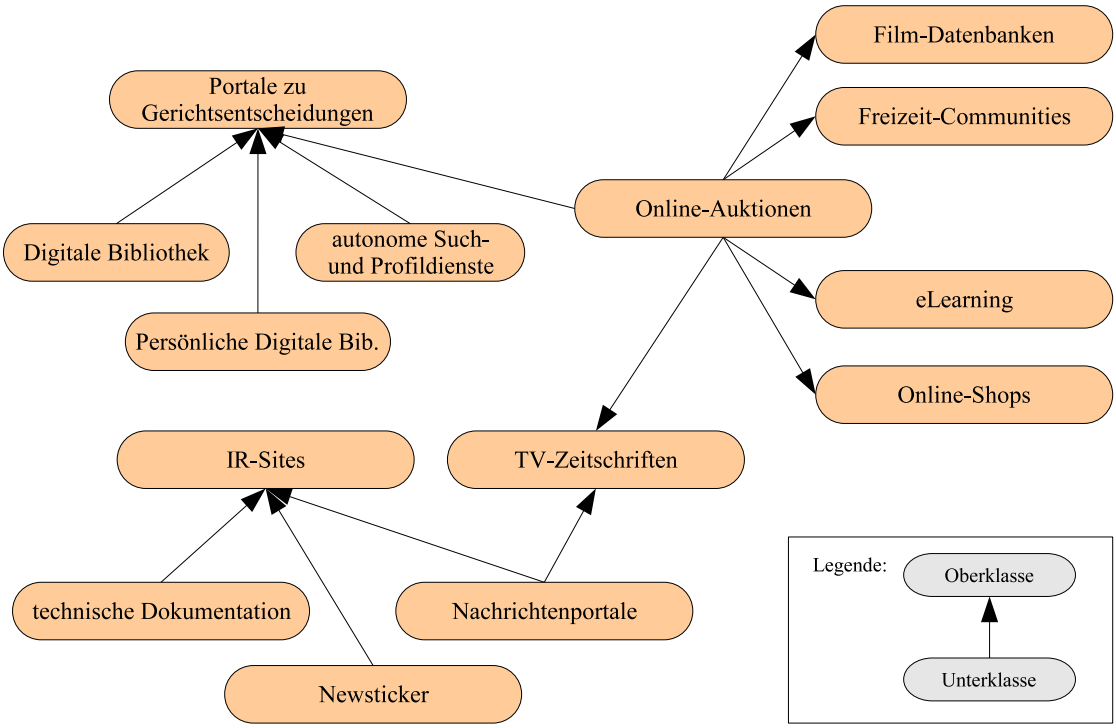

Abbildung 3.5: Oberklassen von Szenarien

Neben dem Vergleich der Szenarien zur Bildung von Klassen mit ähnlichen Signifikanzen der Funktionscluster kann auch eine Bildung von Oberklassen vorgenommen werden. Ein Szenario A ist genau dann eine Oberklasse des Szenarios B, wenn alle Funktionscluster in A mindestens die gleiche Signifikanz wie in B aufweisen. Das Ergebnis dieser Betrachtung ist in der Abbildung 3.5 grafisch dargestellt. Dabei treten die Szenarien „Portale zu Gerichtsentscheidungen“ und „IR-Sites“ besonders hervor, die als Oberklassen für vier bzw. drei andere Szenarien auftreten. Das Szenario der Online-Auktionen wiederum verfügt über sechs Oberklassen.

3.6 Weiterentwicklungsmöglichkeiten

Verbesserung der Suchmöglichkeiten

In diesem Bereich stecken noch sehr viele Verbesserungsmöglichkeiten. Gerade da Sprache nicht eindeutig ist (ein Beispiel: bezeichnet Jaguar nun ein Tier oder einen Automobilhersteller?), kommt es leicht zu irrelevanten Suchergebnissen. Die Qualität der Suche kann durch eine semantische Anreicherung erhöht werden. In diesem Bereich sind Anstrengungen des Semantic Web mit Ansätzen wie Ontologien zu sehen.

Verbesserung der Navigation

Ebenso gibt es noch Verbesserungspotentiale bei der Benutzerunterstützung beispielsweise durch eine Visualisierung in Graphenform. Heutige Browser bieten nur Lesezeichen zum gezielten Anspringen von Dokumenten und eine bidirektionales Backtracking über die besuchten Seiten.

Trotz der anerkannten Bedeutung von globalen Navigationselementen ist es dennoch nicht leicht, sie konsistent und durchgängig auf allen Dokumenten moderner, dezentraler Organisationen zur Verfügung zu stellen. Die meisten sehr großen Unternehmen implementieren sie nur auf ca. 80 % ihrer Seiten [RM02].

Wo immer es möglich ist, sollten die Navigationselemente (globale, lokale und kontextuelle, Sitemaps und Indizes) automatisch generiert werden. Dies spart zum einen menschliche Arbeitszeit und verhindert zum anderen Inkonsistenzen.

Daneben gibt es den Ansatz der sogenannten „sozialen Navigation“ [RM02]. Er basiert auf der Idee, dass für den einzelnen Benutzer wertvolle Schlußfolgerungen aus der Beobachtung des Verhaltens der anderen Benutzer gezogen werden können. Ein Beispiel hierfür ist Amazon’s kollaborativer Filter für Bücherempfehlungen.

Verbesserung der Interaktivität

Eine Verbesserungsmöglichkeit besteht in diesem Bereich darin, die erforderlichen Wartezeiten des Endnutzers aufgrund des bislang üblichen Request–Response-Zyklus von Publikationsanwendungen durch den Einsatz einer asynchronen Kommunikation zwischen Clientprogramm und Server zu reduzieren. Diese Technik wird mit „Ajax“ bezeichnet und findet immer weitere Verbreitung. Mit diesem Ansatz sind auch Möglichkeiten zur Gestaltung der Interaktion gegeben, die sonst nicht praktikabel sind.

Verbesserung der Personalisierung

Die Personalisierung stellt einen großen Forschungsbereich dar. Grundsätzlich besteht der Ablauf der Personalisierung darin, Informationen über den Benutzer zu sammeln, diese zu analysieren und so passende Dokumente für diesen Benutzer zu identifizieren und anzubieten[Buc05]. In jeder dieser Phasen können noch Verbesserungsmöglichkeiten gefunden werden.

Eine Möglichkeit besteht darin, die Personalisierung der Dokumente benutzerseitig erst auf dem System des Nutzers durchzuführen. Dies wird möglich durch die Computerverarbeitbarkeit strukturierter Dokumente. Daher können entsprechende Dokumente auch auf Benutzerseite noch vollständig überarbeitet und umgestellt werden.

4 Funktionalität und Struktur von Publikationssystemen

Software architecture is the set of design decisions which, if made incorrectly, may cause your project to be cancelled. [1]
– EOIN WOODS [RW05]

In diesem Kapitel ist das wesentliche Ziel, die Unterstützung bei der Entwicklung von Publikationssystemen zu verbessern. Zu diesem Zweck wird ein Vorschlag für eine Software-Architektur für Publikationssysteme erarbeitet, die maßgebend von den Funktionsclustern aus Abschnitt 3.1 und somit auch von den Anforderungen an Publikationssysteme aus Abschnitt 2.5 abhängt. Dabei ist nicht das Ziel, eine gänzlich neue Architektur zu erstellen, sondern vielmehr aus dem State-of-the-Art die am besten geeignete Architektur auszuwählen.

Als Erstes wird dazu kurz das Fachgebiet der Software-Architektur vorgestellt. Es wird beschrieben, was Software-Architektur ist, welche Ziele von ihr verfolgt werden, und welche Konzepte existieren, um diese Ziele zu erreichen. Da die Aufteilung eines Systems in Komponenten ein ganz wesentlicher Aspekt ist, wird sich ein Großteil der Abschnitte um dieses Problem drehen.

Anschließend werden die Ziele beschrieben, die mit dem Vorschlag einer Software-Architektur für Publikationssysteme verfolgt werden. Danach wird der Vorschlag präsentiert und es wird erläutert, wie die Maßnahmen zur Zielerreichung im Vorschlag realisiert werden.

Dann werden existierende Software-Systeme darauf untersucht, inwieweit sie die Anforderungen an Publikationssysteme erfüllen und sich somit für den Einsatz als Publikationssysteme oder zumindest als Grundlage für die Weiterentwicklung in Richtung eines Publikationssystems eignen.

Abschließend werden aus den vorangehenden Ergebnissen Richtlinien für die Entwicklung neuer Publikationssysteme abgeleitet.

4.1 Software-Architektur

Software-Architektur spielt eine wesentliche Rolle bei der Festlegung eines guten Designs für Software-Systeme. Auch auf die Frage, was überhaupt eine gute Software-Architektur ist, wird im Folgenden genauer eingegangen werden.

[1] Diese Aussage ist nicht bijektiv zu verstehen, sondern analog zu der Aussage, dass z. B. ein Fußball ein Spielgerät ist. Hier ist nicht notwendigerweise jedes Spielgerät auch ein Fußball. Ebenso gehört nicht alles, was ein Projekt scheitern lassen kann, zu Software-Architektur.

4.1.1 Definition

Für den Begriff der Software-Architektur gibt es eine große Zahl unterschiedlicher Definitionen. Das Software Engineering Institute (SEI) der Carnegie Mellon University[2] hat eine umfangreiche Sammlung von Definitionen zusammengetragen (vgl. [Car05]). Diese Definitionen sind abhängig von ihrer Herkunft in vier verschiedene Kategorien unterteilt: Definitionen aus neueren Arbeiten über Software-Architektur, wichtige und einflussreiche ältere Definitionen, Definitionen aus Artikeln in der Bibliografie des SEI und zu guter Letzt auch noch Definitionen von Besuchern der Internetseite des SEI, die dem Aufruf gefolgt sind und ihre ganz persönliche Definition von Software-Architektur mitgeteilt haben.

Zum Glück gibt es aber auch eine Definition, die dank ihrer Allgemeinheit weitgehend akzeptiert ist und oft zitiert wird (vgl. [Sie04]). Diese Definition soll auch in dieser Arbeit als grundlegende Begriffsfestlegung verwendet werden. Sie stammt aus [BCK03] und lautet:

> The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them.

In Anlehnung an diese Definition wird in [Fow03a] die Architektur eines Systems ebenfalls als Zerlegung des Systems in seine Bestandteile verstanden. Um die Grundaussage der Definition noch etwas zu verdeutlichen, werden im Folgenden zwei weitere Definitionen zitiert, die in dieselbe Richtung gehen:

> Der Software-Entwurf beinhaltet alle Tätigkeiten zur Festlegung der als Software zu realisierenden Komponenten eines Produkts, ihrer Schnittstellen (untereinander und zu den übrigen Systemkomponenten) und Kooperationen. Ergebnis des Entwurfs ist die Software-Architektur. Als solche wird die Beschreibung der Struktur eines Software-Systems auf einer geeigneten Abstraktionsebene in Form von Komponenten (Subsysteme, Komponenten, Pakete, Klassen, . . .) und ihrer Beziehungen zueinander verstanden [Bal01].

> Die Software-Architektur eines Systems ist ein Modell des Systems. Sie macht Aussagen über die Strukturen und die grundlegende Organisation des Systems. Strukturen können Komponenten und deren (Kommunikations-)Beziehungen untereinander und zur Umgebung sein. [Vir06]

So unterschiedlich die einzelnen Definitionen im genauen Wortlaut auch sind, enthalten sie doch eine gemeinsame Aussage: Bei Software-Architektur geht es um die Analyse, Festlegung und Beschreibung der grundlegenden Strukturen von Software-Systemen.

4.1.2 Bedeutung

Eine andere Definition für den Begriff Software-Architektur wird ganz oben auf der bereits erwähnten Internetseite des SEI – hervorgehoben und abgetrennt von den anderen Definitionen

[2] Software Engineering Institute (SEI) Home Page: `http://www.sei.cmu.edu/`

– zitiert und sie dient auch als Eröffnungszitat für dieses Kapitel. Sie stammt von Eoin Woods (Co-Autor von [RW05]) und lautet:

> Software architecture is the set of design decisions which, if made incorrectly, may cause your project to be cancelled.

Auch wenn diese Definition keine formale ist und vielleicht auch mit einem Augenzwinkern geschrieben wurde, so drückt sie doch sehr prägnant aus, welche Bedeutung der Software-Architektur zukommt: In der Systementwicklung werden durch die Software-Architektur sehr früh wesentliche Entscheidungen in Bezug auf das Design eines Systems in Form eines Architekturentwurfs getroffen. Wichtige Eigenschaften des Systems wie Korrektheit, Modifizierbarkeit, Wartbarkeit, Sicherheit, Skalierbarkeit und Performanz sind wesentlich von diesem Entwurf abhängig. Eine einmal realisierte Software-Architektur ist später nur mit hohem Aufwand abänderbar. Die Entscheidung über ihr Design ist somit eine der kritischsten und wichtigsten Punkte im Software-Entwicklungsprozess. Sie kann v. a. bei größeren und komplexeren Systemen sogar wichtiger sein als die Wahl der Datenstrukturen und Algorithmen (vgl. [SG96]). Bei solchen Systemen ist ein Schlüssel zum Erfolg des Softwareentwicklungsprojekts das Festlegen und Befolgen einer einheitlichen Architektur (vgl. [BCH+01]).

4.1.3 Allgemeine Ziele

Mit der Etablierung einer Software-Architektur werden nach [PBG04] neben technischen Zielen auch vier, von der Technik weitgehend unabhängige, allgemeine Ziele verfolgt:

- Unterstützung der Projektplanung.
- Risikominimierung.
- Verbesserung der Kommunikation zwischen beteiligten Personen.
- Festhalten des Wissens über das System.

Desweiteren gibt es Ziele, die diesen Punkten zugeordnet werden können, aber aufgrund ihrer Bedeutung explizit erwähnt werden sollen:

- Erleichterung der Arbeitsteilung bei der Erstellung, Wartung, Weiterentwicklung und Portierung des Systems. Dies ist sowohl eine Folge der Unterstützung der Projektplanung durch die Aufteilung des Systems in möglichst unabhängig voneinander bearbeitbaren Teilproblemen als auch der Verbesserung der Kommunikation zwischen beteiligten Personen.
- Verbesserte Erfassbarkeit des Aufbaus des Systems. Das zum Betrieb und zur Weiterentwicklung notwendige Wissen über die Architektur eines Systems muss in Form von verständlichen Papieren und verfügbaren Mitarbeitern vorgehalten werden. Bei großen Projekten kann niemand das vollständige System im Detail kennen. Umso wichtiger ist es, dass neue Mitarbeiter und andere betroffene Personen sich möglichst einfach einen Überblick über die Architektur verschaffen können.

4.1.4 Strukturen und Sichten

Jedes Software-System ist eine Realisierung seiner Architektur. Software-Systeme können aus unterschiedlichen Perspektiven betrachtet werden: Anwender verwenden das System für die tägliche Arbeit, der Systembetreiber garantiert die tägliche Verfügbarkeit und der Programmierer wartet die Software und entwickelt sie weiter.

Wie bereits in der Definition von Software-Architektur im Abschnitt 4.1.1 angedeutet, besitzt ein Software-System verschiedene Strukturen, die zusammen die Software-Architektur des Systems bilden. Eine Struktur ist dabei eine Menge von Elementen, wie sie in Software- und auch Hardware-Systemen vorkommen (vgl. [BCK03]). Strukturen ermöglichen es, das Prinzip der Separation-of-Concerns auch für die Erstellung einer Software-Architektur einzusetzen.

Beispiele für verschiedene Strukturen sind:

- der Aufbau eines Software-Systems aus verschiedenen Quellcode-Modulen und deren Beziehungen und Abhängigkeiten untereinander.
- der Aufbau eines Software-Systems aus Komponenten und deren Zusammenwirken zur Laufzeit.
- die Verteilung eines Software-Systems auf unterschiedliche Rechner (ein verteiltes System).

Eine Sicht ist eine Repräsentation einer zusammenhängenden Menge von architekturellen Elementen. Sie stellt eine ausgewählte Struktur eines Software-Systems anschaulich dar. Mit Hilfe unterschiedlicher Sichten können die unterschiedlichen Strukturen einer Architektur sichtbar gemacht werden. Für die Auswahl der Sichten ist maßgeblich, welche Bedürfnisse beim Betrachter vorhanden sind. So sind für die im ersten Absatz erwähnten Gruppen von Anwendern, Systembetreibern und Programmierern jeweils unterschiedliche Strukturen und somit auch unterschiedliche Sichten interessant (vgl. [Kru95]).

Für bestehende Systeme kann die Betrachtung aus unterschiedlichen Perspektiven wichtige Erkenntnisse über die Qualität ihrer Architektur liefern. Andersherum müssen schon während des Entwurfs der Software-Architektur diese Perspektiven des späteren Systems im Blick gehalten werden, um eine gute Architektur zu erreichen.

Aus diesem Grund wird im folgenden Abschnitt ein ausgewählter Ansatz kurz vorgestellt, der beschreibt, welche Sichten auf die Strukturen von Software-Architekturen sinnvoll erscheinen und der für die Entwicklung einer Software-Architektur für Publikationssysteme in dieser Arbeit eingesetzt werden soll. Eine Begründung hierfür wird im Abschnitt 4.1.4.2 gegeben.

4.1.4.1 Das 4+1-Sichten Modell

Dieser Ansatz wird in [Kru95] in Form eines Modells für die Beschreibung der Architektur eines Software-Systems vorgeschlagen, das auf der Verwendung von vier verschiedenen Sichten

basiert, die von einer fünften Sicht (der Szenarien-Sicht) vervollständigt werden[3].

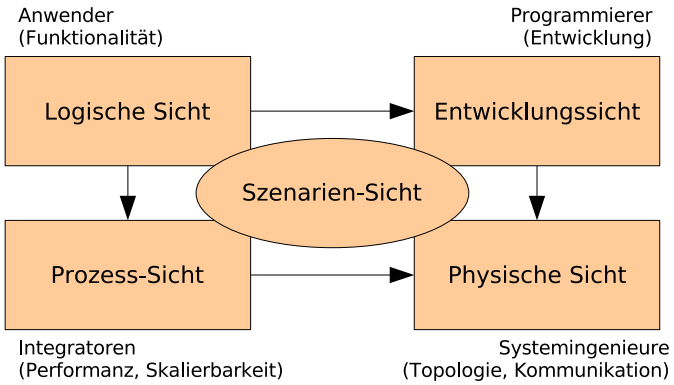

Abbildung 4.1: 4+1-Sichten Modell nach Kruchten

Logische Sicht:
Das System ist aufgeteilt in Schlüsselabstraktionen (die hauptsächlich aus der Problemdomäne stammen) in Form von Objekten oder Objektklassen. Die Logische Sicht unterstützt primär die Erfüllung der funktionalen Anforderungen und dient als Ausgangspunkt für die Modellierung in der Prozess-Sicht und der Entwicklungssicht.

Prozess-Sicht:
betrachtet nicht-funktionale Anforderungen und Aspekte wie Nebenläufigkeit und Verteilung, Fehlertoleranz und betrachtet, wie die Abstraktionen der logischen Sicht in die Prozess-Architektur passen.

Entwicklungssicht:
fokussiert auf die tatsächliche Organisation der Software-Komponenten in der Softwareentwicklungsumgebung. Die Software ist aufgeteilt in kleine Einheiten – Komponenten oder Subsysteme – die von einer kleinen Anzahl von Entwicklern erstellt werden können. Die Subsysteme sind organisiert in einer Hierarchie von Schichten, wobei jede Schicht den oberen Schichten eine schmale und wohldefinierte Schnittstelle anbietet. Diese Sicht wird in Form von Komponenten- und Subsystem-Diagrammen dargestellt, welche die Export- und Import-Beziehungen sichtbar machen.

Physische Sicht:
Beschreibt die Verteilung der Software auf die Hardware. Diese Sicht betrachtet hauptsächlich die nicht-funktionalen Anforderungen an das System wie Verfügbarkeit, Zuverlässigkeit, Performanz und Skalierbarkeit. Die Software wird auf einem Netz von Rechnern oder Verarbeitungsknoten ausgeführt. Die verschiedenen Bestandteile des Software-Systems (Prozesse, Objekte) müssen auf die verschiedenen Verarbeitungsknoten verteilt werden.

[3] In [BRJ99, Seite 31] wird ein gleichartiges Modell beschrieben, bei dem lediglich zwei der Sichten andere Bezeichnungen haben.

Szenarien-Sicht:

Die Zusammenarbeit der Bestandteile der vier oben beschriebenen Sichten wird durch eine kleine Menge von wichtigen Szenarien gezeigt, die allgemeinere Use Cases sind. Diese Zusammenarbeit darzustellen ist die Aufgabe der Szenarien-Sicht. Sie zeigt gewissermaßen die wichtigen Anforderungen. Diese Sicht ist zwar redundant zu den vier anderen Sichten, dient aber zwei Hauptzwecken:

1. als Hilfsmittel um die wichtigsten Bestandteile der Architektur zu identifizieren.
2. zur Überprüfung und Illustration nachdem die Architektur fertiggestellt wurde.

Diese fünf Sichten lassen sich mit Hilfe der folgenden UML Diagramme abdecken:

- Use Cases für die Szenarien-Sicht.
- Use Cases und Klassendiagramme für die logische Sicht.
- Komponentendiagramme für die Entwicklungssicht.
- Deployment Diagramme für die Prozess- und Physische Sicht.

Nicht jede Software-Architektur benötigt alle 4+1 Sichten. Unnötige können weggelassen werden. So ist z. B. die Physische Sicht nicht erforderlich, wenn es nur einen Rechner oder Verarbeitungsknoten gibt und die Prozess-Sicht ist nicht erforderlich, wenn es nur einen Prozess gibt.

4.1.4.2 Fazit

Neben dem hier vorgestellten 4+1-Sichten Modell gibt es noch weitere Ansätze, um ein Software-System in Strukturen aufzuteilen. An dieser Stellen sollen zwei dieser Ansätze kurz erwähnt werden.

Der erste Ansatz stammt aus [BCK03] und teilt die Strukturen eines Software-Systems zunächst in folgende drei Obergruppen auf, zu denen jeweils eine Reihe von Strukturen gehören:

- Komponenten-Strukturen
- Komponenten-und-Konnektoren-Strukturen
- Allokationsstrukturen

Der zweite Ansatz stammt aus [Sie04, BS00] und spielt eine zentrale Rolle im Rahmen des Konzepts von Quasar[4] von sd&m. Hier werden die Strukturen zunächst in zwei Hauptgruppen unterteilt: in die Außenstruktur und die Innenstruktur des Systems. Die Außenstruktur ist dabei vom Anwender wahrzunehmen, während sich für die Innenstruktur nur Software-Architekten, Programmierer und Systembetreiber interessieren.

Die vorgestellten Ansätze liefern Vorschläge, um eine Software-Architektur umfassend mit verschiedenen Sichten zu modellieren. Ich habe mich aus verschiedenen Gründen für den Einsatz

[4] Quasar = „Qualitätssoftwarearchitektur“

des 4+1-Sichten Modells in dieser Arbeit entschieden. Zum einen ist das Modell der Standard im Rational Unified Process (vgl. [Kru99]) und somit State-of-the-Art. Zum anderen ist es gut geeignet für ein inkrementelles Vorgehen, da die unterschiedlichen Sichten direkt mit den verschiedenen Phasen der Software-Entwicklung korrespondieren.

Für diese Arbeit sind nicht alle Sichten des 4+1-Sichten Modells nötig. Auf jeden Fall erforderlich sind die Szenarien-Sicht, die Logische Sicht und die Entwicklungssicht. In der Szenarien-Sicht werden die Use Cases des konkreten Szenarios beschrieben, in dem das Publikationssystem eingesetzt werden soll. Die Logische Sicht beschreibt das Domänenmodell und die Entwicklungssicht umfasst die Komponentenaufteilung des Publikationssystems. Die Prozess-Sicht und die Physische Sicht werden erst interessant, wenn es um die tatsächliche Implementierung und Verteilung des Systems auf verschiedene Rechner geht.

4.2 Qualitätsziele und Zielerreichung durch Software-Architektur

In den folgenden Abschnitten soll zunächst erläutert werden, welche Ziele in Bezug auf die Qualität eines Software-Systems generell existieren. Dies liefert die Grundlage für die Auswahl und Begründung der für Publikationssysteme relevanten Ziele. Anschließend werden Lösungsmöglichkeiten vorgestellt, die im Bereich der Software-Architektur vorhanden sind, um die gewünschten Ziele zu erreichen. Beide Betrachtungen liefern somit somit die Basis für den Vorschlag einer Software-Architektur für Publikationssysteme im Abschnitt 4.3.

4.2.1 Qualitätsziele

Die Funktionalität eines Software-Systems ist die Menge der ausführbaren Funktionen. Vollständige Funktionalität bedeutet die Fähigkeit eines Software-Systems, die Aufgabe zu erfüllen, für die es entworfen wurde (vgl. [BCK03]). Dabei muss das Verhältnis zwischen Funktionsumfang und konzeptueller Komplexität der Architektur beachtet werden (vgl. [Bro75]). Bei gleichem Funktionsumfang ist die Architektur besser, die eine geringere Komplexität aufweist.

Wenn vollständige Funktionalität die einzige Anforderung wäre, könnte das Software-System ein einfacher monolithischer Block ohne jegliche interne Struktur sein. Stattdessen ist es aufgeteilt in eine Reihe von Komponenten, um es zum einen verstehbar zu machen und zum anderen verschiedene Qualitätsziele zu erreichen.

Diese weiteren Qualitätsziele gehören zu den nicht-funktionalen Anforderungen. In der Literatur gibt es einen umfangreichen Fundus an solchen Qualitätszielen (vgl. [BCK03, BS00, SG96, Jes90, Kru95] und IEEE 1061).

Man unterscheidet zwischen der Qualität eines Software-Systems und der Qualität seiner Architektur. Die Qualität eines Software-Systems ist maßgeblich, aber nicht ausschließlich, von der Qualität seiner Architektur bestimmt. Die mangelhafte Realisierung einer Komponente kann

z. B. Performanz und Zuverlässigkeit eines hervorragend strukturierten Software-Systems zunichte machen.

Das wichtigste Ziel bei der Entwicklung eines Software-Systems ist trotz allem, dass es „funktioniert“, also das tut, was es soll. Wenn das – aus welchen Gründen auch immer – nicht der Fall ist, sind alle Betrachtungen zur Erfüllung der Qualitätsziele irrelevant. So spielt die Effizienz eines Algorithmus keine Rolle, wenn er nicht korrekt arbeitet.

Die Erreichung der Qualitätsziele durch Software-Systeme kann man anhand von Kriterien feststellen, von denen einige zur Laufzeit objektiv messbar sind (die harten Kriterien), andere sich jedoch erst im Lauf der Zeit offenbaren und einer quantitativen Bewertung nicht zugänglich sind (die weichen Kriterien).

Die harten Kriterien sind:

Performanz:
Das sind die Antwortzeiten und der Durchsatz des Software-Systems auf einem gegebenen Rechner.

Sicherheit:
Ein gutes Software-System verhindert Missbrauch auf jeder Ebene, beispielsweise durch ein Berechtigungssystem an der Benutzerschnittstelle.

Verfügbarkeit und Zuverlässigkeit:
Verfügbarkeit ist der Quotient aus tatsächlicher Betriebszeit und planmäßiger Betriebszeit. Die Zuverlässigkeit ist die erwartete Zeit bis zum nächsten Ausfall des Software-Systems.

Robustheit:
Falsche Eingaben von Benutzern und korrupte Daten von Nachbarsystemen werden weich abgefangen. Ein robustes Software-System erkennt, dass etwas nicht stimmt und ist darauf ausgelegt.

Benutzbarkeit:
Das Software-System muss vom Benutzer effektiv und effizient gesteuert werden können.

Jedes dieser Kriterien ist ein mittelbares Maß für die Qualität der Software-Architektur. Die Software-Architektur ist dann gut, wenn es möglich ist, jedes Merkmal wie Performanz, Sicherheit oder Benutzbarkeit in jedem sinnvollen Grad herbeizuführen und das mit vernünftigem Aufwand.

Flexibilität ist ein unmittelbares Maß für die Qualität der Architektur. Das Software-System wird im Lauf der Zeit an neue Anforderungen und Randbedingungen angepasst. Hierbei sind sechs weiche Kriterien zu unterscheiden:

Testbarkeit und Integrierbarkeit:
Software-Systeme sind aus Komponenten aufgebaut. Jede Komponente wird einzeln für sich entwickelt und getestet. Anschließend werden die Komponenten zu größeren Sub-Systemen integriert, deren Gesamtheit das Software-System ergeben.

Wartbarkeit:
Ein Software-System ist wartbar, wenn man Fehler mit vernünftigem Aufwand lokal reparieren kann, ohne neue Fehler an anderen, unerwarteten Stellen zu verursachen.

Modifizierbarkeit:
Ein Software-System ist modifizierbar, wenn Veränderungen (Ergänzungen, Verbesserungen, Anpassungen) mit möglichst geringem Aufwand und wenigen Seiteneffekten machbar sind. Die Grenze zwischen Wartung und Änderung ist dabei nicht immer offensichtlich. Zur Modifizierbarkeit gehört zum einen die Erweiterbarkeit, um ein System um zusätzliche Funktionalität erweitern zu können, und zum anderen die Anpassbarkeit, um veränderte Anforderungen und Rahmenbedingungen erfüllen zu können.

Portierbarkeit:
Verkraftet ein Software-System z. B. den Wechsel des Betriebssystems oder des DBMS? Man muss sich Abhängigkeiten von der Systemsoftware genau klarmachen und sie durch geeignete Zwischenschichten reduzieren.

Skalierbarkeit:
Jedes Software-System wird mit bestimmten Erwartungen an z. B. Benutzerzahlen, Transaktionen und Datenmengen geplant. Häufig wachsen diese Zahlen im Laufe der Zeit stärker als ursprünglich angenommen. Das Software-System ist skalierbar, so weit es seine Funktionalität trotzdem weiter erbringt.

Wiederverwendbarkeit:
Projektübergreifende Wiederverwendbarkeit von in einem Projekt entwickelten Komponenten wird nur selten erreicht. Der Grund dafür ist, dass ein Projekt normalerweise nicht darauf abzielt, auch in anderen Projekten wiederverwendbare Komponenten zu entwickeln. Wiederverwendbare Komponenten haben jedoch Produktcharakter und die sogenannte Dreier-Regel besagt: Eine Komponente ist projektübergreifend wiederverwendbar nach dem dritten Einsatz und der dritten Re-Implementierung. Das erste Ziel ist daher die Wiederverwendung innerhalb eines Projekts.

4.2.2 Zielerreichung

Eine gute Software-Architektur hat zwei maßgebliche Eigenschaften (vgl. [GI 06]):

1. Sie führt zu einem Software-System, das – adäquate Programmierung vorausgesetzt – die gerade genannten Qualitätsmerkmale besitzen wird.
2. Der Aufwand, der während des Lebenszyklus des Software-Systems anfällt, ist nicht höher, sondern eher niedriger als der für Systeme mit weniger guter Architektur und gleicher Funktionalität.

Software-Architektur selbst kann keine Qualitätsziele erreichen. Aber sie liefert das entscheidende Fundament für die Erreichung der Ziele (vgl. [BCK03]). Dabei muss auf die Details geachtet werden. Innerhalb komplexer Systeme können Qualitätsziele nie für sich alleine genommen erreicht werden. Beispielsweise stehen Sicherheit und Zuverlässigkeit oft in einem gegenseitigen

Spannungsverhältnis. Und auch Performanz und Flexibilität sind konträr zueinander, denn um eine optimale Performanz zu erreichen, müssen die Spezifika der Systemumgebung ausgenutzt werden, was sich negativ auf die Flexibilität auswirkt. Vgl. dazu Fisher's Fundamental Theorem in [Wei72]:

> The better a system is adapted to a particular environment, the less adaptable it is to new environments.

Sowohl die funktionalen als auch die nicht-funktionalen Anforderungen, die bei der Entwicklung eines Software-Systems erreicht werden sollen, beeinflussen die Entscheidungen bei der Erstellung der Software-Architektur stark. Im Unterschied zu den funktionalen Anforderungen müssen viele Systeme nicht-funktionale Anforderungen nur aus bestimmten Kategorien erfüllen. So spielt etwa die Skalierbarkeit bei vielen Systemen keine Rolle.

Es gibt kein Patentrezept für eine gute Software-Architektur. Bass et al. charakterisieren in [BCK03] eine gute Software-Architektur dahingehend, dass diese es einem System ermöglichen muss, seine Verhaltens-, Qualitäts- und Lebenszyklusanforderungen zu erfüllen. Es gibt auch keine generell richtige Architektur (vgl. [BCK03, BS00]). Eine Software-Architektur ist für eine spezielle Problemstellung mehr oder weniger geeignet und kann im Kontext konkreter Ziele bewertet werden.

Es gibt jedoch eine wichtige allgemeine Voraussetzung für eine gute Software-Architektur. Dies ist die einer konzeptuellen Konsistenz, d. h. dass ähnliche Probleme auf ähnliche Art und Weise gelöst werden (vgl. [BCK03, Bro75]). Sie stellt die durchgängige Vision dar, die das Design des Systems auf allen Ebenen durchzieht. Konzeptuelle Konsistenz verbessert die Erfassbarkeit und Modifizierbarkeit, reduziert die Entwicklungszeit und erhöht die Zuverlässigkeit.

Darüber hinaus existieren allgemeine Aufgaben und Ziele, die jede Architektur erfüllen muss. Eine gut entworfene Software-Architektur setzt sich aus wohldefinierten Abstraktionsschichten zusammen. Die wohl wichtigste Leitlinie lautet: Trennung der Zuständigkeiten. Schon im Jahr 1972 wurden in [Par72] die Bedeutung und die Vorteile der modularisierten Entwicklung von Software-Systemen betont.

4.2.3 Aufteilung in Komponenten

Eine gute Software-Architektur zu finden heißt, sich mit einigen wesentlichen Fragen zu beschäftigen, nach deren Antworten gesucht wird, seitdem es Software gibt:

1. Wie zerlegt man ein System in Komponenten?
2. Wie erreicht man eine sinnvolle Trennung der Zuständigkeiten, z. B. eine Trennung von fachlichen und technischen Komponenten oder von stabilen und sich häufig ändernden Systemteilen?
3. Wie vermeidet man unnötige Abhängigkeiten der Komponenten voneinander?

Eine Komponente erbringt eine bestimmte, abgegrenzte Dienstleistung, und sie ist eine zweckmäßige Entwicklungseinheit der Softwareentwicklung.

Jede Komponente macht minimale Annahmen über ihre Aufrufumgebung. Die Idee ist immer, dass jede Komponente mehrfach zum Einsatz kommt oder zumindest kommen könnte. Die Nutzungshäufigkeit ist dabei oft umgekehrt proportional zur Größe der Komponente. Die Schwierigkeit liegt hierbei darin, die Komponenten nicht zu stark zu reduzieren, da sonst die Kontext-Abhängigkeiten zu anderen Komponenten sehr stark anwachsen: „Maximizing reuse minimizes use“ (vgl. [Gr"05]).

Jedes System befasst sich mit der fachlichen Anwendung und mit technischen Schnittstellen (Betriebssystem, Datenbank, Middleware). Daher gehört jede Software-Komponente zu genau einer von vier Kategorien. Sie kann sein:

- unabhängig von Anwendung und Technik (0-Software)
- bestimmt durch die Anwendung, unabhängig von der Technik (A-Software)
- unabhängig von der Anwendung, bestimmt durch die Technik (T-Software)
- bestimmt durch Anwendung und Technik (AT-Software)

0-Software ist ideal wiederverwendbar, für sich alleine aber ohne Nutzen. Sie implementiert immer ein abstraktes Konzept. Ein Beispiel hierfür ist die STL von C++.

A-Software kann immer dann verwendet werden, wenn vorhandene Anwendungslogik ganz oder teilweise benötigt wird.

T-Software kann immer dann wiederverwendet werden, wenn ein neues System dieselbe technische Komponente einsetzt (JDBC, ODBC, . . .). Komponenten, die Eigenheiten der darunter liegenden Infrastruktur kapseln, gehören zu dieser Gruppe.

AT-Software befasst sich mit Technik und Anwendung zugleich. Sie ist schwer zu warten, widersetzt sich Änderungen, kann kaum wiederverwendet werden und ist daher zu vermeiden.

Der Anteil an AT-Software ist ein wichtiges Maß für die Qualität einer Software-Architektur. Sauber getrennte Zuständigkeiten sind die Voraussetzung für eine gute Implementierung und klaren, übersichtlichen und kompakten Code: Mehrere kleine Probleme unabhängig voneinander zu lösen ist immer einfacher als ein großes Problem direkt anzugehen.

Es geht also um die geschickte Aufteilung eines Systems in technische und fachliche Komponenten, denn das ist vielleicht die wesentlichste Stellschraube für die Qualität der Architektur (vgl. [BS00, BBRS03]). Wichtige Gütekriterien für die Aufteilung in Komponenten sind dabei (vgl. [Jes90]):

- Zerlegung in mittlere Anzahl (3-10) von Komponenten (rekursiv angewandtes Prinzip).
- Komponenten mit Funktionalität nach folgenden Anforderungen:
  - Konsistenz (wenige, allgemein angewandte Eigenschaften)
  - Orthogonalität (freie Kombinierbarkeit der Eigenschaften)

  - Sauberkeit (möglichst keine Sonderfunktionen).
  - Gute Erlernbarkeit
  - Evtl. Gestaltbarkeit
  - Testbarkeit
  - Verbergen der Eigenschaften, die nicht zur bestimmungsgemäßen Funktion gehören.
- wohldefinierte „schmale“ Schnittstellen.
- unabhängige Entwerfbarkeit (Testbarkeit, Fertigbarkeit, Wartbarkeit, Modifizierbarkeit).
- sichere Integrierbarkeit des Systems aus seinen Komponenten.
- Typenarmut / Wiederverwendbarkeit

Bei der Frage, wie man eine gute Komponentenaufteilung findet, sind auch sogenannte Architekturmuster vorteilhaft einsetzbar. Auf Architekturmuster wird in 4.2.5 noch genauer eingegangen werden.

Ein einfaches Kriterium für eine gute Komponentenbildung ist die funktionale Disjunktheit: Keine Funktion ist in mehr als einer Komponente realisiert. Die Wiederverwendung innerhalb eines Systems ist genau dann optimal, wenn die funktionale Redundanz gleich Null ist. In der Praxis wird man dieses Ziel zwar wegen Schnittstellen-Overheads nicht erreichen, dennoch ist es erstrebenswert, die funktionale Redundanz möglichst weit zu reduzieren.

Ein weiteres Kriterium, das zur Aufteilung in Komponenten benutzt werden kann, unterscheidet Komponenten danach, ob sie primär Daten produzieren oder Daten konsumieren. Komponenten, die Daten hauptsächlich produzieren sollten getrennt sein von Komponenten, die Daten hauptsächlich konsumieren. Diese Art der Aufteilung bewirkt eine verbesserte Modifizierbarkeit des Software-Systems, da sich Änderungen an einem Software-System häufig auf die Datenproduktions- oder die Datenverbrauchsseite beschränken (vgl. [BCK03]). In Fällen, in denen beide Seiten von Änderungen betroffen sind, erleichtert diese Aufteilung die inkrementelle Anpassung der Komponenten.

4.2.4 Strategien

Strategien[5] sind fundamentale Designentscheidungen (vgl. [BCK03]) in Bezug auf die Software-Architektur. Sie stellen ähnlich wie die Muster aus [GHJV95] einen Satz von Regeln dar. Durch ihre Einhaltung wird die Erfüllung bestimmter nicht-funktionaler Anforderungen erreicht. In manchen Fällen hängen Strategien auch voneinander ab. Beispielsweise verlangen Strategien, die Redundanz einführen, auch Strategien zur Synchronisation der redundanten Komponenten.

Eine einzelne Strategie oder mehrere logisch zusammenhängende Strategien können durch Architekturmuster zu einer größeren Einheit zusammengefasst werden. Ein Architekturmuster, das Verfügbarkeit unterstützt, verwendet z. B. mit großer Wahrscheinlichkeit sowohl Redundanz als

[5] Das englische Wort „tactics“ aus [BCK03] wurde hier bewusst mit „Strategien“ übersetzt, da es meiner Meinung nach die Bedeutung besser trifft, als „Taktiken“.

auch Synchronisation. Außerdem wird es diese beiden Strategien sehr wahrscheinlich in einer konkreteren Ausprägung enthalten.

Wie im Abschnitt 4.3.1 weiter unten begründet wird, geht es in dieser Arbeit vor allem um die Erreichung des Qualitätsziels der Modifizierbarkeit. Im Folgenden werden Strategien beschrieben, die die Erreichung dieses Ziels unterstützen. Die Strategien sind dabei in drei Gruppen eingeteilt (vgl. [BCK03]):

1. Änderungen lokal halten
2. Verhinderung von Seiteneffekten
3. Bindungszeitpunkt verzögern

Änderungen lokal halten:

Auch wenn es nicht zwangsläufig immer einen Zusammenhang gibt zwischen der Anzahl der Komponenten, die von einer Änderung betroffen sind und den Kosten, diese Änderungen durchzuführen, ist es doch im Allgemeinen so, dass eine Begrenzung auf eine kleine Menge von Komponenten die Kosten reduziert. Das Ziel der Strategien in dieser Gruppe ist es daher, Verantwortlichkeiten während des Designs so auf Komponenten zu verteilen, dass erwartete Änderungen nur geringe Auswirkungen auf andere Komponenten haben.

Semantische Kohärenz bewahren:

Diese Strategie bezieht sich auf die Beziehungen der Verantwortlichkeiten innerhalb einer Menge von Komponenten. Das Ziel dabei ist es, sicherzustellen, dass alle diese Verantwortlichkeiten ohne übermäßige Abhängigkeiten zu Komponenten außerhalb der Menge erfüllt werden können. Metriken zu Kopplung und Kohäsion versuchen, die semantische Kohärenz zu messen. Sie erfassen jedoch nicht die Auswirkungen von Veränderungen.

Eine Unterstrategie ist, gemeinsame Dienste zu abstrahieren. Das Anbieten gemeinsamer Dienste durch spezialisierte Komponenten wird allgemein als förderlich für die Wiederverwendung angesehen[6]. Zum anderen unterstützen gemeinsame Dienste aber auch die Modifizierbarkeit: Interne Veränderungen in einer solchen Komponenten müssen nur dort vorgenommen werden und nicht in allen Komponenten, die diesen Dienst verwenden. Nach außen sichtbare Veränderungen können zum Teil durch Entwurfsmuster wie das Adapter-Muster (vgl. [GHJV95]) abgefangen werden.

Vorwegnahme erwarteter Änderungen:

Die Betrachtung einer Menge von erwarteten Änderungen bietet die Möglichkeit, eine bestimmte Zuordnung von Verantwortlichkeiten zu beurteilen. Die Frage dabei ist: Für jede Änderung, begrenzt die vorgeschlagene Aufteilung die Menge der Komponenten, die verändert werden müssen? Diese Strategie befasst sich also mit der Minimierung der Auswirkungen von Änderungen. In der Realität ist diese Strategie nur begrenzt einsetzbar, da es nicht möglich ist, alle möglichen Änderungen vorherzusehen. Aus diesem Grund wird sie meist zusammen mit der Strategie der Semantischen Kohärenz verwendet.

[6] Die Auslagerung von querschnittlichen Aufgaben wie Logging oder Fehlerbehandlung in getrennte Komponenten ist das Ziel von Aspect Oriented Programming (AOP)

Komponenten generalisieren:

Generische Komponenten können breiter eingesetzt werden, d. h. in gewisser Weise können sie mit einer größeren Menge von Input umgehen. Der Input kann als Sprache für die Komponente aufgefasst werden. Das kann so einfach sein wie die Umwandlung von Konstanten in Eingabeparameter oder so kompliziert wie die Implementierung der Komponente als Interpreter, der die Eingabeparameter als Programm Je allgemeiner eine Komponente ist, desto eher können Änderungen durch eine Anpassung der Eingabesprache, statt einer Veränderung der Komponente selbst, durchgeführt werden.

Mögliche Optionen beschränken:

Veränderungen, besonders in einer Produktlinie, können sehr weitreichend sein und viele Komponente betreffen. Eine Beschränkung der möglichen Optionen reduziert die Auswirkungen von Änderungen.

Verhinderung von Seiteneffekten:

Ein Seiteneffekt einer Änderung ist die Notwendigkeit, Änderungen an Komponenten vorzunehmen, die nicht direkt betroffen sind. Wenn z. B. an der Komponente A eine bestimmte Änderung durchgeführt wird, dann muss die Komponente B nur aufgrund dieser Änderung ebenfalls angepasst werden. Die Komponente B hängt also in irgendeiner Form von A ab.

Daten kapseln:

Datenkapselung (Information Hiding) ist die Aufteilung der Verantwortlichkeiten einer Entität in kleinere Teile und die Festlegung, welche Daten nach außen sichtbar sind und welche unsichtbar sein sollen. Das Ziel ist, Änderungen innerhalb einer Komponente zu isolieren und sie von der Ausbreitung auf andere Komponente abzuhalten. Datenkapselung ist hierfür das älteste Verfahren.

Bestehende Schnittstellen beibehalten:

Wenn B von einer Schnittstelle von A abhängt, dann erlaubt die Beibehaltung dieser Schnittstelle, dass B nicht verändert werden muss, auch wenn sich A geändert hat[7]. Die Stabilität einer Schnittstelle kann auch erreicht werden durch eine Trennung der Schnittstelle von der Implementierung.

Beschränkung der Kommunikationspfade:

begrenzt die Komponenten, mit denen eine gegebene Komponente Daten teilt. Das bedeutet, die Anzahl von Komponenten zu reduzieren, die Daten verarbeiten, die von der gegebenen Komponente produziert werden und umgekehrt.

Benutze einen Vermittler:

Wenn B irgendeine Form von Abhängigkeit von A hat, die nicht semantisch ist, dann ist es möglich, einen Vermittler zwischen B und A einzubauen, der sich um die Abhängigkeit kümmert. (Entwurfsmuster dazu sind Facade, Bridge, Mediator, Strategy, Proxy und Factory, vgl. [GHJV95]). Das Broker-Muster kann benutzt werden, um

[7] Hier wird angenommen, dass sich trotz der Änderung von A die Wertebereiche und Bedeutungen der Parameter in der Schnittstelle nicht geändert haben

Änderungen in der Identität einer Schnittstelle zu maskieren. Ein Nameserver erlaubt, dass sich die Position von A ändern kann, ohne dass B davon betroffen ist. Ein Ressourcenmanager ist ein Vermittler, der für die Ressourcenallokation zuständig ist. Das Factory-Muster hat die Fähigkeit, Instanzen nach Bedarf zu erzeugen, so dass die Abhängigkeit von B von der Existenz von A durch die Factory erfüllt wird.

Bindungszeitpunkt verzögern:
Die Verzögerung des Bindungszeitpunkts erlaubt es dem Systemadministrator, dem Endnutzer oder dem laufenden Prozess, dynamisch Einstellungen oder Input vorzunehmen, die das Verhalten des Systems beeinflussen, ohne dass dieses neu erstellt werden muss.

Registrierung zur Laufzeit:
ermöglicht das Austauschen von Komponenten zum Preis von höheren Kosten durch zusätzlichen Overhead für die Registrierung der Komponenten.

Konfigurationsdateien:
dienen dazu, Parameter beim Systemstart zu setzen.

Polymorphie:
erlaubt das späte Binden von Methodenaufrufen.

Einhalten definierter Protokolle:
erlaubt das Binden von unabhängigen Prozessen zur Laufzeit.

Die Modifizierbarkeit wird darüber hinaus neben einer sorgfältigen Aufteilung in Komponenten auch durch Maßnahmen wie den Einsatz Tabellen-getriebener Techniken verbessert (vgl. [Bro75]). Die Verwendung eines Meta-Modells, das Publikationsanwendungen modelliert kombiniert sowohl Tabellen-getriebene Techniken als auch die Verzögerung des Bindungszeitpunkts. Dieser Ansatz wird aufgrund seiner Bedeutung im nachfolgenden Kapitel gesondert beschrieben.

4.2.5 Architekturmuster

Ein Architekturmuster[8] fasst eine oder mehrere Strategien (siehe 4.2.4) für den Entwurf einer Software-Architektur zusammen. Ein Architekturmuster hat meist eine starke Auswirkung auf die Struktur von Software-Systemen, da es grundlegende Strukturentscheidungen trifft. In diesem Sinne definiert ein Architekturmuster gewissermaßen eine Familie von Systemen, die in der grundlegenden Struktur übereinstimmen, sich aber in Details unterscheiden.

In den folgenden Abschnitten werden Architekturmuster näher vorgestellt, die für die Software-Architektur von Publikationssystemen interessant sind.

4.2.5.1 Schichtenarchitektur

Wie bereits erwähnt, besteht ein System aus einer Menge von Komponenten. Eine Möglichkeit, diese Menge von Komponenten zu strukturieren, wird vom Architekturmuster der Schichtenar-

[8] In der Literatur manchmal auch „Architekturstil“ genannt, z. B. in [SG96].

chitektur beschrieben (vgl. [SG96, Kru95, BMR+96]). Dabei werden Teilmengen von Komponenten gebildet. Jede Teilmenge entspricht einer Schicht der Architektur.

Innerhalb einer Schicht können die Komponenten beliebig miteinander kommunizieren. Bei der Kommunikation zwischen Komponenten verschiedener Schichten sind gewisse Regeln zu beachten.

Bei der sogenannten strikten Schichtenaufteilung sind die einzelnen Schichten linear geordnet und jede Kommunikation wird immer nur von einer höheren Schicht initiiert und ist immer an die unmittelbar darunter liegende Schicht gerichtet (vgl. das ISO/OSI-Referenzmodell).

Bei der nicht-strikten Schichtenaufteilung ist diese Regel in der Form aufgeweicht, dass höhere Schichten auch direkt auf alle weiteren Schichten zugreifen dürfen, die unterhalb liegen. Und manchmal dürfen auch tiefere Schichten die Kommunikation mit höheren Schichten initiieren, etwa um bestimmte Ereignisse (z. B. von der Hardware-Schicht) mitzuteilen.

Der Leitgedanke der Schichtenarchitektur ist die Trennung der Zuständigkeiten zwischen den einzelnen Schichten. Eine Schichtenarchitektur verbessert verschiedene nicht-funktionale Eigenschaften eines Systems (vgl. [BCH+01]):

- Wiederverwendbarkeit: Sauber abgetrennte Schichten können leichter in anderen Systemen zum Einsatz kommen.
- Skalierbarkeit: Die Verbindung zwischen zwei Schichten kann derart gestaltet werden, dass die diensterbringende Schicht auch auf mehreren Rechnern ausgeführt werden kann (z. B. eine verteilte Datenbank).
- Portierbarkeit: Die Spezifika des Betriebssystems oder der Hardware-Plattform können leichter vor den höheren Schichten verborgen werden. Damit ist es einfacher, die untere Schicht an andere Spezifika anzupassen.
- Wartbarkeit: Durch die Trennung in Zuständigkeiten werden die Auswirkungen von Fehlerbehebungen reduziert.
- Anpassbarkeit: Auch hier hilft die Trennung der Zuständigkeiten. Oft sind Anpassungen nur innerhalb einer Schicht notwendig und die anderen Schichten können unverändert bleiben.

Weitere Vorteile sind (vgl. [Fow03a]):

- man kann eine einzelne Schicht oder Folgen von Schichten als zusammenhängendes Ganzes verstehen, ohne viel über die restlichen Schichten wissen zu müssen.
- man kann einzelne Schichten durch alternative Implementierungen ersetzen.
- man kann Abhängigkeiten zwischen den Schichten minimieren.
- Schichten bieten gute Ausgangspunkte für Standardisierungen.
- wenn eine Schicht einmal implementiert ist, kann sie vielen höheren Schichten Dienste zur Verfügung stellen.

Eine Schichtenarchitektur kann auch Nachteile mit sich bringen:

- Schichten können manches, aber nicht alles, gut kapseln. Daher kommt es manchmal zu kaskadierenden Änderungen. Beispielsweise wenn ein Feld in der Datenbank hinzugefügt wird, das auch in der Benutzerschnittstelle angezeigt werden muss.

Ein möglicher Kritikpunkt an der strikten Schichtenaufteilung ist, dass Performanzprobleme möglich sind (vgl. [Fow03a, BCH+01]). Typischerweise müssen Daten beim Übergang von einer Schicht zur anderen Schicht in eine andere Repräsentation umgewandelt werden. Allerdings verbessert die Spezialisierung der Komponenten die Optimierbarkeit, so dass dieser Effekt unter Umständen sogar überkompensiert werden kann. Im Vergleich zur nicht-strikten Schichtenaufteilung ist es bei der strikten Schichtenaufteilung auch leichter,

- die Schichten strategisch auf unterschiedliche Orte zu verteilen
- den Domain Layer in anderen Anwendungen mit anderer Benutzerschnittstelle zu nutzen
- verschiedene Data Sources und Benutzerschnittstellen zu nutzen

Die schwierigste Aufgabe bei der Einführung einer Schichtenarchitektur ist es, zu entscheiden, welche Schichten das System haben soll und für welche Aufgaben jede Schicht verantwortlich sein soll. Zur Einordnung von Funktionen bzw. Komponenten in Schichten, kann man als Orientierung folgende Erfahrungswerte zu Rate ziehen (vgl. [Jes90]):

- je tiefer, desto allgemeiner verwendbar, desto weniger komplex soll eine Funktion sein.
- je tiefer, desto höher ihre Arbeitsgeschwindigkeit.
- je tiefer, desto robuster und störunanfälliger.

Es gibt auch einige bereits vielfach bewährte und recht weit verbreitete Schichtenaufteilungen, die sich zumindest zur Orientierung anbieten. Hierzu zählen Ansätze wie die 3-Schichten-Aufteilung mit Schichten für Presentation, Domain und Data Source oder das Model-View-Controller Muster. Beide werden in den folgenden Abschnitten kurz näher vorgestellt.

3-Schichten-Architektur Die 3-Schichten-Architektur orientiert sich an [Fow03a] und unterscheidet folgende Schichten:

Presentation:
Diese Schicht kümmert sich um die Interaktion zwischen dem Benutzer und dem System. Sie stellt auch die Schnittstelle dar, die vom System nach außen angeboten wird.

Domain Logic:
Diese Schicht wird manchmal auch als Business Logic Schicht bezeichnet. Hier findet die eigentliche Arbeit statt, die das System in seiner Anwendungsdomäne erbringen muss:

- Berechnungen basierend auf Eingaben und gespeicherten Daten.
- Überprüfung von allen Daten, die von der Presentation Schicht kommen.
- Entscheidung, wie die Data Source Schicht benutzt wird, basierend auf Daten, die von der Presentation Schicht erhalten wurden.

Data Source:
In dieser Schicht geht es um die Kommunikation mit anderen Systemen, die für das eigentliche System Dienste anbieten.

Eine wichtige Regel bei der Schichtenaufteilung ist, dass die Domain und Data Source Schichten nie von der Presentation Schicht abhängen dürfen. Dies ermöglicht den leichten Austausch oder Erweiterung der Benutzerschnittstelle des Systems, z. B. vom Rich-Client hin zum Web-Browser.

Modell-View-Controller Ein wichtiges Muster für die Aufteilung von Komponenten innerhalb einer Schichtenarchitektur wie etwa der 3-Schichten-Architektur ist das Model-View-Controller Muster (MVC, vgl. [BMR+96, Fow03a]). Es legt dabei die Aufgaben von verschiedenen Komponenten fest, die auf verschiedenen Schichten angesiedelt sind.

Das MVC-Muster unterteilt die Benutzerinteraktion in drei verschiedene Rollen:

Model:
ist die Menge von Objekten, die Informationen über die Anwendungsdomäne repräsentieren. Die Objekte sind für den Benutzer nicht sichtbar und enthalten alle Daten und alles Verhalten, das nicht für die Benutzerschnittstelle verwendet wird.

Die Komponenten, die zum Model gehören, sind in den Schichten der Domain Logic und der Data Source angesiedelt.

View:
kümmert sich um die Darstellung des Model in der Benutzerschnittstelle. Hier geht es nur um die Anzeige von Daten. Jede Änderung der Daten wird vom Controller durchgeführt.

Die Komponenten des View sind in der Presentation Schicht angesiedelt.

Controller:
nimmt die Eingabedaten vom Benutzer entgegen, ändert das Model und sorgt dafür, dass der View entsprechend aktualisiert wird.

Die Komponenten des Controller sind wie die des View in der Presentation Schicht angesiedelt und stellen die Verbindung zum Model in den unteren Schichten dar.

Dieser Ansatz bietet vor allem eine saubere Trennung zwischen der Implementierung der Business Logic (im Model) und der Präsentation (im View). Diese Trennung ist nach [Fow03a] eine der fundamentalsten Heuristiken für gutes Software-Design. Dafür gibt es verschiedene Gründe:

- Bei der Präsentation und dem Model geht es um fundamental unterschiedliche Dinge.
- Benutzer möchten dieselben Daten aus dem Model abhängig vom Kontext auf verschiedene Arten angezeigt bekommen können.
- Nicht-sichtbare Objekte sind üblicherweise leichter zu testen als sichtbare Ojekte.

Für das Model-View-Controller Muster gibt es unterschiedliche Möglichkeiten, wie die Aufgaben genau auf die einzelnen Teile (Model, View, Controller) des Musters verteilt werden können.

Hier hat ebenfalls Martin Fowler in [Fow03a] eine gute Auswahl zusammengestellt. In den folgenden Abschnitten werden die für diese Arbeit vorteilhaft erscheinenden Möglichkeiten kurz vorgestellt.

Front Controller Front Controller kümmert sich um die Bearbeitung von Requests, die vom Web-Server geliefert werden. In einer komplexen Webanwendung gibt es viele Dinge, die für alle Requests anfallen, z. B. Beachtung von Sicherheit, Internationalisierung, zur Verfügung Stellung von bestimmten Ausgaben für bestimmte Benutzer. Wenn diese Bearbeitung über viele Objekte verteilt ist, kann es zu großen Redundanzen kommen. Darüber hinaus erschwert es die Änderung des Verhaltens zur Laufzeit.

Ein Front Controller fasst diese Request-Bearbeitung zusammen, indem alle Requests durch ein einzelnes Objekt geleitet werden. Dieses Objekt kann die gemeinsame Bearbeitung koordinieren und diese Bearbeitung kann zur Laufzeit angepasst werden.

Der Front Controller ist üblicherweise in zwei Teile geteilt:

Webhandler:
: empfängt die Requests vom Web-Server. Hier werden gerade genug Daten extrahiert, um zu entscheiden, welcher Befehl ausgeführt werden soll. Diese Entscheidung kann statisch oder dynamisch erfolgen. Bei der dynamischen Auswahl können neue Befehle hinzugefügt werden, ohne dass der Webhandler geändert werden muss.

Command Hierarchy:
: Die Objekte in diesem Bereich kontrollieren die eigentliche Befehlsausführung. Durch die Trennung vom Webhandler benötigen sie kein Wissen über die Web-Umgebung und können somit auch unabhängig von einem Web-Server getestet werden. Außerdem verbessert dies die Wiederverwendbarkeit und Portierbarkeit zwischen Web-Servern.

Eine Möglichkeit, das Verhalten des Front Controller zu erweitern, besteht darin, den Webhandler um eine Filterkette zu ergänzen, die sich um Aufgaben wie Authentifizierung kümmert (vgl. Intercepting Filter Muster in [ACM02]).

Eine weitere Möglichkeit zur Anpassung besteht darin, den Webhandler noch einmal zu unterteilen in einen minimalen Webhandler und einen Dispatcher. Der minimale Webhandler extrahiert die Grunddaten aus dem Request und reicht sie an den Dispatcher derart weiter, dass dieser nichts mehr von der Existenz eines Web-Servers mitbekommt. Dies erleichtert die Testbarkeit des Systems. Durch diese zentrale Ablaufsteuerung ergeben sich einige Vorteile. Zum einen sind neue Abläufe und Funktionalitäten leichter realisierbar, da die vorhandene Funktionalität in der zentralen Komponente entsprechend aufgerufen werden kann. Zum anderen wird die Erweiterbarkeit erleichtert, da neue Funktionalität einfach hinzugefügt und genutzt werden kann.

Transform View Hier werden die Daten, die aus der Domänenschicht kommen, Element für Element in das Ausgabeformat transformiert. Ein anderer Ansatz dazu ist Template View. Hier ist die Ausgabe im wesentlichen schon vorbestimmt und es werden nur noch an bestimmten Stellen formularartig Daten eingesetzt. Im Vergleich dazu ist Transform View deutlich flexibler und auch besser geeignet für strukturierte Dokumente.

Two-Step-View Two-Step-View ist eine Verfeinerung zu Transform View. Hier wird die Ausgabe nicht direkt aus den Domänendaten generiert, sondern es gibt noch einen Zwischenschritt, in dem aus den Domänendaten erst ein Zwischenergebnis generiert wird. Dieses Zwischenergebnis orientiert sich zwar schon an der endgültigen Ausgabe, wird aber erst im zweiten Schritt in das finale Ausgabeformat umgewandelt.

Dadurch ist es einfacher, alle Ausgaben in einer konsistenten Art und Weise zu erzeugen. Außerdem werden Anpassungen erleichtert, da sie nur an einer einzigen Stellen vorgenommen werden müssen.

4.2.5.2 Pipes and Filters

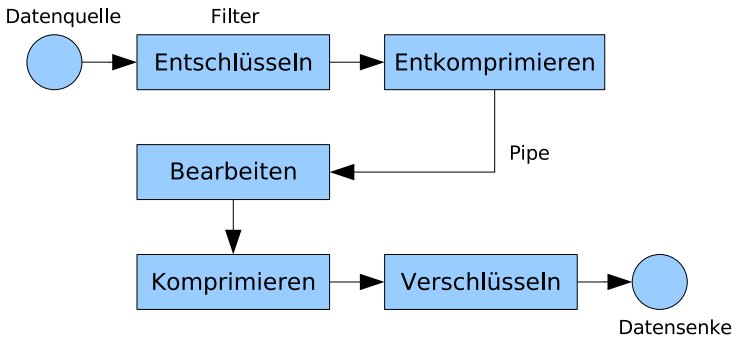

Abbildung 4.2: Architekturmuster Pipes and Filters

Pipes and Filters (vgl. [BMR+96, SG96]) ist ein recht einfaches und knapp beschreibbares Architekturmuster. Es zerlegt die Gesamtaufgabe eines Systems, das einen Strom von Daten verarbeitet, in mehrere sequentielle Bearbeitungsschritte (siehe Beispiel in der Abbildung 4.2). Die einzelnen Bearbeitungsschritte sind durch den Datenfluss innerhalb des Systems miteinander verbunden. Jeder Bearbeitungsschritt ist dabei in eine Komponente gekapselt, die als Filter bezeichnet wird. Daten werden mit Hilfe von sogenannten Pipes zwischen aufeinander folgenden Filter-Komponenten weitergereicht. Dabei sind die Ausgabedaten des einen Filters die Eingabedaten des nächsten. Filter haben eine sehr einfache Schnittstelle. Sie empfangen Daten über die eingehende Pipe, verarbeiten die Daten und schicken die Ergebnisse an die ausgehende Pipe.

Dieses Architekturmuster ist bekannt z. B. aus der UNIX-Welt. Hier existieren viele kleinere Werkzeuge/(Hilfs-)Programme, die einfach kombiniert werden können, um eine spezielle Aufgabe zu lösen.

Die wesentlichen Vorteile von Pipes and Filters liegen in der Verbesserung der Wiederverwendbarkeit, Flexibilität und Modifizierbarkeit eines solchermaßen gestalteten Systems. Da alle Filter-Komponenten die gleiche externe Schnittstelle besitzen, können sie leicht neu kombiniert werden, um eine andere Lösung zu liefern. Dadurch ist es einfach möglich, ganze Familien von verwandten Systemen aufzubauen. Es können neue Filter hinzugefügt werden, existierende

Filter können weggelassen oder zu einer neuen Abfolge kombiniert werden, ohne dass die bestehenden Filter selbst geändert werden müssen. Die Kapselung kleinerer Bearbeitungsschritte in Komponenten erleichtert die Wiederverwendbarkeit, da sie zum einen überschaubarer sind und zum anderen besser in anderen Kontexten einsetzbar sind. Außerdem wird der Prinzip des Information Hiding realisiert, da unterschiedliche Komponenten keine gemeinsamen globalen Daten miteinander teilen.

Diese Vorteile sind für die Software-Architektur eines flexiblen und modifizierbaren Publikationssystems nutzbringend einsetzbar, um z. B. die Komponenten des Publikationssystems je nach den Anforderungen des jeweiligen Szenarios miteinander kombinieren zu können. Dabei muss dieses Architekturmuster nicht zwangsläufig für alle Komponenten eines Systems eingesetzt werden. Es kann auch auf eine Untermenge von Komponenten beschränkt sein. Denn eine wichtige Voraussetzung für dieses Architekturmuster ist, dass eine Aufgabe grundsätzlich in solch getrennte Verarbeitungsschritte aufteilbar sein muss. Dies ist nicht immer der Fall. Im Abschnitt 4.3 wird noch genauer darauf eingegangen, wie dieses Architekturmuster in dieser Arbeit eingesetzt wird.

4.2.5.3 Reflection

Das Reflection Architekturmuster (vgl. [BMR+96]) liefert einen Mechanismus, um die Struktur und das Verhalten von Software-Systemen dynamisch anzupassen. Es unterstützt die Modifikation grundlegender Aspekte wie Datenstrukturen und Mechanismen für den Funktionsaufruf. In diesem Architekturmuster ist ein System in zwei Teile aufgeteilt: Eine Meta-Ebene liefert Informationen über ausgewählte Eigenschaften des Systems und macht damit das System „self-aware“. Eine Basis-Ebene enthält die Anwendungslogik. Veränderungen an den Informationen in der Meta-Ebene beeinflussen das Verhalten der Basis-Ebene.

Ein System zu entwerfen, das von vornherein eine große Bandbreite unterschiedlicher Anforderungen erfüllt, ist eine sehr schwierige Aufgabe. Eine bessere Lösungsmöglichkeit ist es, eine Software-Architektur zu entwerfen, die offen ist für Modifikationen und Veränderungen. Ein System, das auf einer solchen Architektur basiert, kann dann bei Bedarf leichter an die veränderten Anforderungen angepasst werden.

Eine Software „self-aware“ zu machen heißt, ausgewählte Aspekte ihrer Struktur und ihres Verhaltens für Anpassungen zugänglich zu machen. Dies führt zu einer Architektur mit der oben genannten Zweiteilung in Meta-Ebene und Basis-Ebene.

Die Meta-Ebene stellt eine Selbst-Repräsentation der Software zur Verfügung, um ihr ein Wissen über ihre eigene Struktur und ihr Verhalten zu geben. Sie besteht aus sogenannten Meta-Objekten. Meta-Objekte kapseln und repräsentieren Informationen über die Software. Dazu gehören zum Beispiel Datenstrukturen, Algorithmen, oder auch Mechanismen für den Funktionsaufruf.

Die Implementierung der Basis-Ebene benutzt die Meta-Objekte, die unabhängig bleiben von den Aspekten, die sich möglicherweise ändern. Beispielsweise könnten Komponenten auf der Basis-Ebene nur über Mechanismen miteinander kommunizieren, die von Meta-Objekten zur

Verfügung gestellt werden. Durch eine Änderung dieser Meta-Objekte kann der Kommunikationsmechanismus geändert werden, ohne dass die Komponenten auf der Basis-Ebene auch geändert werden werden müssen.

Durch diese Eigenschaften verbessert diese Architektur die Flexibilität und Anpassbarkeit der Software und erscheint ebenfalls sinnvoll für einen Einsatz im Rahmen einer Software-Architektur für Publikationssysteme.

4.3 Eine Software-Architektur für Publikationssysteme

In diesem Abschnitt werden zunächst die Ziele beschrieben, die mit der Entwicklung dieser Software-Architektur erreicht werden sollen und es werden Begründungen gegeben, warum diese Ziele als relevant betrachtet werden. Anschließend wird eine zu diesen Zielen passende Software-Architektur für Publikationssysteme vorgeschlagen.

4.3.1 Auswahl und Begründung der relevanten Ziele

Bei der Auswahl und Begründung der für die Entwicklung der Software-Architektur relevanten Ziele wird wieder zwischen funktionalen und nicht-funktionalen Zielen unterschieden. Dies erfolgt entsprechend der Unterscheidung zwischen funktionalen und nicht-funktionalen Anforderungen bei der Entwicklung von Software-Systemen. Oftmals ergeben sich aus primären Zielen weitere Unterziele, deren Berücksichtigung positive Auswirkungen auf die Erreichung der primären Ziele haben. Solche Unterziele sind ebenfalls in den beiden folgenden Abschnitten beschrieben.

4.3.1.1 Funktionale Ziele

Funktionale Ziele betreffen die Funktionalität, die das Publikationssystem erbringt. Das primäre funktionale Ziel ist die vollständige Funktionalität, d. h. ein Publikationssystem, das auf der vorgeschlagenen Software-Architektur basiert, soll alle funktionalen Anforderungen, die an Publikationssysteme gestellt werden (siehe Abschnitt 2.5), erfüllen. Die Funktionscluster aus Abschnitt 3.1, die ähnliche funktionale Anforderungen zu größeren Einheiten zusammenfassen, können bei der Entwicklung der Software-Architektur entsprechend zur Orientierung berücksichtigt werden.

4.3.1.2 Nicht-funktionale Ziele

Bei den nicht-funktionalen Zielen steht im Vordergrund, eine Software-Architektur zu entwerfen, die flexibel an unterschiedliche Anforderungen angepasst werden kann.

Der Grund hierfür ist das sehr breite Spektrum von Szenarien für den Einsatz von Publikationssystemen und das Ziel, ein inkrementelles Vorgehen zu unterstützen. Obwohl die Liste von 22

Szenarien in Abschnitt 3.2 schon ein sehr breites Gebiet abdeckt, ist sie zum einen sicherlich nicht als vollständig zu betrachten und zum anderen wird es im Laufe der Zeit immer wieder zu geänderten Anforderungen innerhalb einzelner Szenarien kommen. Diese Szenarien sollten besser nicht alle auf einmal unterstützt werden. Stattdessen bietet ein inkrementelles Vorgehen bei der Entwicklung eines Publikationssystems verschiedene wichtige Vorteile: So kann beispielsweise frühzeitig Betriebserfahrung gesammelt werden, die zusammen mit dem Feedback der Benutzer des Systems Korrekturen bei der Entwicklung zu einem frühen Zeitpunkt erlauben.

Das primäre Ziel ist also die möglichst gute Unterstützung der Modifizierbarkeit der Publikationssysteme. Modifizierbare Software-Systeme sind umso wichtiger, je vielfältiger die Umgebungen sind in denen die Systeme arbeiten soll, je ungenauer die Anforderungen an die Systeme sind, je häufiger sich diese verändern und je allgemeiner die durch die Systeme bearbeiteten Aufgaben sind. All diese Punkte treten bei Publikationssystemen auf.

Wichtige Kriterien für die Modifizierbarkeit eines Software-Systems sind zum einen die saubere logische Trennung von funktional unterschiedlichen Bestandteilen in der Software-Architektur und zum anderen die Verwendung standardisierter Techniken in Schnittstellen.

Um diese Ziele zu erreichen, ist zunächst eine vorteilhafte Aufteilung in Komponenten erforderlich. Außerdem soll die Software-Architektur in Schichten aufgeteilt werden, um die Komponenten zu größeren logischen Einheiten zu gruppieren. Durch diese Maßnahmen wird eine inkrementelle Erreichung der funktionalen Ziele erleichtert und sie ermöglicht die Wiederverwendbarkeit von Komponenten und die Modifizierbarkeit der Software-Architektur selbst.

4.3.2 Vorschlag und Zielerreichung

In der Abbildung 4.3 ist übersichtsartig zusammengestellt, welche Konzepte im Architekturvorschlag für welche Zielerreichung eingesetzt werden. Die Konzepte sind bereits im Abschnitt 4.2.2 genau beschrieben worden. In diesem Abschnitt wird daher nur noch erläutert, wie die Konzepte im Architekturvorschlag konkret umgesetzt sind.

Eine wichtige Möglichkeit zur Verbesserung der Modifizierbarkeit und Flexibilität ist der Einsatz von Maßnahmen, die es ermöglichen, Veränderungen des Verhaltens eines Systems zu konfigurieren anstatt diese programmieren zu müssen. Anregungen dazu liefert das Reflection Architekturmuster, das Informationen über Datenstrukturen und Verhalten in eine Meta-Ebene verlagert (siehe Abschnitt 4.2.5.3). In dieser Arbeit wird dieser Ansatz kombiniert mit dem Einsatz tabellen-getriebener Techniken, was die Auswertung dieser Informationen zur Laufzeit durch Interpreter ermöglicht. Um diese Kombination zu ermöglichen, wird ein Meta-Modell entwickelt, das zur Beschreibung von Publikationsanwendungen dient und das es dem Publikationssystem erlaubt, das Verhalten zur Laufzeit zu interpretieren und auszuführen. Dieses Meta-Modell wird im Kapitel 5 getrennt vorgestellt.

In der Abbildung 4.4 ist der Vorschlag für eine Software-Architektur für Publikationssysteme dargestellt. Hier sind die Aufteilung der Funktionalität in Komponenten und die Verteilung der Komponenten auf Schichten erkennbar. Dadurch ist die logische Sicht aus dem 4+1-Sichten Modell aus Abschnitt 4.1.4.1 beschrieben.

	Modifizierbarkeit				
	Erweiterbarkeit	Anpassbarkeit	Wiederverwendbarkeit	Portierbarkeit des Systems	Kommunizierbarkeit
einheitliche Architektur					x
Schichtenarchitektur	x	x	x	x	
Pipes and Filters	x	x	x		
Reflection	x	x			
Interpreter	x	x			
Model-View-Controller	x	x	x		
Transform-View				x	
Two-Step-View	x	x			
Front Controller	x	x	x	x	
Komponentenzerlegung	x	x	x	x	
Information Hiding	x	x	x		

Abbildung 4.3: Ziele und Maßnahmen zur Zielerreichung

Die Wechselwirkung und der Kontrollfluss zwischen den Komponenten ist hier noch nicht modelliert, denn diese sind nicht allgemein festgelegt, sondern ergeben sich erst mit der Modellierung eines konkreten Szenarios. Genauer gesagt: Das Modell eines Szenarios (das zum später vorgestellten Meta-Modell konform ist) dient der Dispatcher Komponente dazu, den Kontrollfluss zwischen den restlichen Komponenten zu steuern.

Das Modell legt aber eine einheitliche Architektur fest. Der Einsatz einheitlicher Architekturkonzepte und die damit verbundene konzeptuelle Konsistenz bei Modellen konkreter Systeme ist ein wichtiger Faktor bei der Verbesserung der Erfassbarkeit und Modifizierbarkeit der Systeme (vgl. hierzu auch Abschnitt 4.2.2).

4.3.2.1 Schichtenarchitektur

Die Software-Architektur ist zunächst analog zu Abschnitt 4.2.5.1 in drei wesentliche Schichten unterteilt:

- Presentation
- Domain
- Data Source

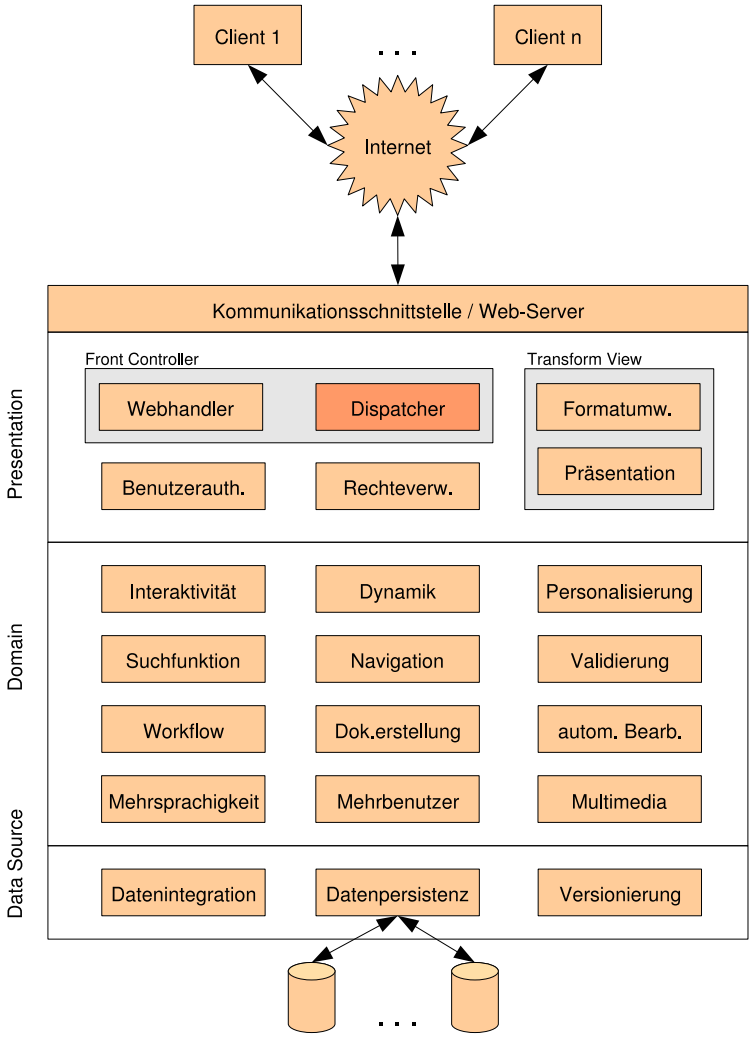

Abbildung 4.4: Software-Architektur für Publikationssysteme

Presentation:

Die Requests an das Publikationssystem können mit einem reichhaltigen Satz von Parametern versehen sein. Möglich sind:

- Request-Metadaten (z. B. aufgerufene URI, verwendete Client-Software, bevorzugte Sprachen).
- Query-String-Parameter, die mit beliebigen Parametern in Form von Attribut-/Wert-Paaren gefüllt sind.
- Beliebige weitere Daten, inkl. XML-Dokumente (z. B. aus XForms Formularen heraus) und Datei-Uploads.

Zunächst muss eine differenzierte Analyse des Requests durchgeführt werden, z. B. zur

Bestimmung welche Ressource angefordert wurde und welche anderen Ressourcen für die Beantwortung des Request benötigt werden. Außerdem wird der Plan bestimmt, wie die Ressourcen zu komponieren sind. Anschließend werden die Ressourcen angefordert und der Kompositionsplan wird ausgeführt.

Realisiert ist dieses Vorgehen durch das Model-View-Controller Muster. Der Controller-Teil ist dabei als leicht abgewandelte Form des Front Controller Musters realisiert. Der Front Controller ist aufgeteilt in einen minimalen Webhandler und einen Dispatcher. Der minimale Webhandler extrahiert alle relevanten Daten aus dem Request vom Web-Server und reicht sie in einer neutralen Form an den Dispatcher weiter. Dieser entscheidet anhand der erhaltenen Daten, welche weiteren Bearbeitungsschritte erforderlich sind und steuert den Kontroll- und Datenfluss zwischen den Komponenten. Er benutzt also die angebotenen Funktionen der Komponenten, um den Request zu beantworten und er stellt damit die Verbindung zwischen dem Controller-Teil und dem Domänen-Teil des Software-Systems her.

Die eigentliche Bearbeitung findet in den unteren Schichten statt, die nachfolgend beschrieben werden. Nach der Bearbeitung besteht die Aufgabe der Presentation Schicht darin, das Ergebnis der Bearbeitung dem Benutzer zu präsentieren. Diese Aufgabe wird vom View-Teil übernommen, der im Architekturvorschlag durch zwei Muster realisiert ist: Zum einen durch Transform View und zum anderen durch Two-Step-View. Der zusätzliche Einsatz des Two-Step-View ist sinnvoll, da die strukturierten Dokumente ausgehend von einem einheitlichen Zwischenformat in verschiedene Ausgabeformate transformiert werden. Diese Transformation ist in der Komponente Formatumwandlung realisiert. Die Komponente Präsentation verknüpft zuvor eine Stilvorlage mit dem Dokument, die für die Transformation benutzt werden soll. Der Ablauf in beiden Komponenten entspricht dem Transform View Muster, da die Ausgabe beliebig aus den Eingabedaten generiert werden kann.

Domain:

In dieser Schicht sind alle Komponenten angesiedelt, die sich direkt mit der Domäne des Elektronischen Publizierens befassen und nicht zu Controller oder View in der Presentation Schicht gehören. Dazu zählen etwa die Komponente zur Validierung strukturierter Dokumente, die Komponente zur Unterstützung automatischer Bearbeitungsschritte und auch die Komponenten zur Multimedia-Unterstützung.

Data Source:

In der untersten Schicht sind die Komponenten angesiedelt, die sich direkt mit den Datenquellen befassen, z. B. die Komponenten zur Integration der verschiedenen Datenquellen und Komponenten zur Unterstützung der Versionierung von Dokumenten.

Als Datenquellen können beispielsweise dienen:

- Dateisysteme
- Repositories / Content-Management-Systeme
- Web-Services (z. B. Google, Amazon, . . .)

Neben dem Model-View-Controller Muster werden noch andere Architekturmuster eingesetzt. Das erste davon ist das Pipes and Filters Architekturmuster. Die Entscheidung für dieses Muster hängt maßgeblich mit dem Wesen der Publikationskette zusammen: Bei beiden ist der grundsätzliche Ablauf sehr ähnlich (nämlich eine Folge von miteinander verbundenen Bearbeitungsstationen), so dass Pipes and Filters hier vorteilhaft eingesetzt werden können, um Dokumente zusammenzustellen. Die Stationen der Publikationskette (Abschnitt 2.3.4) werden von den Dokumenten sequentiell durchlaufen und damit gibt die Publikationskette zu einem hohen Maß die Abarbeitungsreihenfolge eines Request vor. Das Architekturmuster Pipes and Filters kommt nicht deshalb zum Einsatz, um alle Komponenten eines Publikationssystems miteinander zu verbinden, sondern es wird innerhalb einzelner Komponenten eingesetzt, die Bearbeitungsstationen der Publikationskette implementieren. Die Realisierung des Pipes and Filters Musters ist daher in der Abbildung 4.4 auch nicht direkt erkennbar.

Ein weiteres Architekturmuster, das verwendet wird, ist das Reflection Muster. Allerdings kommt es nicht in der Form vor, die in 4.2.5.3 vorgeschlagen wurde, sondern in Kombination mit einem Interpreter-Ansatz: Ein Modell des Einsatz-Szenarios macht das Publikationssystem „self-aware“ und erlaubt dem Interpreter zu entscheiden, welche Komponenten wie zusammenarbeiten müssen. Das Meta-Modell aus Kapitel 5 definiert eine gemeinsame Sprache für die Modellierung der Szenarien.

4.3.2.2 Komponentenzerlegung

Die Zerlegung des Systems in Komponenten soll den in 4.2.3 dargestellten Kriterien genügen und stellt entsprechend die Entwicklungssicht aus dem 4+1-Sichten Modell dar (vgl. Abschnitt 4.1.4.1).

Es bietet sich an, sich bei der Aufteilung der Funktionalität auf Komponenten an den bereits festgestellten Funktionsclustern aus Abschnitt 3.1 zu orientieren. Da dort bereits ähnliche Anforderungen an die Funktionalität zu größeren Einheiten zusammengefasst sind, wird schon eine weitgehende funktionale Disjunktheit erreicht. Die Aufteilung erfolgt also funktionsorientiert (vgl. [BS00, Par72]). Damit werden auch die Abhängigkeiten zwischen den Komponenten reduziert, denn jede Komponenten erbringt eine bestimmte, abgegrenzte Dienstleistung. Diese Orthogonalität ermöglicht die freie Kombinierbarkeit der Komponenten.

Die Trennung der Zuständigkeiten spiegelt sich bereits durch die Einteilung der Komponenten in verschiedene Schichten wider. Komponenten, die zur Gruppe der A-Software (fachliche Komponenten) zählen sind in der Domain Schicht zu finden, Komponenten der T-Software (technische Komponenten) stecken primär in der Data Source Schicht und in Form des Webhandlers in der Presentation Schicht. 0-Software (siehe 4.2.3) kann auf allen Schichten vorkommen.

Die Komponente zur Mehrbenutzerunterstützung ist zweigeteilt. Es gibt zum einen die Benutzerauthentifizierung und zum anderen die eigentliche Mehrbenutzerunterstützung. Die Benutzerauthentifizierung ist auf der Presentation Schicht angesiedelt und dient dazu, entsprechende Daten zur sicheren Unterscheidung von Benutzern aus dem Request zu extrahieren. Sie kann

dazu in Form eines Intercepting Filter realisiert werden. Die zweite Komponente der Mehrbenutzerunterstützung kennt die verschiedenen Benutzer und nutzt die Daten der Benutzerauthentifizierung zur Zuordnung eines Requests mit dem dazugehörigen Benutzer. Die Komponente der Personalisierung greift beispielsweise auf diese Zuordnung zurück, um ihre Dienstleistung zu erbringen.

4.3.2.3 Strategien zur Unterstützung der Modifizierbarkeit

In diesem Abschnitt wird dargestellt, welche Strategien aus 4.2.4 zusätzlich zum Reflection / Meta-Modell-Ansatz in welcher Form in der vorgeschlagenen Architektur zum Einsatz kommen.

Änderungen lokal halten:
Die Menge an Komponenten, die bei einer Anpassung geändert werden müssen, wird durch die Trennung in verschiedene Zuständigkeiten begrenzt. Dies verhindert zwar nicht, dass bei größeren Änderungen, die mehrere Funktionsbereiche betreffen, auch mehrere Komponenten geändert werden müssen, aber es erleichtert das Identifizieren der betroffenen Komponenten. Dies hilft, die semantische Kohärenz der Komponenten zu erhalten. Außerdem unterstützt die Komponentenaufteilung die Generalisierung der Komponenten. Die Details liegen hier in der Verantwortung der Implementierung. Es sollte darauf geachtet werden, dass die Schnittstelle der Komponenten nicht spezifischer als nötig gehalten ist. Durch die Generalität der Komponenten ist es dem Dispatcher möglich, durch angepasste Aufrufe der Komponenten auch andere Szenarien abzudecken.

Information Hiding:
Jede Komponente verbirgt Eigenschaften und Verhalten, das nicht nach außen hin sichtbar sein muss. Dies ist ebenfalls ein Detail der Implementierung, das von allen modernen objektorientierten Programmiersprachen unterstützt wird.

Bindungszeitpunkt verzögern:
Die Komponenten sind zunächst lose miteinander verbunden. Der Einsatz des Modells eines bestimmten Szenarios und dessen Interpretation durch den Dispatcher, ermöglicht es, die Komponenten erst zur Laufzeit so zu kombinieren, wie es die Aufgabe erfordert.

4.3.2.4 Zusammenfassung

Das vorgestellte Komponentenmodell ist möglichst allgemein gehalten. Es lässt die genauen Kooperationsbeziehungen zwischen den Komponenten noch offen, da diese sich erst durch das konkrete Szenario ergeben, in dem das Publikationssystem eingesetzt wird. Im Abschnitt 5.4 wird anhand eines Beispiels beschrieben, wie diese Anpassung ablaufen kann und wie das Ergebnis dieser Anpassung aussieht.

4.3.3 Richtlinien für die Entwicklung von Publikationssystemen

Die Entwicklung neuer Publikationssysteme sollte bevorzugt inkrementell geschehen. Sie beginnt damit, dass man aus der Liste von Szenarien in Abschnitt 3.2 ein initiales Szenario wählt, das dem voraussichtlichen Hauptszenario entspricht oder möglichst nahe kommt. Falls das zu implementierende Szenario zu verschieden von den in dieser Arbeit betrachteten Szenarien ist, kann alternativ auch eine eigene Bewertung der Bedeutung der Funktionscluster im geplanten Szenario vorgenommen werden. Ausgehend von der Bewertung der Funktionscluster im Szenario wird festgelegt, in welcher Reihenfolge die Funktionscluster implementiert werden.

Wenn alle wesentlichen Funktionscluster, die für das Szenario erforderlich sind, implementiert sind, kann das Publikationssystem bereits für dieses Szenario eingesetzt werden. Entweder man belässt es dabei, oder man sucht nach weiteren Szenarien, die unterstützt werden sollen und implementiert nach und nach die noch fehlenden Funktionscluster.

Wenn ein bereits bestehendes Publikationssystem erweitert werden soll, orientiert man sich wiederum an der Bewertung der Funktionscluster im geplanten Szenario und implementiert die noch fehlenden und passt die bestehenden an die geänderten Anforderungen an. Die Reihenfolge kann sich dabei wieder an der Wichtigkeit der Funktionscluster orientieren.

4.4 Ausgewählte Systeme

Publikationssysteme stellen den grundlegenden Rahmen für die Realisierung von Publikationsanwendungen dar und müssen daher in erster Linie den Anforderungen aus Abschnitt 2.5 gerecht werden. In diesem Abschnitt werden unterschiedliche Publikationssysteme ausgewählt und näher untersucht, inwieweit sie diese Anforderungen erfüllen. Neben TYPO3 und Cocoon wurde mit Mambo noch ein weiteres System untersucht, das aber sehr ähnlich zu TYPO3 ist und daher nicht in diese Arbeit aufgenommen wurde. Die Beschreibung dieses Systems ist in [Sch09] zu finden.

4.4.1 TYPO3

Bei TYPO3[9] handelt es sich um ein Open Source Content Management System für mittlere bis große WebSites, das sehr weit verbreitet ist. Weltweit sind mehrere 10.000 Installationen von TYPO3 im Einsatz [Skå]. Darunter finden sich auch die WebSites bekannter Unternehmen und Institutionen wie Volkswagen, Villeroy & Boch, Karstadt, ThyssenKrupp, Metabo, die Homepages von Pixelpark, der Technischen Universität München, 1-2-3.tv und von Bündnis 90/Die Grünen.

TYPO3 besitzt einen großen Funktionsumfang und soll nach Meinung der Autoren auch zu hochpreisigen kommerziellen Content Management Systemen konkurrenzfähig sein. Dementsprechend hoch sind allerdings auch die Anforderungen an die Hardwarebasis. Auch die Zeit

[9] TYPO3.org: `http://typo3.org`

für die Einarbeitung in die Bedienung und Programmierung von TYPO3 ist nicht zu vernachlässigen.

Kasper Skårhøj begann im Jahr 1997 mit der Entwicklung [Skå05a]. Nach einer ersten Phase der kommerziellen Entwicklung wurde TYPO3 als Open Source freigegeben. Inzwischen kommt der Open Source Community eine große Bedeutung bei der Weiterentwicklung zu, nicht zuletzt durch Entwicklung von sogenannten Extensions.

TYPO3 ist vollständig in PHP geschrieben, die Softwareanforderungen für den Einsatz des Systems umfassen:

- HTTP-Server (z. B. Apache HTTP Server).
- MySQL Datenbank
- PHP4
- einige Grafikbibliotheken (GDLib, Freetype, ImageMagick) für die Unterstützung des vollen Funktionsumfangs inkl. automatischer Generierung von Grafiken.

Diese Anforderungen können sowohl von Windows- als auch von Unix-/Linux-Systemen erfüllt werden.

TYPO3 ist modular aufgebaut und kann mit von anderen Nutzern oder selbstentwickelten Plug-Ins (den Extensions) erweitert werden (siehe Abbildung 4.5 auf Seite 133). Derzeit sind über 1000 Erweiterungen registriert[10], mit denen sich sehr viele Anwendungsfälle (bis hin zu vollständigen WebShops) abdecken lassen, ohne dass eigener Code geschrieben werden muss. Ein in TYPO3 integrierter „Extension Manager“ bietet die benötigten Managementfunktionen für den Umgang mit Erweiterungen.

TYPO3 nutzt sogenannte „Templates“ zur Darstellung des Content, in denen Seitenaufbau und Formate definiert werden (z. B. an welcher Stelle die Navigationsleisten und Inhalte angezeigt werden, die Schriftfarben und -größen, die Positionierung von Überschriften, etc.). Der verwaltete Content steht unabhängig vom jeweils verwendeten Template zur Verfügung. Dadurch kann das Erscheinungsbild der WebSite durch den Austausch des Template vollständig geändert werden, ohne den Content erneut erstellen zu müssen. Somit können auch dieselben Inhalte in unterschiedlichen Layouts präsentiert werden. Zu einem Template gehört neben TypoScript Anweisungen auch ein HTML-Gerüst, das den Grundrahmen der Präsentation festlegt. In diesem HTML-Gerüst sind über Platzhalter Bereiche gekennzeichnet, die erst zur Laufzeit von TYPO3 durch Inhalte aus der Datenbank ersetzt werden. Die TypoScript Anweisungen konfigurieren TYPO3 für die Verwendung des HTML-Gerüsts.

Ein Template besteht auf verschiedenen Bestandteilen:

Statische Teile:
bleiben auf allen Seiten gleich (z. B. Logos, Hintergrundgrafiken).

Indirekt-dynamische Teile:
werden auf jeder Seite durch das Template aktualisiert.

[10] TYPO3.org: Repository: `http://typo3.org/extensions/`

Dynamische Teile:
werden aus der Datenbank entnommen. Das Template hat keinen Einfluss auf diese Teile, außer der Festlegung von Schriftarten und Farben.

TYPO3 verfügt über die Möglichkeit, die WebSeiten zu statifizieren, sie also im voraus zu generieren und als Dateien abzulegen, so dass Requests schneller beantwortet werden können.

Alle Zugriffe auf TYPO3 erfolgen über den Browser. Dabei wird unterschieden zwischen dem sogenannten „Front-End“ und dem „Back-End“. Das Front-End stellt die Sicht für normale Nutzer auf die verwalteten Inhalte dar (die von TYPO3 generierte WebSite), während das Back-End die Administrationssicht bildet. Hier werden die Inhalte erstellt und alle weiteren Verwaltungstätigkeiten durchgeführt. Für die Arbeit im Back-End ist eine Anmeldung mit Benutzername und Passwort erforderlich.

Die WebSeiten werden von TYPO3 in einer Baumstruktur – dem sogenannten „Page Tree“ – verwaltet. Eine Seite – oder „Page“ – besteht wiederum aus untergeordneten Bestandteilen: den sogenannten „Page Content Elements“[11]. Eine Page kann dabei aus beliebig vielen Page Content Elements mit unterschiedlichen Typen zusammengesetzt sein. Mögliche Typen für Page Content Elements sind beispielsweise Text mit Überschrift, Text mit Grafik, Aufzählungslisten, Tabellen, Bildlisten, Such- und Login-Formulare und Erweiterungen wie Gästebücher und Diskussionsforen. Um eine Page zu erstellen, werden also die enthaltenen Page Content Elements erstellt. Für den Typ „Text“ können verschiedene HTML-Tags benutzt werden. Beim Einsatz des Microsoft Internet Explorer steht auch ein WYSIWYG-Editor zur Verfügung. Verknüpfungen zu anderen Seiten werden über ein spezielles Tag („<LINK>“) definiert.

TYPO3 kann über PHP praktisch beliebig erweitert werden. Außerdem kann es über TypoScript weitreichend angepasst werden. Bei TypoScript handelt es sich mehr um eine deklarative Konfigurationsprache als eine Skriptsprache [Skå05b]. TypoScript erzeugt eine Datenhierarchie, die intern von der Template-Engine benutzt wird, um zu bestimmen, was in welcher Reihenfolge erledigt werden soll. Der Ansatz ist vergleichbar mit dem Konzept der Registrierungsdatenbank des Windows Betriebssystems. Die eigentliche Skriptsprache von TYPO3 ist PHP. TypoScript verbindet Content, Templates und PHP miteinander.

4.4.1.1 Bewertung

Es stellt sich die Frage, wie und an welcher Stelle TYPO3 in die Architektur von Publikationssystemen integriert werden kann. Zunächst ist festzustellen, dass TYPO3 nicht alle Anforderungen erfüllt, die an Publikationssysteme gestellt werden, somit also nicht als vollständiges Publikationssystem dienen kann, sondern höchstens als eine Komponente eines Publikationssystems. Die Hauptgründe, die es als Publikationssystem ausschließen, sind:

- TYPO3 selbst unterstützt nur eine einzige Datenquelle, nämlich die angebundene MySQL Datenbank.

[11] An verschiedenen Stellen in der Literatur werden auch die kürzeren Bezeichnungen „Content Elements“ oder einfach „Elements“ verwendet.

- Unzureichende Unterstützung der Wiederverwendung und Komposition neuer Inhalte aus bestehenden Inhalten. Hierzu zählt auch die fehlende Adressierbarkeit von Dokumenten(teilen) sowie die fehlende Unterstützung der Integration und Kombination mehrerer Datenquellen.
- Die Verwendung eines nur gering strukturierten Dokumentenmodells ohne die Möglichkeit zur Erweiterung der erlaubten Elementnamen. Eine Prüfung der korrekten Strukturierung ist ebenfalls nicht möglich. Es wird eine spezifische Ausprägung des Modells der strukturierten Dokumente in der Form von HTML-Auszeichnungen verwendet. Für TYPO3-spezifische Konstrukte (wie etwa Verknüpfungen) wird jedoch nicht die HTML-Syntax verwendet, sondern eine proprietäre Syntax. Dementsprechend bemüht sich TYPO3 nicht um die Unterstützung der besonderen Möglichkeiten von strukturierten Dokumenten. Hierzu zählen die automatischen Bearbeitungsschritte sowie eine Schnittstelle zu Programmiersprachen.
- Es wird als einziges Ausgabeformat lediglich HTML unterstützt.
- Es gibt nur geringe Möglichkeiten zur Personalisierung der bereitgestellten Inhalte.
- Die Mehrsprachigkeitsunterstützung muss noch weiter ausgebaut werden.

In Abschnitt 4.4.1.2 werden die Anforderungen an Publikationssysteme aus Abschnitt 2.5 einzeln betrachtet.

Somit ist noch die Frage zu beantworten, ob TYPO3 als Teilkomponente eines Publikationssystems einsetzbar ist. Dabei gibt zwei mögliche Einsatzszenarien:

1. TYPO3 als Datenquelle in einem Publikationssystem.
2. TYPO3 als Front-End eines Publikationssystems, erweitert um noch fehlende Bestandteile.

TYPO3 als Datenquelle TYPO3 ist zunächst als Content Management System auf die Verwaltung einzelner Inhalte ausgelegt. Allerdings ist die typische Nutzung eines CMS nicht das Abrufen einzelner Inhalte wie es beim Einsatz als Datenquelle der Fall wäre, sondern das CMS bettet die Inhalte in eine Web-Seite ein, die neben Navigationselementen wie Menüs noch weitere Seitenbestandteile enthält, die meist in keinem engen logischen Zusammenhang mit dem Inhalt stehen. Es wäre aber möglich, TYPO3 mit einem entsprechenden Template zu versehen, das alle „inhaltsfremden“ Bestandteile ausblendet und nur den angeforderten Inhalt ausliefert. Dabei ist jedoch das Problem zu lösen, dass die bereitgestellten Inhalte nur grob strukturiert sind. Das verwendete HTML bietet nur sehr begrenzte Möglichkeiten zur Auszeichnung semantischer Rollen der Elemente. Die Weiterverarbeitung im Front-End des Publikationssystems wird dadurch zumindest deutlich erschwert.

TYPO3 erweitert um fehlende Bestandteile TYPO3 bietet bereits sehr viele Funktionen, die von einem Publikationssystem gefordert werden (u. a. Benutzerverwaltung, Management von Zugriffsrechten, Versionierung). Im Wesentlichen fehlt Funktionalität in den Bereichen:

- Unterstützung verschiedener Datenquellen
- Verwendung eines strukturierten Dokumentenmodells. Damit einhergehend:
  - Unterstützung verschiedener Ausgabeformate
  - Unterstützung automatischer Bearbeitungsschritte

Es ist denkbar, TYPO3 entsprechend zu erweitern. Der Aufwand hierfür ist schwer abzuschätzen, dürfte jedoch beträchtlich sein. Es gibt die Möglichkeit, ein Template so zu erweitern, dass Funktionen in PHP beim Anfordern einer Seite durch den Benutzer aufgerufen werden. Hier kann angesetzt werden, um weitere Datenquellen zu integrieren oder sogar XML-Daten durch einen XSLT-Prozessor zu leiten [Skå05c]. Allerdings stehen für diese Erweiterungen nicht die bereits vorhandenen Möglichkeiten zur Bearbeitung und Verwaltung in TYPO3 zur Verfügung. Eine entsprechende Integration müsste ebenfalls programmiert werden. Das Hauptproblem ist, dass die Architektur von TYPO3 nicht unter Einbeziehung der Möglichkeiten von strukturierten Dokumenten entworfen wurde, so dass sehr wahrscheinlich bedeutende Änderungen nötig sind, die sich über alle Bereiche des Systems erstrecken können.

Unabhängig von der Überlegung, TYPO3 als Front-End- oder Back-End System einzusetzen, gibt es folgende Kritikpunkte:

Fehlende horizontale Skalierbarkeit:
In TYPO3 ist keine Lastverteilung auf mehrere Server oder Cluster vorgesehen.

Programmierunterstützung:
Die Erstellung von Templates und TypoScript wird nur durch statische Sichten auf die durch TypoScript konfigurierten Werte und Objekte unterstützt. Die Beobachtung der dynamischen Abläufe zum Zeitpunkt einer Dokumentenanforderung ist jedoch nicht möglich. Dies kann schnell zu einem erhöhten Zeitaufwand beim Debugging führen.

4.4.1.2 Erfüllung der Anforderungen an Publikationssysteme

A1.1 Dynamisches Zusammenstellen der Inhalte:
Von TYPO3 erfüllt. Die Inhalte werden zum Zeitpunkt des Abrufens durch den Benutzer aus der Datenbank geladen und zusammengestellt. TYPO3 verfügt über Caching-Mechanismen, um die Antwortzeiten für häufig angeforderte Seiten zu verringern. Zusätzlich können Seiten im voraus generiert und statisch abgelegt werden, was die Antwortzeiten weiter verbessert.

A1.2 Unterstützung verschiedener Datenquellen:
Von TYPO3 nicht erfüllt. Alle Inhalte werden in einer einzigen Datenbank abgelegt. Eine Verteilung über mehrere Datenbanken – und damit auch die gleichzeitige Verwendung verschiedener Datenquellen – ist nicht vorgesehen. Eine Seite, die im Browser des Benutzers angezeigt wird, besteht zwar aus mehreren, dynamisch zusammengestellten Inhalten, die Zusammenstellung ist jedoch nicht beliebig frei. Welche Bereiche der Seite getrennt von anderen Bereichen aus bestimmten Inhalten stammen, wird vom Template festgelegt.

Es ist auch nicht vorgesehen, ein einzelnes Page Content Element auf verschiedenen Pages zu verwenden.

A1.3 Adressierbarkeit von Dokumenten:
Wird von TYPO3 nicht unterstützt.

A1.4 Verknüpfbarkeit von Dokumenten:
TYPO3 unterstützt einfache Verknüpfungen in Form von einfachen HTML Links. Dabei können auch gezielt bestimmte Page Content Elements auf einer Seite angesprochen werden.

A1.5 Unterstützung multimedialer Komponenten:
TYPO3 unterstützt multimediale Komponenten auf den Seiten zum einen über die Möglichkeiten, die HTML dafür anbietet. Zum anderen existieren verschiedene Typen von Page Content Elements, die hierfür verwendet werden können.

A1.6 Unterstützung dynamischer Dokumente:
Wird von TYPO3 grundsätzlich unterstützt. Inhalte können dynamisch zusammengestellt werden. Beispielsweise existieren Erweiterungen für Gästebücher oder Diskussionsforen, die dynamisch aus in der Datenbank gespeicherten Ergebnissen eine Inhaltsseite generieren.

A1.7 Unterstützung interaktiver Dokumente:
Wird von TYPO3 unterstützt.

A1.8 Definition und Validierung eines gemeinsamen Vokabulars:
TYPO3 verwendet für alle Inhalte das Vokabular, das durch den HTML-Standard definiert ist. Es ist nicht möglich, davon abweichend eigene Elementtypen zu definieren und Dokumente auf die Einhaltung einer bestimmten Struktur zu prüfen.

A1.9 Unterstützung externer Präsentationsanweisungen:
Wird von TYPO3 durch Templates unterstützt. Im HTML-Gerüst können beliebige Präsentationsanweisungen, normalerweise in Form von CSS, festgelegt werden. Die Trennung ist allerdings nicht konsequent durchgezogen. Dazu dürften in den Inhalten gar keine Präsentationsanweisungen auftauchen. Bestimmte HTML-Elemente können jedoch weiterhin direkt in den verwalteten Content eingebettet werden, z. B. das <b>-Tag.

A1.10 Late Binding von Präsentationsanweisungen:
Wird unterstützt. Die Präsentationsanweisungen sind in Templates bzw. das HTML-Gerüst (inkl. CSS) ausgelagert.

A1.11 Unterstützung verschiedener Ausgabeformate:
Wird nicht unterstützt.

A1.12 Unterstützung automatischer Bearbeitungschritte:
Über Extensions gibt es zwar die Möglichkeit, beliebige Funktionalität mit mehr oder weniger großem Aufwand einzubinden, allerdings ist es grundsätzlich in TYPO3 nicht vorgesehen, Page Content Elemente zu transformieren, zu kombinieren oder anderweitig automatisch zu bearbeiten.

A1.13 Schnittstellen zu Programmiersprachen:
Es ist keine besondere Unterstützung der Verknüpfung der Dokumenteninhalte mit Programmiersprachen vorgesehen.

A2.1 Verwaltung aller Ressourcen einer Publikationsanwendung:
Wird unterstützt. In TYPO3 können in sich geschlossene Publikationsanwendungen implementiert werden, bei denen alle benötigten Ressourcen von TYPO3 verwaltet werden. Dies schließt nicht die Implementierung von Publikationsanwendungen aus, bei denen dies nicht der Fall ist.

A2.2 Management verschiedener Ressourcen und Datentypen:
Wird grundsätzlich unterstützt. Die Art des Managements ist dabei festgelegt auf zwei verschiedene Ressourcenarten: Zum einen gibt es die Inhalte, deren Präsentation von Templates festgelegt wird. Zum anderen gibt es den „Rest“. Darunter fallen Grafiken, Videos und sonstige Dateien (Word Dokumente, ZIP-Archive). Diese Dateien können in einem von den normalen Inhalten getrennten Pool abgespeichert und referenziert werden. Templates haben keinen Einfluss auf die Präsentation dieser Daten.

A2.3 Dynamisches Generieren der angeforderten Ressourcen:
Wird grundsätzlich unterstützt.

Datenquellen integrieren und kombinieren:
Wird in begrenztem Maß durch TYPO3 unterstützt. Templates spielen ebenfalls eine Rolle, allerdings gibt es hier eine recht strikte Aufteilung, welche Bereiche der Seite aus unterschiedlichen Datenquellen stammen. Die Art der Datenquelle ist eingeschränkt auf die TYPO3-eigene Datenbank. Dies könnte ebenfalls durch Extensions umgangen werden, ist aber nicht vorgesehen.

Dynamische Bestandteile ausfüllen:
Wird in Form von Extensions unterstützt.

In Präsentationsformate überführen und ausliefern:
Wird unterstützt.

A2.4 Unterstützung der Erstellung der Dokumente:
TYPO3 verfügt über einen integrierten Editor, mit dem die verwalteten Inhalte erstellt werden können. Externe Editoren werden nicht unterstützt. Der Editor unterstützt nur wenige Elemente, die HTML anbietet. Allerdings ist die Philosophie der Seitenerstellung bei TYPO3 etwas anders als bei vielen anderen Content Management Systemen. In TYPO3 werden nur Page Content Elements direkt editiert. Für jede Seite wird dann festgelegt, welche Page Content Elements in welcher Reihenfolge enthalten sind.

A2.5 Mehrbenutzerfähigkeit:
Wird unterstützt und zwar sowohl für das Back-End wie auch das Front-End der WebSite. Sperren bei der Editierung im Back-End arbeiten dabei nicht mit zwangsweiser Durchsetzung. Stattdessen wird nur angezeigt, dass bereits ein anderen Benutzer dieses Element bearbeitet.

A2.6 Navigationsunterstützung:
Hier bietet TYPO3 eine recht einfach gehaltene Unterstützung in Form von automatisch

generierten Menüstrukturen an, die dem Benutzer einen Anhaltspunkt liefern, wo er sich gerade befindet und ihm ermöglicht, leicht zu übergeordneten Dokumenten zu navigieren.

A2.7 Unterstützung von Wiederverwendung:
Wird lediglich dadurch unterstützt, dass einzelne Page Content Elements im Back-End einfach kopiert und in eine andere Seite eingefügt werden können. Änderungen am Original haben anschließend keine Auswirkungen auf die Kopie und umgekehrt. Die Kombinierbarkeit mehrerer Page Content Elements zu einem neuen Page Content Element ist nicht vorgesehen.

A2.8 Unterstützung verschiedener Medienkanäle:
Es wird nur der Kanal „Internet“ unterstützt. Es gibt keine besondere Unterstützung für andere internetfähige Endgeräte als die üblichen Browser (vgl. vorherigen Punkt).

A2.9 Versionierung und Change-Management:
Wird unterstützt. Alle Änderungen werden aufgezeichnet. Es gibt eine Undo-Funktion mit unbegrenzter Anzahl der Wiederherstellungsschritte und eine vollständige Versionshistorie.

A2.10 Wiederherstellung eines bestimmten Standes:
Wird unterstützt. Zum einen kann der aktuelle Stand aus der Datenbank exportiert werden, zum anderen können die Versionsstände der einzelnen Seiten gezielt zurückgesetzt werden.

A2.11 Vergleich unterschiedlicher Versionen:
Verschiedene Versionen einer Seite können miteinander verglichen werden.

A2.12 Rechte-Verwaltung:
Es können Benutzer und Gruppen verwaltet werden. Für jede Seite im Seitenbaum können detailliert Berechtigungen festgelegt werden. Auch die restlichen Verwaltungsbereiche von TYPO3 können eingeschränkt werden.

A2.13 Änderungsprotokolle:
Werden unterstützt.

A2.14 Beachtung von Urheberrechten:
Wird nicht unterstützt. TYPO3 geht davon aus, über alle Urheberrecht zu verfügen.

A2.15 Unterstützung von Recherche und Information-Retrieval:
Wird unterstützt in Form einer Volltextsuche über alle von TYPO3 verwalteten Inhalte. Dabei können auch Dokumente in OpenOffice- und Microsoft-Office-Formaten für die Suchfunktion indiziert werden.

A2.16 Unterstützung von Suchwerkzeugen:
Neben der integrierten Volltextsuche kann die Verwendung externer Suchwerkzeuge durch „Search Engine Friendly URLs“ erleichtert werden.

A2.17 Modellierung und Steuerung des Publikationsprozesses:
Die Publikationskette in TYPO3 besteht nur aus wenigen Stationen. Es gibt keine besondere Unterstützung zur Modellierung und Steuerung des Prozesses.

A2.18 Verarbeitungsfunktionen:
Werden nicht unterstützt.

A2.19 Personalisierungsfunktionen:
Die Möglichkeiten zur Personalisierung fallen gering aus. Grundsätzlich ist eine rudimentäre Benutzerverwaltung vorhanden, so dass sich Benutzer anmelden und von einander unterschieden und wiedererkannt werden können. Dabei ist es aber z. B. nicht vorgesehen, verschiedene Benutzer unterschiedliche Templates auswählen zu lassen. Es ist möglich, den Zugriff auf bestimmte Bereiche der WebSite nur für bestimmte Benutzer zu erlauben.

A2.20 Unterstützung von Mehrsprachigkeit:
Wird unterstützt. Für alle Seiten können mehrere Sprachversionen gepflegt werden. Auch das Back-End kann in mehreren Sprachen angezeigt werden.

A2.21 Flexibilität in Bezug auf Online/Offline Nutzung:
Wird nicht mit spezieller Funktionalität unterstützt.

A2.22 Unterstützung eines redaktionellen Workflow:
TYPO3 besitzt ein internen „Task center“, mit dem Aufgaben zugewiesen und bearbeitet werden können. Der Workflow kann über das „Task center“ gesteuert werden. Hier können Aufgaben an bestimmte Nutzer zugewiesen werden (z. B. das Erstellen eines Textes). Anschließend kann das Ergebnis überprüft werden. Es ist nur die Unterstützung eines einfachen, linearen Workflows vorhanden. Komplexe Workflows mit Verzweigungen und Parallelästen werden nicht unterstützt.

Basierend auf diesen Ergebnissen kann man nun mit Hilfe von Abschnitt 3.1 vergleichen, welche Funktionscluster von Typo3 grundsätzlich unterstützt werden. Dabei handelt es sich um die Funktionscluster Navigation, Multimediateile, Interaktivität, Trennung Präsentation, Dynamik, Autorenunterstützung, Mehrbenutzer, Versionierung, Rechte Management, Suchmöglichkeit, Personalisierung und Mehrsprachigkeit. Vergleicht man dies wiederum mit den szenarienspezifischen Signifikanzen aus der Tabelle 3.2 auf Seite 81, dann ist das Szenarien mit den meisten Übereinstimmungen das Szenario „Freizeitcommunities“ und das mit den wenigsten das Szenario „Redaktionssysteme von Fachverlagen“.

4.4.2 Apache Cocoon

In diesem Abschnitt wird die Software Apache Cocoon[12] (kurz: Cocoon) näher untersucht. Cocoon verfolgt einen Ansatz beim Umgang mit Dokumenten, der sowohl konzeptuell als auch technisch interessant erscheint. Wichtige konzeptuelle Ansätze in Cocoon sind die Kodierung der Dokumente in XML und die Transformation dieser Dokumente mit XSLT. Cocoon setzt also aktuelle und allgemein anerkannte Standards ein. Dies und die Tatsache, dass es als Open Source Projekt entwickelt wird, machen Cocoon weitgehend herstellerunabhängig. Projekte, die Cocoon einsetzen sind daher nicht von den Entscheidungen und dem Schicksal eines einzelnen Herstellers abhängig, sondern können Cocoon auch selbst mit relativ geringem Aufwand an ihre Bedürfnisse anpassen und weiterentwickeln.

[12]The Apache Cocoon Project: `http://cocoon.apache.org`

Cocoon wird auf der Internetseite des Projekts als „Web Development Framework“ bezeichnet. Aufgrund seiner Eigenschaften wird es teilweise auch als „XML Publishing Framework“ bezeichnet (vgl. [MA03, Nie04]). Die beiden Bezeichnungen schließen sich nicht gegenseitig aus und ich halte beide für zutreffend. Abhängig davon wie Cocoon konkret eingesetzt wird, ist mal die eine und mal die andere Bezeichnung zutreffender.

Cocoon ist ein „Web Development Framework“, da es einen Grundrahmen für die Entwicklung webbasierter Software zur Verfügung stellt. Diese Bezeichnung ist etwas allgemeiner als „XML Publishing Framework“. Die Bezeichnung als Framework ist beiden gemeinsam. Die Kerntechnologie von Cocoon ist eindeutig XML. Cocoon verarbeitet strukturierte Dokumente, die mit Hilfe von XML dargestellt sind.

Die wesentliche Idee hinter Cocoon ist der Einsatz von sogenannten „Pipelines“ zur Bearbeitung und Beantwortung von Requests (vgl. Abbildung 4.6 auf Seite 133). Der Bereich zwischen Request und Response wird dabei mit Hilfe der Pipeline in einzelne Arbeitsschritte unterteilt.

Die wichtigsten Bestandteile einer Pipeline sind Generatoren, Transformer und Serializer. Generatoren stehen am Anfang der Bearbeitungskette und erzeugen einen SAX-Strom, der an den ersten Transformer weitergereicht wird. Transformer können diesen SAX-Strom mit Hilfe von XSLT beliebig bearbeiten. Am Ende der Bearbeitungskette steht ein Serializer, der aus dem SAX-Strom das endgültige Ausgabedokument erzeugt.

Cocoon verwendet zur Konfiguration eine sogenannte Sitemap, in der mehrere Pipelines definiert werden können. Abhängig vom Request des Benutzers (hierzu dienen sogenannte Matcher und Selektoren) wird eine Pipeline ausgewählt, zusammengestellt (d. h. benötigte Objekte werden instanziiert und vorbereitet) und durchlaufen. Als Besonderheit nimmt Cocoon eine Virtualisierung des Requests vor, d. h. innerhalb der Pipeline sind die speziellen technischen Eigenheiten des Requests verborgen.

Cocoon stellt damit über Standards wie XML, XSLT und SAX eine Möglichkeit zur Aufteilung von Funktionen in einzelne Teilfunktionen bereit. Dies unterstützt die Entwicklung von komponentenbasierten Anwendungen und ermöglicht, dass sich jede Komponente der Pipeline auf eine bestimmte Teilfunktion spezialisieren kann.

Die Komponenten können wie bei einem Baukastensystem miteinander kombiniert werden, um eine bestimmte Anwendung zu implementieren. Momentan gibt es rund 50 solcher Komponenten, die im Cocoon Jargon „Blocks“ genannt werden[13]. Teile, die in mehreren Pipelines gleich sind, können mit Hilfe von sogenannten Ressources an einer einzigen Stelle definiert und beliebig oft in Pipelines wiederverwendet werden. Ressourcen verhalten sich zu Pipelines also wie Unterfunktionen zu Funktionen.

Die Anbindung z. B. einer SQL-Datenbank kann derart geschehen, dass zunächst vom Generator eine Datei mit einer SQL-Anweisung vom Datenträger gelesen wird. Ein spezieller Transformer nimmt die SQL-Anweisung als Eingabe, reicht sie an die SQL-Datenbank weiter und stellt die Ergebnisse der Datenbankabfrage wieder in Form eines SAX-Stroms zur Verfügung. Ein weiterer Transformer ist dann in der Lage, mit diesem Ergebnis zu arbeiten und die Ausgabe durch den Serializer vorzubereiten.

[13] BlockDescriptions: `http://wiki.apache.org/cocoon/BlockDescriptions`

Neben dieser baukastenartigen Kombinierbarkeit unterstützt Cocoon das sogenannte „Separation-of-Concerns“, indem Inhalte, Stilvorlagen und Programmlogik voneinander getrennt werden. Diese Trennung wird unterstützt durch das sogenannte „Control Flow“[14]. Control Flow erleichtert es auch, Abläufe zu erstellen, die mehrere Interaktionsschritte mit dem Nutzer erfordern und nicht schon nach einem einfachen Request-Response-Zyklus abgeschlossen sind. Es modelliert hierbei den Kontrollfluss mit Hilfe eines JavaScript Programms und verwendet sogenannte „Continuations“ als Wiedereinstiegspunkte an beliebigen Stellen.

Dies lässt Cocoon grundsätzlich sehr gut geeignet erscheinen, um als Publikationssystem zur Realisierung von Publikationsanwendungen zu dienen. Cocoon bietet beispielsweise eine entsprechende Architektur, die auf die dynamischen Aspekte strukturierter Dokumente fokussiert, um deren Möglichkeiten auszunutzen. Wir betrachten daher im Folgenden näher, welche Anforderungen aus Abschnitt 2.5 es bereits erfüllt, welche Komponenten aus Abschnitt 4.2.5.1 vorhanden sind und schließlich, wo noch Entwicklungsbedarf besteht.

An dieser Stelle wird nicht detailliert auf die Funktionen und Eigenschaften von Cocoon eingegangen. Für einen kurzen Überblick sei auf die Features-Liste auf der Webseite des Projekts[15] und die umfangreiche Darstellung in [Nie04] verwiesen.

4.4.2.1 Erfüllung der Anforderungen an Publikationssysteme

A1.1 Dynamisches Zusammenstellen der Inhalte:
Von Cocoon erfüllt. Die Inhalte werden zum Zeitpunkt des Abrufens durch den Benutzer von einem Generator erzeugt (aus einer Datenquelle generiert) und in die Pipeline eingespeist. Zur Verbesserung der Performanz verfügt Cocoon über mehrere Caching-Mechanismen auf verschiedenen Ebenen. Zusätzlich können Seiten auch im voraus generiert und statisch abgelegt werden, was die Antwortzeiten weiter verbessert.

A1.2 Unterstützung verschiedener Datenquellen:
Von Cocoon erfüllt. Durch den Pipeline-Ansatz ist Cocoon so flexibel, dass beliebige Datenquellen integriert und gleichzeitig genutzt werden können. Es gibt bereits rund zwei Dutzend verschiedene Generatoren. Manche Datenquellen wie z. B. MySQL-Datenbank benötigt keinen eigenen Generator sondern es wird der Standard-Filegenerator benutzt, um eine Datei mit einem SQL-Ausdruck einzulesen. Das Ergebnis der SQL-Abfrage wird dann von einem Transformer in ein XML-Format umgewandelt und so der Pipeline zur Verfügung gestellt.

A1.3 Adressierbarkeit von Dokumenten:
ist problemlos möglich, da alle Daten (und damit auch die Dokumente) grundsätzlich als SAX-Strom zur Verfügung gestellt werden und in nachfolgenden Transformationsschritten mit Hilfe von XSLT beliebige Teile davon ausgewählt und weiterverwendet werden können.

[14] Apache Cocoon - Control Flow: `http://cocoon.apache.org/2.1/userdocs/flow/index.html`
[15] Cocoon Features: `http://cocoon.apache.org/2.1/features.html`

A1.4 Verknüpfbarkeit von Dokumenten:
Wird von Cocoon unterstützt. Da jede Verknüpfung auch wieder mit einem eigenen Pipeline-Durchlauf verbunden ist, der mit eigenen Parametern initiiert werden kann, kann bei jeder Verknüpfung auch wieder komplexe Programmlogik zur Ausführung kommen.

A1.5 Unterstützung multimedialer Komponenten:
Wird von Cocoon erfüllt. Auch wenn die idealen Dateiformate zur Verarbeitung in Cocoon auf XML basieren, ist es gleichzeitig möglich, Inhalte in beliebigen Binärformaten zur Verfügung zu stellen.

A1.6 Unterstützung dynamischer Dokumente:
Wird von Cocoon erfüllt. Die gesamte Architektur von Cocoon ist auf dynamische Generierung und Bearbeitung von Dokumenten ausgelegt.

A1.7 Unterstützung interaktiver Dokumente:
Wird von Cocoon erfüllt. Prinzipiell ist innerhalb der Dokumente alles an interaktiver Unterstützung möglich, was durch das finale Dokumentenformat (meist HTML mit CSS) angeboten wird.

A1.8 Definition und Validierung eines gemeinsamen Vokabulars:
Wird von Cocoon unterstützt. Mit Hilfe des ValidationTransformers kann die Struktur der Dokumente an beliebigen Stellen der Pipeline auf ihre Konformität mit Strukturvorschriften geprüft werden. Dieser mitgelieferte Transformer bricht jedoch die Pipeline bei einem Fehler einfach ab. Falls also eine eigene Fehlerbehandlung gewünscht wird muss Cocoon hier erweitert werden. Dies ist jedoch mit geringem Aufwand möglich.

A1.9 Unterstützung externer Präsentationsanweisungen:
Wird von Cocoon erfüllt. Der Einsatz von XML ermöglicht die Trennung von Content, Logic und Style. Für die Präsentationsanweisungen können z. B. CSS-Daten mit den generierten HTML-Dokumenten verknüpft werden. Aber auch andere Wege sind möglich, je nach Format des generierten Dokuments.

A1.10 Late Binding von Präsentationsanweisungen:
Wird unterstützt. Die Präsentationsanweisungen sind in CSS-Dateien oder XML-Vorlagen gespeichert.

A1.11 Unterstützung verschiedener Ausgabeformate:
Wird von Cocoon erfüllt. Dieser Schritt wird von Serializern durchgeführt. Cocoon bringt bereits von Haus aus die Unterstützung u. a. der Ausgabeformate PDF, PS, HTML, XHTML, WAP/WML, SVG, JPEG, PNG, TIFF und ZIP mit. Dokumente und Dateien, die nicht in einen XML-Strom umgewandelt werden müssen oder sollen, können über sogenannte Reader direkt eingelesen und ausgegeben werden.

A1.12 Unterstützung automatischer Bearbeitungschritte:
Diese Fähigkeit ist ein wesentlicher Aspekt der Architektur von Cocoon. Durch Transformer können in der Pipeline beliebige Bearbeitungsschritte vorgenommen werden.

A1.13 Schnittstellen zu Programmiersprachen:
Werden angeboten. Cocoon selbst ist in Java programmiert und verwendet XML. Daher kann über SAX auf die Dokumentendaten zugegriffen werden. Darüber hinaus unterstützt

Cocoon über XSP Taglibs und Logicsheets die Einbettung von Java Programmcode in die Dokumente.

A2.1 Verwaltung aller Ressourcen einer Publikationsanwendung:
Wird von Cocoon unterstützt. Es können alle von einer Publikationsanwendungen benötigten Ressourcen im Dateisystem und auch einer oder mehreren Datenbanken gespeichert werden.

A2.2 Management verschiedener Ressourcen und Datentypen:
Wird von Cocoon grundsätzlich unterstützt. Bevorzugt werden aber Ressourcen, die als XML bereitgestellt werden können.

A2.3 Dynamisches Generieren der angeforderten Ressourcen:
Wird von Cocoon unterstützt.

Datenquellen integrieren und kombinieren:
Wird von Cocoon unterstützt. Wenn die vorhandenen Generatoren nicht genügen sollten, kann Cocoon um benutzerdefinierte Generatoren erweitert werden.

Dynamische Bestandteile ausfüllen:
Ist problemlos möglich. Cocoon bietet dazu einen XSP-Prozessor (eXtensible Server Pages), für den Programmcode in sogenannten Logicsheets hinterlegt werden kann.

In Präsentationsformate überführen und ausliefern:
Wird ebenfalls unterstützt. Unter Punkt A1.11 wurden die verfügbaren Präsentationsformate angesprochen.

A2.4 Unterstützung der Erstellung der Dokumente:
Wird von Cocoon nicht unterstützt. Aufgrund der Positionierung als Entwicklungsplattform für webbasierte Anwendungen ist diese Funktionalität offenbar nicht für nötig erachtet worden.

A2.5 Mehrbenutzerfähigkeit:
Wird von Cocoon grundsätzlich unterstützt. Mehrere Requests können gleichzeitig dieselbe Pipeline durchlaufen. Falls während des Durchlaufens schreibend auf Datenquellen zugegriffen wird, ist zu beachten, ob diese Datenquellen Nebenläufigkeit unterstützen. Hier ist es denkbar, Cocoon um eine allgemeine Komponente zu erweitern, die eine Unterstützung der Mehrbenutzerfähigkeit unabhängig von den Eigenschaften konkreter Datenquellen bietet.

A2.6 Navigationsunterstützung:
Hier bietet Cocoon keine besondere Unterstützung an. Es steht aber die technische Basis für die Implementierung einer solchen Unterstützung zur Verfügung.

A2.7 Unterstützung von Wiederverwendung:
Wird von Cocoon dadurch unterstützt, dass alle Daten in XML codiert sind und mit Hilfe von XSLT beliebige Teile der XML-Dokumente ausgewählt werden können. Siehe auch den Punkt A1.3.

A2.8 Unterstützung verschiedener Medienkanäle:
Grundsätzlich kann von Cocoon jeder Kanal bedient werden, der Requests über HTTP unterstützt. Für mobile Geräte wird das WAP Protokoll unterstützt.

A2.9 Versionierung und Change-Management:
Ist in Cocoon nicht vorhanden. Dieser Funktionsbereich muss für eine Publikationsanwendung, die Versionierung benötigt, extra entwickelt werden.

A2.10 Wiederherstellung eines bestimmten Standes:
Diese Anforderung wird folglich auch nicht von Haus aus unterstützt.

A2.11 Vergleich unterschiedlicher Versionen:
Wird von Cocoon ebenfalls nicht unterstützt.

A2.12 Rechte-Verwaltung:
Wird von Cocoon nicht direkt unterstützt. Über die Requests können zwar Parameter zur Unterscheidung von Benutzern mitgeliefert werden, die Nutzung dieser Daten ist jedoch offen gelassen.

A2.13 Änderungsprotokolle:
Werden von Cocoon nicht unterstützt. Dies hängt zusammen mit der noch gering ausgeprägten Unterstützung der Anforderungen A2.9, A2.10 und A2.12.

A2.14 Beachtung von Urheberrechten:
Wird nicht unterstützt. Cocoon kümmert sich nur um die technische Seite der Dokumentenbereitstellung. Eine Komponente, die Urheberrechte berücksichtigt, muss erst entwickelt werden.

A2.15 Unterstützung von Recherche und Information-Retrieval:
Die Volltextsuche über die verwalteten Dokumente wird über einen optionalen „Block“ (Apache Lucene) unterstützt. Apache Lucene kann eine Vielzahl von Dateiformaten indizieren und wird bereits in einigen anderen Software-Systemen genutzt.

A2.16 Unterstützung von Suchwerkzeugen:
Neben der integrierten Volltextsuche wird der Einsatz von externen Suchwerkzeugen durch „Search Engine Friendly URLs“ erleichtert.

A2.17 Modellierung und Steuerung des Publikationsprozesses:
Der Publikationsprozess wird über die Pipeline-Definitionen in der Sitemap modelliert und gesteuert. Da die Sitemap eine normale XML Datei ist, können die üblichen Werkzeuge zur Bearbeitung von XML Dateien eingesetzt werden. Ein spezieller grafischer Editor, der das visuelle Zusammenstellen der Pipeline ermöglicht, wird von Cocoon nicht angeboten.

A2.18 Verarbeitungsfunktionen:
Werden über Transformer in der Pipeline unterstützt, die XSLT zur Umformung der Dokumente verwenden.

A2.19 Personalisierungsfunktionen:
Cocoon bringt von sich aus keine spezielle Unterstützung für die Personalisierung von Dokumenten mit. Technisch unterstützt die Pipeline die Übertragung aller grundlegender Daten, um eine Personalisierung durchzuführen, aber über diese sehr unspezifische Unterstützung hinaus ist noch nichts implementiert.

A2.20 Unterstützung von Mehrsprachigkeit:
Wird grundsätzlich von Cocoon unterstützt. Ebenso wie bei den Personalisierungsfunktionen ist die Unterstützung aber nur sehr grundlegend.

A2.21 Flexibilität in Bezug auf Online/Offline Nutzung:
Wird nicht mit spezieller Funktionalität unterstützt.

A2.22 Unterstützung eines redaktionellen Workflow:
Wird von Cocoon nicht erfüllt. Dies ist eine Folge davon, dass keine Autorenunterstützung (siehe Punkt A2.4) angeboten wird.

In der Abbildung 4.7 sind die Komponenten von Cocoon, die noch fehlen oder deren Funktionalität noch ausgebaut werden muss, durch einen transparenten Hintergrund gekennzeichnet.

4.4.2.2 Einsatz der Architekturmuster

Ein wichtiges Architekturmuster der in Abschnitt 4.3.2 vorgeschlagenen Software-Architektur ist das Model-View-Controller Muster. Die einzelnen Teile werden von Cocoon folgendermaßen realisiert:

Model:
Das Model entspricht der jeweils zur Bearbeitung eines Request ausgewählten Pipeline innerhalb der Sitemap. Die Pipeline fügt die Funktionalität unterschiedlicher Komponenten in Form von einzelnen Bearbeitungsschritten zusammen. In diesem Teil findet der Großteil der Bearbeitung statt.

View:
Der View kümmert sich mit den beiden Komponenten Präsentation und Formatumwandlung um die letzte Stufe der Erzeugung der Dokumente. Diese Funktionalität wird in Cocoon über die Wahl des Serializers zusammen mit der vorhergehenden letzten Stufe der Transformer implementiert.

Controller:
Der Controller-Teil kümmert sich um die Auswahl der zuständigen Pipeline aus der Sitemap und steuert den Ablauf, also das Ausführen der Bestandteile der Pipeline und der Übergabe der Daten von einem Bestandteil zum nächsten.

Als weiteres Architekturmuster kommt in Cocoon das Pipes and Filters Muster aus Abschnitt 4.2.5.2 zum Einsatz. Dadurch dass Cocoon innerhalb einer Pipeline alle Komponenten mit Hilfe eines SAX-Stroms miteinander verbindet, es ist problemlos möglich, die Funktionalität einer Komponente auf mehrere Pipeline-Bestandteile aufzuteilen und diese so über das Pipes and Filters Muster miteinander zu kombinieren.

4.4.3 Zusammenfassung

Die untersuchten Software-Systeme zeigen noch unterschiedlich deutliche Defizite was die Eignung für den Einsatz als vollwertiges Publikationssystem betrifft. Solche vollständigen Publika-

tionssysteme sind offenbar noch nicht Stand der Technik.

Jedoch gibt es mit Cocoon ein System, das mit den eingesetzten Techniken und Konzepten eine vielversprechende Basis für die Weiterentwicklung zum vollständigen Publikationssystemen besitzt. In einigen Bereichen ist noch Entwicklungsaufwand zu leisten, viele wichtige Anforderungen sind jedoch bereits erfüllt.

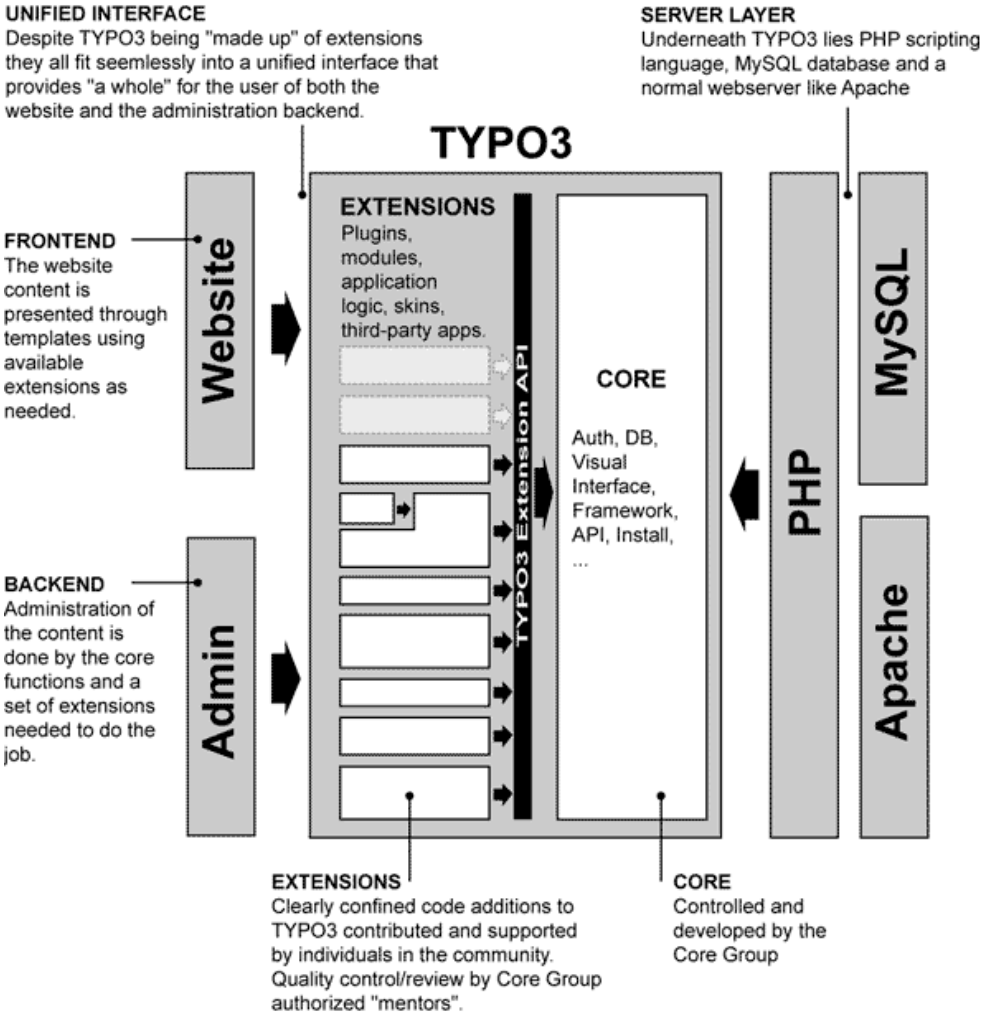

Abbildung 4.5: Architektur von TYPO3 [Skå05d]

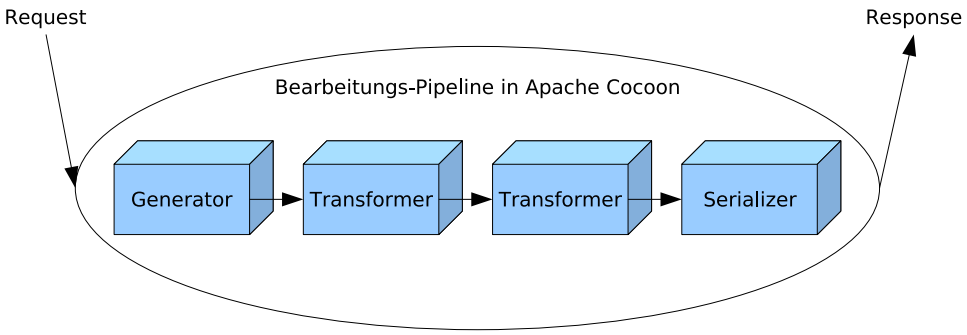

Abbildung 4.6: Cocoon Request/Response-Zyklus

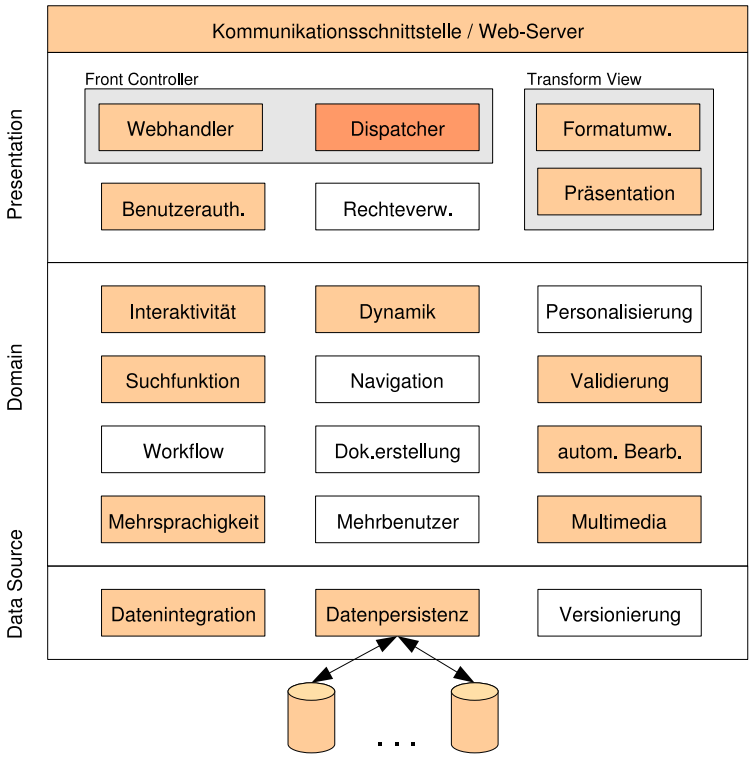

Abbildung 4.7: Cocoon Komponentenübersicht

5 Verbesserung der Modifizierbarkeit durch Einsatz eines Meta-Modells

Il semble que la perfection soit atteinte
non quand il n'y a plus rien à ajouter,
mais quand il n'y a plus rien à retrancher.[1]
– ANTOINE DE SAINT-EXUPÉRY

Das Ziel dieses Kapitels ist es, zur Unterstützung der Modifizierbarkeit von Publikationssystemen die Modellierung von Publikationsanwendungen unter Einsatz der Meta-Modellierung in einer Form zu ermöglichen, so dass diese Modelle von einem Publikationssystem verwendet werden können, um sich an das jeweilige Einsatzszenario anzupassen. Zu diesem Zweck wird nach einer kurzen Beschreibung der Meta-Modellierung und der Eigenschaften und Vorteile von Meta-Modellen ein entsprechendes Meta-Modell entwickelt, das die Eigenschaften und das Verhalten von Publikationsanwendungen beschreibt. Dieses Meta-Modell wird nach seiner Vorstellung anhand eines Beispiel-Szenarios illustriert. Außerdem wird das Vorgehen beschrieben, um das Meta-Modell auf andere Szenarien anzuwenden und zu erweitern. Abschließend wird dargelegt, wie das Meta-Modell in Publikationssystemen genutzt werden kann.

5.1 Einleitung

Die vorliegende Arbeit wendet die in Abschnitt 1.3 beschriebenen Ansätze zur Komplexitätsbeherrschung auf die Domäne der Publikationsanwendungen und -systeme an. Um dabei das Ziel zu erreichen, die Realisierung von Publikationsanwendungen besser zu unterstützen, sind unterschiedliche Möglichkeiten gegeben:

Die erste Möglichkeit ist die Definition eines Meta-Modells. Analytisch betrachtet handelt es sich dabei um die Standardisierung von Modellen, da gefordert wird, dass diese Modelle zum Meta-Modell konform sind. Das Meta-Modell und folglich die zum Meta-Modell konformen Modelle von Publikationsanwendungen können auf zwei unterschiedliche Arten verwendet werden:

Zum einen kann das Modell einer Publikationsanwendung, das dem definierten Meta-Modell entspricht, automatisiert durch Generatoren in ein spezielles Publikationssystem bzw. Code für

[1] „Perfektion ist erreicht, nicht, wenn sich nichts mehr hinzufügen lässt, sondern, wenn man nichts mehr wegnehmen kann."

das zugrunde liegende Publikationssystem übertragen werden. Das Ergebnis muss normalerweise noch manuell durch Code ergänzt und fertig angepasst werden. Das Modell ist allgemein gehalten, die Generatoren hingegen sind spezifisch für das anvisierte Publikationssystem. Dieser Ansatz ist vergleichbar mit dem Vorgehen beim sogenannten Model Driven Architecture (MDA, vgl. [SG00]) der Object Management Group.

Es gibt zur Zeit noch zwei Punkte, die die Einsetzbarkeit von MDA begrenzen (vgl. [Sie04]). Zum einen sind die UML-Sprachen, in denen die plattformunabhängigen Modelle für MDA beschrieben werden, noch vergleichsweise neu und noch nicht durchgängig standardisiert. Es ist auch noch unklar, wie weit sie sich verbreiten werden, denn die Modellierung in einer solchen UML-Sprache kann sich nur dann durchsetzen, wenn die Modellierung damit einfacher und schneller geht als die Programmierung direkt auf der Zielplattform (z. B. J2EE oder .NET). Möglicherweise ist eine Entwicklungsumgebung mit dem Funktionsumfang von J2EE oder .NET eben auch so umfangreich und komplex wie J2EE oder .NET. Der andere Punkt ist, dass der generierte Code oft noch unvollständig ist und die mehr oder weniger großen Lücken erst durch manuelle Programmierung aufgefüllt werden müssen. Dabei bedürfen die Lösungen zu einer Reihe von Problemen (wie unklare Annahmen über den Kontext, unklare Schnittstellen, schwierige Fehlersuche und Probleme bei der Neugenerierung) noch der Verbesserung.

Die zweite Verwendungsart des Meta-Modells besteht darin, es zur dynamischen Konfiguration eines Publikationssystems zu verwenden. Dabei ist das Publikationssystem selbst in der Lage, die Meta-Modell-Sprache zu verstehen und interpretiert das Modell der Publikationsanwendung zur Laufzeit.

Eine weitere Möglichkeit, die Realisierung von Publikationsanwendungen besser zu unterstützen, besteht in der Definition von standardisierten Schnittstellen zu den Komponenten von Publikationssystemen. Ein solcher Ansatz wird beispielsweise von Sun Microsystems Inc. bei J2EE verfolgt, das eine Plattform für die komponentenbasierte Entwicklung von Enterprise-Applikationen ist. J2EE besteht aus einer Sammlung von Standards, die von einer ganzen Reihe von Softwareherstellern in Produkte umgesetzt wurden (z. B. BEA WebLogic, IBM WebSphere und JBoss Application Server). Anwendungen, die zu diesen Standards konform sind, sind herstellerunabhängig auf allen diesen Produkten einsetzbar. Für alle zur Verwendung bereitgestellten Komponenten des Systems muss eine standardisierte Schnittstelle vorhanden sein. Für J2EE wird z. B. momentan daran gearbeitet, eine Schnittstelle für den Zugriff auf sogenannte „Content Repositories“ zu beschreiben (siehe JSR 170 [NP]), in denen Anwendungen bestimmte Daten einfacher verwalten können als mit Hilfe von allgemeinen Datenbanken.

Ein solcher Ansatz hat v. a. dann Aussicht auf Erfolg, wenn es – wie im Fall von J2EE – eine zentrale Institution gibt, die sich um die Entwicklung und Einhaltung der Standards kümmert. Ist dies nicht der Fall, kann es leicht zu einem Wildwuchs von Standards kommen, was die Vorteile der Standardisierung zunichte macht.

Zu diesem Zeitpunkt ist es noch zu früh, eine bestimmte Möglichkeit zu favorisieren. Alle drei Ansätze und auch eine Kombination daraus sind möglich und jeder Ansatz hat unter Umständen unter bestimmten Randbedingungen spezielle Vorteile. In dieser Arbeit werden mit der Definition des Meta-Modells, der Architektur und der Vorgehensweise die Grundlagen für alle drei

Ansätze gelegt, so dass die in der jeweiligen Situation optimale Lösung erzielt werden kann. Die zuletzt beschriebene Möglichkeit der standardisierten Schnittstellen verlangt eine genaue Beschreibung der Funktionalität der Komponenten. Dies ist zum jetzigen Zeitpunkt noch nicht gegeben, kann aber später mit Hilfe des Meta-Modells entwickelt werden.

5.2 Meta-Modellierung

5.2.1 Eigenschaften und Vorteile von Meta-Modellen

Meta-Modelle sind auf einer höheren Abstraktionsebene angesiedelt als Modelle. Meta-Modelle beschreiben, was es heißt, ein Modell zu sein. D. h. sie beschreiben die allgemeine Struktur von Modellen, deren Eigenschaften, Bestandteile, Verknüpfungen, Aufbau und Einschränkungen. Ein Meta-Modell stellt eine Menge der Konzepte und Regeln dar, die nötig sind, um spezifische Modelle in einer sogenannten „Domain of Interest“ zu erstellen. Meta-Modelle sind wie Modelle ein Versuch, die Welt zu beschreiben und zwar zu einem bestimmten Zweck (vgl. [Pid03]). Sie leisten dies in Form einer Sammlung von Bausteinen und Regeln für die eigentliche Modellierungssprache, in der Modelle erstellt werden (vgl. [HS03]). Die Modellierungssprache besteht dann aus Instanzen von Konzepten aus dem Meta-Modell. Sie liefern wie Modelle eine zusätzliche Abstraktionsebene, die den speziellen Zweck besitzt, andere Modelle zu beschreiben.

Meta-Modelle können verschiedene Einsatzzwecke besitzen (vgl. [Pid03]). Sie können z. B. zur Speicherung und zum Austausch semantischer (Meta-)Daten verwendet werden. Beispielsweise dient XMI[2] als Austauschformat von Modellen, die dem Meta-Modell der MOF[3] entsprechen, zwischen verschiedenen Software-Entwicklungsumgebungen. Außerdem können Meta-Modelle als Sprache dienen, die eine bestimmte Methodik oder einen bestimmten Prozess unterstützt. Ein Beispiel hierfür ist das UML Meta-Modell, das den Prozess der Softwareentwicklung unterstützt. Darüber hinaus können Meta-Modelle auch eingesetzt werden, um zusätzliche Semantik zu bestehender Information auszudrücken.

Das hier entwickelte Meta-Modell beschreibt, wie Modelle für Publikationsanwendungen aussehen können. Es dient also dazu, eine einheitliche, abstrakte Modellierungssprache für Publikationsanwendungen einzuführen. Es abstrahiert von den spezifischen Eigenschaften des darunter liegenden Publikationssystems, das die Basis für die Publikationsanwendung bildet.

Durch das Einziehen einer solchen Abstraktionsschicht zwischen Publikationssystem und Publikationsanwendungen wird eine Unabhängigkeit zwischen Publikationssystemen und Publikationsanwendungen erreicht. Dies kapselt die Heterogenität der einsetzbaren Publikationssysteme. Dadurch wird die Austauschbarkeit des verwendeten Publikationssystems erreicht. Man ist zur Implementierung einer Publikationsanwendungen nicht auf das Publikationssystem eines bestimmten Softwareherstellers angewiesen, sondern es können Publikationssysteme von unterschiedlichen Herstellern und mit unterschiedlichen Detailimplementierungen eingesetzt werden,

[2] XML Metadata Interchange, OMG Standard: `http://www.omg.org/technology/documents/formal/xmi.htm`

[3] MetaObject Facility, `http://www.omg.org/mof/`

um ein und dieselbe Publikationsanwendung zu realisieren. Andersherum betrachtet kann ein Publikationssystem, das das Meta-Modell versteht, beliebige Publikationsanwendungen realisieren, die mit Hilfe dieses Meta-Modells beschrieben sind.

Außerdem verbessert ein Meta-Modell wie bereits erwähnt die Modifizierbarkeit. Durch das Meta-Modell wird erreicht, dass ein Publikationssystem nicht mehr programmiert, sondern nur noch konfiguriert werden muss, um an die Anforderungen eines Szenarios angepasst zu werden. Damit wird mit der Meta-Modellierung etwas Vergleichbares wie mit deklarativer Programmierung erreicht: Es wird beschrieben, *was* gemacht werden soll, aber nicht, *wie* es erreicht werden soll. Bestehende Modelle können erweitert werden, um neuen Anforderungen gerecht zu werden. Abbildung 5.1 zeigt die Beziehung vom Meta-Modell und darauf basierenden Modellen. Hier ist auch dargestellt, dass die Modelle von einem auf dem Meta-Modell basierenden Editor erstellt werden können.

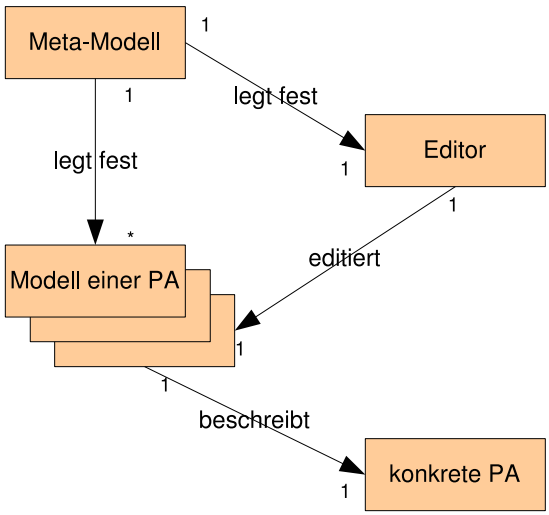

Abbildung 5.1: Beziehung von Meta-Modell zu Modellen

Ein weiterer Vorteil eines Meta-Modells für Publikationsanwendungen betrifft die sich inzwischen nahezu täglich ändernden Anforderungen durch die aktuelle Marktsituation, den globalen Wettbewerb und häufige technische Innovationen (vgl. [KP02]). Diese sich ändernden Anforderungen müssen immer wieder erfüllt werden. Durch das Meta-Modell wird die Dauer zur Anpassung an geänderte Anforderungen verkürzt und die Flexibilität erhöht. Solche Anpassungen können durch das Meta-Modell sogar dynamisch zur Laufzeit vorgenommen werden (vgl. das Reflection-Muster in [BMR+96]), da ein Publikationssystem mit Hilfe des Meta-Modells „self-aware“ gemacht wird.

Gleichzeitig dient ein solches Meta-Modell auch als Grundlage für die Aufstellung von Richtlinien für die Entwicklung von Publikationssystemen und die Anpassung bestehender Publikationssysteme an Publikationsanwendungen.

5.2.2 Meta-Modell-Hierarchien

5.2.2.1 Golden Braid Meta-Modell-Hierarchie

Eine Möglichkeit, die Abhängigkeiten zwischen den verschiedenen Abstraktionsstufen von Modellen darzustellen, ist die Golden Braid Meta-Modell-Hierarchie (vgl. [CESW06, Hof79]). Diese Hierarchie betont die Tatsache, dass Meta-Modelle, Modelle und Instanzen miteinander verbunden sind durch das fundamentale Konzept der Instantiierung.

Die Golden Braid Hierarchie betrachtet zunächst alles als Objekte. Manche Objekte können nicht weiter instantiiert werden, andere dagegen schon. Objekte, aus denen weitere Objekte instantiiert werden können, werden als Klassen bezeichnet. Nun können die Objekte, die aus Klassen instantiiert werden, selbst Klassen sein (also weiter instantiierbar sein). In diesem Fall ist die Ursprungsklasse die Meta-Klasse für alle Objekte, die aus dem Zwischenobjekt instantiiert werden.

Diese Sichtweise von Meta-Klasse, Klasse und Objekt kann beliebig weitergeführt werden. Die im folgenden beschriebene Meta-Modell-Hierarchie der OMG kann als Anwendung der Golden Braid Hierarchie über vier Ebenen betrachtbar werden.

5.2.2.2 Meta-Modell-Hierarchie der OMG

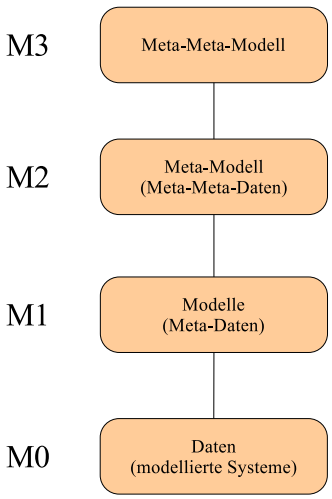

Abbildung 5.2: Meta-Modell-Hierarchie der OMG

In der Abbildung 5.2 sind die vier Stufen der Meta-Modell-Hierarchie der OMG dargestellt. Auf der untersten Stufe M0 stehen die Daten und Objekte konkreter Systeme. Auf der nächsthöheren Stufe M1 stehen die Modelle dieser konkreten Systeme. Diese Modelle sind mit Hilfe der UML beschrieben. Auf der Stufe M2 steht das Meta-Modell. Im Fall der Meta-Modell-Hierarchie der OMG ist hier beschrieben, welche Bestandteile die UML besitzt, also welche Modelle mit Hilfe

der UML darstellbar sind. Die Stufe M3 wiederum legt fest, wie diese Beschreibung auf Stufe M2 aussehen kann. Von der OMG wird hier die Meta-Object Facility (MOF) angeboten. Die UML ist mit Hilfe von MOF darstellbar.

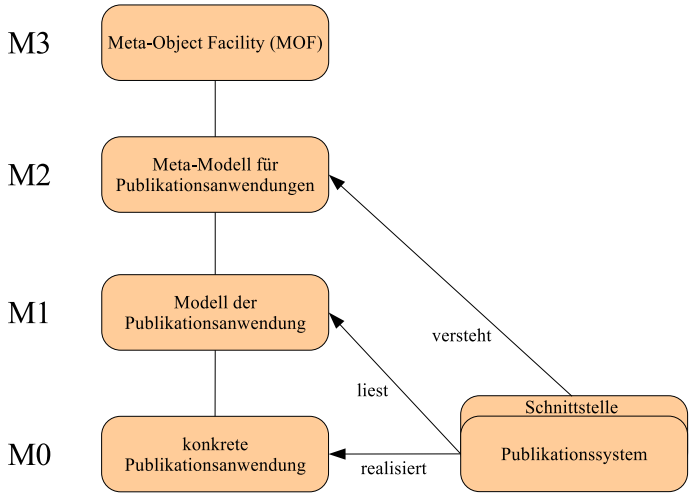

Abbildung 5.3: Meta-Modell-Hierarchie für Publikationssysteme

Die Abbildung 5.3 stellt analog dazu die Verwendung der Meta-Modell-Hierarchie der OMG in dieser Arbeit vor. Auf der Stufe M0 befindet sich die konkrete Publikationsanwendung, die von einem Publikationssystem realisiert wird. In der Stufe M1 ist diese konkrete Publikationsanwendung mit Hilfe eines Modells beschrieben. Diese Modelle sind in UML abgefasst und der Aufbau der Modelle ist in der Stufe M2 beschrieben. In dieser Stufe ist auch das zu entwickelnde Meta-Modell angesiedelt. Es erweitert dabei die UML um spezielle Konzepte für Publikationsanwendungen. Von der UML werden verschiedene Mechanismen (Stereotypen, Tags und Constraints) für die Erweiterbarkeit angeboten (vgl. [BRJ99]). Da die Erweiterungen in dieser Arbeit über vorhandene Mechanismen auf der Stufe M2 stattfinden, wird die Stufe M3 nicht berührt. Das Publikationssystem muss die Meta-Modell-Sprache verstehen, um das Modell der Publikationsanwendung lesen zu können.

5.3 Ein Meta-Modell für webbasierte Publikationsanwendungen

Zunächst stellt sich die Frage, wie man Publikationsanwendungen möglichst vollständig modelliert. Es gibt drei unterschiedliche Ebenen, die durch funktionale, statische und dynamische Modellierung erfasst werden (vgl. [Roq01]). Das Ergebnis der Betrachtung dieser Ebenen definiert eine auf UML basierende Modellierungssprache, mit deren Hilfe man Publikationsanwendungen spezifizieren kann.

Da das Meta-Modell für Publikationsanwendungen für tatsächliche Szenarien verwendbar sein und für die Modellierung realer Publikationsanwendungen eingesetzt werden soll, wird zur Ent-

wicklung des Meta-Modells ein bestehendes Szenario betrachtet. Dieses Szenario ist das Wiki-Szenario. Es ist in [Sch09] ausführlich modelliert.

Der Grund, warum speziell dieses Szenario ausgewählt wurde, liegt in der Bedeutung, die ihm von mir beigemessen wird. Meiner Meinung nach üben Wikis bereits starken Einfluss darauf aus, wie Dokumente und auch Wissen verwaltet werden und zwar sowohl für kleine Gruppen oder Einzelpersonen als auch für weltweit operierende Unternehmen. Dieser Einfluss wird noch weiter zunehmen. Außerdem nutzen Wikis einige weitergehende Möglichkeiten des elektronischen Publizierens, z. B. Dynamik und Unterstützung der einfachen Erstellung von Dokumenten.

Grundsätzlich müssten konkrete Publikationsanwendungen für alle 22 im Kapitel 3 vorgestellten Szenarien modelliert werden und in das Meta-Modell einfließen. Davon wird jedoch aus verschiedenen Gründen in dieser Arbeit abgesehen. Mit dem untersuchten Wiki-Szenario wird bereits das interessanteste und anspruchsvollste Szenario ausgewählt, das über innovative Funktionen verfügt. Daher sind die Ansprüche durch dieses Szenario bereits sehr hoch und dementsprechend liefert die Untersuchung bereits gute Ergebnisse. Darüber hinaus wurde in [Ren06] das Blogs-Szenario aus Abschnitt 3.2.1 genauer modelliert. Auch wenn die Modellierung nicht genau meinem Meta-Modell folgt, kann die Arbeit doch sehr gut für weitere Schritte herangezogen werden.

Ein guter Teil dessen, was Publikationsanwendungen ausmacht, wird bereits durch die Betrachtung von Wikis abgedeckt. Weitere Szenarien liefern mit zunehmender Wahrscheinlichkeit Ergebnisse, die nicht mehr in allen Szenarien zum Einsatz kommen werden. Die hier identifizierten Konzepte von Publikationsanwendungen dienen als wiederverwendbarer Grundstock, der im Meta-Modell festgehalten ist. Es sind darüber hinaus keine Szenarien absehbar, die sich nicht über das Meta-Modell darstellen lassen.

5.3.1 Statisches Meta-Modell

Das statische Meta-Modell zeigt die Schlüsselabstraktionen aus der Logischen Sicht des 4+1-Sichten Modells (siehe Abschnitt 4.1.4.1) und zerfällt in zwei Teile:

- ein Meta-Modell für Dokumente innerhalb von Publikationsanwendungen.
- ein Meta-Modell für Publikationsanwendungen, das die zusätzlich zu Dokumenten verwendeten Datenstrukturen beschreibt und in dem das Meta-Modell für Dokumente dementsprechend wieder auftaucht.

In der Abbildung 5.4 ist das Meta-Modell für strukturierte Dokumente – dem zentralen Artefakt in Publikationsanwendungen – dargestellt. Dieses Meta-Modell hat als zentralen Ansatzpunkt die Meta-Klasse Document. Jedes Document besitzt eine Wurzel, die vom Typ StructuralElement ist. Jedem StructuralElement können beliebig viele Attribute-Objekte zugeordnet sein und sie dienen als Basisklasse für spezialisierte Strukturelemente wie z. B. Verknüpfungen. Da Documents selbst auch vom Typ StructuralElement sind, können Documents auch in andere Documents geschachtelt werden.

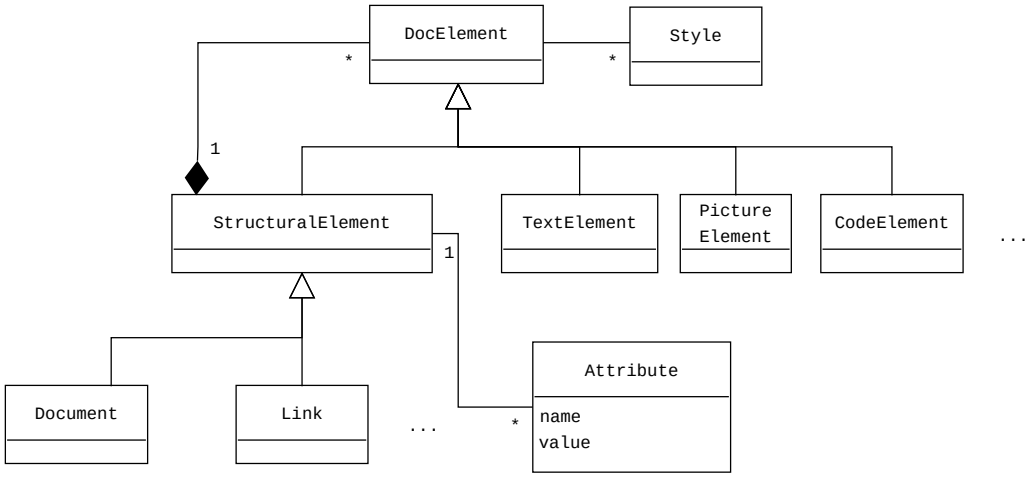

Abbildung 5.4: Meta-Modell für Dokumente

StructuralElement ist auch der zentrale Bestandteil im Entwurfsmuster der Komposition, das im Meta-Modell in Form der Beziehungen zwischen DocElement, StructuralElement, TextElement, PictureElement, CodeElement usw. vorkommt. PictureElement ist ein Beispiel für multimediale Dokumentenbestandteile, CodeElemente können Dokumenten eigene Dynamik verleihen. Ebenso wie TextElement sind dies Dokumentenbestandteile, die nicht weiter untergliedert werden können. Dagegen kann ein StructuralElement selbst wieder aus Textelementen etc. oder weiteren StructuralElements bestehen. Damit lässt sich eine beliebig tief geschachtelte Struktur innerhalb eines Dokuments aufbauen.

Präsentationsvorschriften sind in Form von Style-Objekten direkt der Oberklasse DocElement zugeordnet. Auf diese Weise können sie sowohl an Objekte vom Typ StructuralElement als auch für Objekte vom Typ TextElement gebunden werden. Gleichzeitig wird damit auch die Vererbung von Präsentationsvorschriften von übergeordneten Strukturelementen auf enthaltene Unterelemente unterstützt.

Die Erweiterbarkeit des Meta-Modells für Dokumente ist auf zwei Arten gegeben: Zum einen können unterhalb von DocElement weitere Kindelemente eingefügt werden, die wie TextElemente keine weiteren Elemente enthalten können. Zum anderen können weitere strukturelle Elemente unterhalb von StructuralElement eingefügt werden.

Nach der Beschreibung des Meta-Modells für Dokumente wird nun das Meta-Modell für Publikationsanwendungen vorgestellt, das in der Abbildung 5.5 dargestellt ist.

Der zentrale Ansatzpunkt im Meta-Modell ist das Publikationssystem. Die wesentliche Aufgabe eines Publikationssystems besteht in der anwendungsgerechten Verwaltung und Verarbeitung einer Menge von Ressourcen (Meta-Klasse Resource), zu denen neben Dokumenten auch Benutzerkonten (Meta-Klasse Account) und Stilvorlagen gehören. Dokumente sind dabei also eine spezialisierte Art von Ressourcen. Eine Ressource ist ein Datenobjekt, das mit Hilfe der Datenpersistenzkomponente dauerhaft verwaltet wird und das von außen über Requests zugreifbar

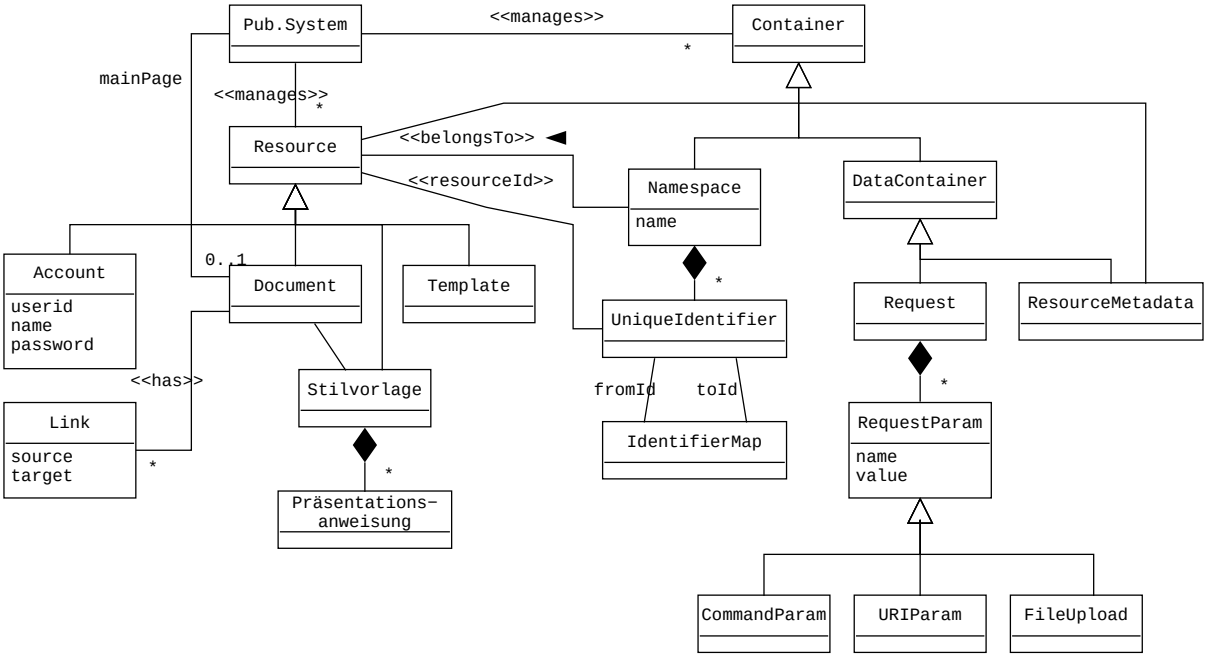

Abbildung 5.5: Meta-Modell für Publikationsanwendungen

ist. Die Meta-Klasse ResourceMetadata stellt einen Datencontainer dar, der beschreibende Daten über eine Ressource enthält.

In dieser Menge von Dokumenten kann es ein Dokument geben, das einen besonderen Status einnimmt, da es als zentrales Hauptdokument dient, das bei Benutzeranfragen bereitgestellt wird, in denen kein anderes Dokument explizit angefordert wurde. Dokumenten ist eine Stilvorlage zugeordnet. Eine Stilvorlage fasst eine Menge von Präsentationsanweisungen zusammen und legt die Präsentation des Dokuments fest. Die Meta-Klasse Link verknüpft zwei Dokumente miteinander.

Eine wichtige Möglichkeit, die Menge von Dokumenten innerhalb eines Publikationssystems in unterschiedliche Teilmengen zu zerlegen, ist durch das Konzept der Namespaces gegeben. Ein Namespace ist ein logischer Container und besitzt einen eindeutigen Namen. Jedes Dokument kann einem bestimmten Namespace zugeordnet sein, innerhalb dessen der UniqueIdentifier des Dokuments eindeutig ist. Ein Dokument ist also über den Namespace und UniqueIdentifier innerhalb des Publikationssystems eindeutig identifizierbar.

Das Konzept der IdentifierMap ermöglicht es, Identifikatoren aus verschiedenen Namespaces aufeinander abzubilden. Damit ist es beispielsweise möglich, einen Identifikator unverändert zu lassen (z. B. weil er extern für den Zugriff auf ein Dokument durch den Benutzer verwendet wird), aber intern die Zuordnung zum Zieldokument beliebig verändern zu können.

Ein DataContainer ist ebenso wie ein Namespace auch ein Container, unterscheidet sich davon aber dadurch, dass nun Nutzdaten zusammengefasst werden und nicht wie bei Namespaces nur

eine logische Zuordnung zu einem Namensraum gegeben ist.

Dementsprechend sind Unterklassen von DataContainer dazu gedacht, Daten von Benutzeranfragen oder Metadaten zu Ressourcen zusammenzufassen. Benutzeranfragedaten werden durch die Meta-Klasse Request beschrieben. Sie besitzt eine Menge von RequestParametern, die sich weiter unterscheiden lassen in CommandParameter, die Befehle darstellen, URIParameter, die eine URI-Adresse beeinhalten und FileUpload-Parameter, die beim Hochladen von Binärdaten in das Publikationssystem benutzt werden.

5.3.2 Funktionales und dynamisches Meta-Modell

Das funktionale Modell muss die funktionalen Anforderungen erfüllen und stellt damit ebenso wie das statische Meta-Modell die logische Sicht des 4+1-Sichten Modells dar. Das dynamische Modell zeigt das Zusammenwirken der verschiedenen Komponenten und fällt somit sowohl unter die Entwicklungs- als auch unter die Szenariensicht.

Zur Modellierung der dynamischen Abläufe innerhalb einer Publikationsanwendung stellt die UML Sequenzdiagramme und Activity Diagramme zur Verfügung. Activity Diagramme wurden hier bevorzugt, da es mit ihnen im Unterschied zu Sequenzdiagrammen auch möglich ist, parallele Abläufe zu modellieren. (vgl. [Fow03b]).

Die funktionale Modellierung ist von den dynamischen Modellen abgedeckt: Aufgrund der Tatsache, dass in dieser Arbeit webbasierte Softwaresysteme modelliert werden, tauchen die Use Cases (die normalerweise zur funktionalen Modellierung verwendet werden, vgl. [Coc01]) in den Activity Diagrammen in Form von Aktionen auf, die mit der generischen «Request»-Aktion gekennzeichnet sind.

Innerhalb der Diagramme kommen generische Aktionen zum Einsatz. Diese Aktionen können über Parameter an ihren konkreten Einsatzzweck anpassbar sein. Die Aktionen in Activity Diagrammen konkreter Szenarien verwenden diese generischen Aktionen in Form von UML-Stereotypen, um hierüber dem Publikationssystem mitzuteilen, welche Bedeutung die Aktion besitzt. Sie dienen somit als Bausteine, aus denen die dynamischen Abläufe eines Szenarios zusammengestellt werden.

Im Folgenden werden die bei der Modellierung des Wiki-Szenarios gefundenen generischen Aktionen genauer beschrieben und verschiedenen Komponenten (hinter dem Namen der Aktion in Klammern angegeben) zugeordnet.

Request: (Dispatcher)
: Durch diesen Aktions-Stereotypen wird gekennzeichnet, dass die betreffende Aktion durch einen Browser-Request initiiert wird und als Parameter eine Request-Datenstruktur besitzt. Der technisch vom Web-Server abhängige Browser-Request wird vom Webhandler in einen vom Web-Server unabhängigen Request umgewandelt und an die Dispatcher-Komponente weitergereicht.

Extract external ResourceId: (Dispatcher)
: Diese Aktion extrahiert die Daten, die zu einer ResourceId gehören, aus dem Request und stellt sie anderen Aktionen zur Verfügung.

Parse external ResourceId: (Dispatcher)
Diese Aktion zerlegt die Daten, die zuvor extrahiert wurden, so, dass ResourceName und Namespace getrennt werden können.

Extract ResourceName: (Dispatcher)
Diese Aktion extrahiert den Wert für den ResourceName.

Extract Namespace: (Dispatcher)
Diese Aktion extrahiert den Namen des Namespace.

Deliver Document: (Dispatcher)
Diese Aktion sendet das erstellte Dokument an den Browser, vom dem der Request empfangen wurde.

ExtractRequestParameter: (Dispatcher)
Diese Aktion extrahiert einen bestimmten Parameter aus dem Request, der z. B. auch per HTTP-POST geliefert worden sein kann.

UpdateIndex: (Multimedia)
Diese Aktion aktualisiert den Index, in dem die Multimedia-Komponente Metadaten zu Multimedia-Dateien verwaltet. Metadaten sind beispielsweise der Typ der Datei, Größe von Bildern, Uhrzeit und Datum des Hochladens und ein vom Benutzer angegebener Kommentar. Parameter sind ein UniqueIdentifier, der Name und der Typ der Datei.

StoreBinary: (Datenpersistenz)
Diese Aktion speichert beliebige binäre Daten persistent. Als Parameter werden ein UniqueIdentifier, Namespace und die Daten erwartet.

CreateNewId: (Datenpersistenz)
Diese Aktion liefert einen neuen eindeutigen Identifikator zurück. Als Parameter ist der Namespace anzugeben, innerhalb dessen der Identifikator eindeutig ist.

GetMultimediaData: (Multimedia)
Diese Aktion lädt die von der Multimedia-Komponente zu einer Multimedia-Datei verwalteten Metadaten. Als Parameter ist ein UniqueIdentifier erforderlich.

GetMetadata: (Datenpersistenz)
Diese Aktion lädt die zu einer Ressource gespeicherten Metadaten. Als Parameter wird ein UniqueIdentifier übergeben.

SetRequestParameter: (Dispatcher)
Diese Aktion setzt einen Parameter in einem Request auf einen bestimmten Wert bzw. fügt ihn dem Request hinzu, wenn er im Request noch nicht vorhanden ist. Dies kann z. B. für die Realisierung von Default-Werten für Request-Parameter verwendet werden.

Search: (Suchfunktion)
Diese Aktion durchsucht den Dokumentenbestand nach den angegebenen Suchbegriffen. Als Parameter wird neben den Suchbegriffen der Namespace angegeben, der durchsucht werden soll. Das Ergebnis ist eine Liste von eindeutigen Identifikatoren.

Search Index: (Datenpersistenz)
Die Suchfunktion kann zur Verbesserung der Performanz einen Suchindex verwalten, auf

den sie zur Suche nach Suchbegriffen zugreift. Die Verwaltung eines solchen Index wird von der Datenpersistenz-Komponente geleistet.

ListResources: (Datenpersistenz)
Diese Aktion liefert eine Liste mit den eindeutigen Identifikatoren aller verwalteten Ressourcen innerhalb eines bestimmten Namespace, der als Parameter angegeben wird.

ListLinkingDocuments: (Navigation)
Diese Aktion liefert eine Liste mit den eindeutigen Identifikatoren aller Dokumente, die eine Verknüpfung besitzen, die auf das angegebene Dokument verweist.

ListResources: (Datenpersistenz)
Diese Aktion liefert eine Liste mit den eindeutigen Identifikatoren aller Ressourcen, die ein angegebenes Prädikat erfüllen. Ein solches Prädikat kann z. B. vergleichen, ob eine Ressource jünger ist als ein bestimmtes Vergleichsdatum.

FindDeadLinks: (Navigation)
Diese Aktion sucht innerhalb des angegebenen Namespace nach Verknüpfungen, deren Ziel nicht existiert. Als Ergebnis wird eine Liste dieser Verknüpfungen geliefert.

ListDocumentsWithoutReferers: (Navigation)
Diese Aktion liefert eine Liste mit eindeutigen Identifikatoren der Dokumente, die innerhalb des angegebenen Namespace kein Ziel einer Verknüpfung von einem anderen Dokument sind.

GetRandomId: (Datenpersistenz)
Diese Aktion liefert aus allen eindeutigen Identifikatoren innerhalb des Namespace einen zufällig ausgewählten Identifikator zurück.

GetStatistics: (Datenpersistenz)
Diese Aktion liefert statistische Daten über die in der Datenhaltung gespeicherten Ressourcen.

CreateAccount: (Mehrbenutzer)
Diese Aktion legt einen neuen Benutzer in der Mehrbenutzerverwaltung an. Als Parameter werden eine eindeutige UserId und weitere Daten über den Benutzer erwartet.

LoadUserData: (Mehrbenutzer)
Diese Aktion lädt Daten über den Benutzer, der über eine UserId als Parameter identifiziert wird, aus der Datenhaltung.

StoreUserData: (Mehrbenutzer)
Diese Aktion speichert die Daten eines Benutzers in der Datenhaltung.

SaveUserSpecific: (Mehrbenutzer)
Diese Aktion speichert Daten, die verknüpft sind mit einem bestimmten Benutzer. Die Angabe eines Namespace-Parameters ermöglicht es, unterschiedliche Gruppen von Daten zu speichern. Zum Beispiel kann so eine benutzerspezifische Watchlist verwaltet werden.

LoadUserSpecific: (Mehrbenutzer)
Diese Aktion lädt Daten, die verknüpft sind mit einem bestimmten Benutzer. Als Para-

meter werden Identifikatoren für den Benutzer und den Namensraum der gespeicherten Daten erwartet.

DeleteUserSpecific: (Mehrbenutzer)
Diese Aktion löscht Daten, die verknüpft sind mit einem bestimmten Benutzer. Als Parameter werden Identifikatoren für den Benutzer, die Daten und den Namensraum der gespeicherten Daten erwartet.

FilterResourceList: (Mehrbenutzer)
Diese Aktion filtert aus der Liste diejenigen Einträge aus, die das angegebene Prädikat nicht erfüllen.

FillInData: (Automatische Bearbeitung)
Diese Aktion füllt die angegebenen Daten in eine ebenfalls als Parameter angegebene Datenstruktur.

LookupUserId: (Benutzerauthentifizierung)
Diese Aktion extrahiert eine UserId aus dem als Parameter übergebenen Request.

VerifyLogin: (Benutzerauthentifizierung)
Diese Aktion prüft, ob der angegebene Benutzer existiert und das angegebene Passwort hat.

LoadDataContainers: (Datenpersistenz)
Diese Aktion lädt eine Liste von DataContainer-Objekten aus der Datenhaltung. Als Parameter wird ein eindeutiger Identifikator übergeben.

StoreResourceMetadata: (Datenpersistenz)
Diese Aktion speichert das angegebene Metadaten-Objekt in der Datenhaltung. Als weiterer Parameter wird ein eindeutiger Identifikator der Ressource angegeben, zu der die Metadaten gehören.

StoreResource: (Datenpersistenz)
Diese Aktion speichert die angegebene Ressource in der Datenhaltung. Als Parameter ist zusätzlich zur Ressource ein eindeutiger Identifikator angegeben.

DeleteResource: (Datenpersistenz)
Diese Aktion löscht eine Ressource aus der Datenhaltung. Als Parameter ist der eindeutige Identifikator der Ressource angegeben.

LoadResource: (Datenpersistenz)
Diese Aktion lädt die angegebene Ressource aus der Datenhaltung. Als Parameter wird ein eindeutiger Identifikator der gewünschten Ressource angegeben.

TransformDocument: (Automatische Bearbeitung)
Diese Aktion dient dazu, ein Dokument einer generischen Transformation zu unterziehen. Als Parameter sind das Dokument und die Transformationsvorschriften erforderlich.

Validate: (Validierung)
Diese Aktion dient dazu, die Struktur eines Dokuments auf ihre Konformität zu den ebenfalls angegebenen Strukturvorschriften zu überprüfen.

AddLatestVersion: (Versionierung)
Diese Aktion fügt eine neue Version einer Ressource zur Versionshistorie hinzu. Als Parameter sind die eindeutigen Identifikatoren für die alte und die neue Version der Ressource erforderlich. Die Ressource selbst wird von der Aktion „StoreResource“ in der Datenhaltung abgelegt.

ListAllVersions: (Versionierung)
Diese Aktion liefert eine Liste mit den eindeutigen Identifikatoren aller Versionen einer bestimmten Ressource. Die Ressource wird dabei in Form eines eindeutigen Identifikators als Parameter identifiziert.

UpdateIdMapping: (IdMapper)
Diese Aktion aktualisiert die Verbindung zwischen zwei eindeutigen Identifikatoren. Dabei kann nur ein einzelner Identifikator der Verbindung auf einmal verändert werden, der andere Identifikator dient zur Festlegung, welcher Mapping-Eintrag betroffen ist. D. h. neben den beiden Identifikatoren gibt ein dritter Parameter an, welche Seite der Zuordnung geändert werden soll.

Neben diesen generischen Aktionen gibt es in den Activity Diagrammen auch sogenannte Verzweigungen, die den dynamischen Ablauf abhängig von einer Bedingung in unterschiedliche Zweige aufteilen. Die generischen Verzweigungen, die identifiziert wurden, sind im Folgenden beschrieben.

ResourceExists: (Dispatcher)
Diese Verzweigung überprüft, ob die angegebene Ressource existiert. Die Ressource wird durch einen UniqueIdentifier identifiziert.

ResourceIsProtected: (Dispatcher)
Diese Verzweigung überprüft, ob die angegebene Ressource durch Zugriffsrechte eingeschränkt ist oder ob der Zugriff immer erlaubt ist. Die Ressource wird durch einen UniqueIdentifier identifiziert.

CheckAuthorization: (Rechteverwaltung)
Diese Verzweigung prüft, ob ein Zugriff auf die angegebene Ressource erlaubt ist. Als Parameter sind eine UserId, ein UniqueIdentifier für die Ressource und der gewünschte Zugriff anzugeben.

ResourceIsCached: (Datenpersistenz)
Diese Verzweigung prüft, ob die angegebene Ressource in einer aktuellen Version im Cache gespeichert ist.

5.4 Vorgehen bei der Modellierung von Publikationsanwendungen

In diesem Abschnitt wird zunächst das Vorgehen bei der Modellierung von Publikationsanwendungen und der Erweiterung bestehender Modelle beschrieben. Anschließend wird dieses Vorgehen anhand des Wiki-Szenarios illustriert.

Die Modellierung von Publikationsanwendungen zerfällt in die folgenden Schritte:

1. Als Erstes werden die funktionalen Aspekte der Publikationsanwendung modelliert. Dazu werden die Akteure bestimmt, die mit dem System interagieren.
2. Anschließend werden die Anwendungsfälle der Akteure festgehalten und zunächst textuell beschrieben.
3. Nach der textuellen Beschreibung der Anwendungsfälle werden die dynamischen Aspekte modelliert, indem zu jedem Anwendungsfall ein oder mehrere Activity Diagramme erstellt werden. Dabei werden die vorhandenen generischen Aktionen verwendet oder neue Aktionen hinzugefügt, falls diese im Meta-Modell noch nicht vorhanden sind. Die Aktionen werden den jeweiligen Komponenten zugeordnet.
4. Nach der dynamischen Modellierung werden die identifizierten Konzepte (Klassen und Beziehungen) der Publikationsanwendung im Domänenmodell beschrieben. Dabei wird auf die vom Meta-Modell zur Verfügung gestellten Konzepte zurückgegriffen, die in Form von Stereotypen referenziert werden.
5. Mit diesen Ergebnissen kann das Meta-Modell entsprechend erweitert werden bzw. das erstellte Modell kann mit dem Meta-Modell abgeglichen werden.

Die Qualifikation, die erforderlich ist, um eine solche Modellierung vorzunehmen, ist vergleichbar mit der nötigen Qualifikation für die normale Software-Entwicklung, wobei hier keine direkte Programmierung des Publikationssystems mehr erforderlich ist, also auch kein Detail-Know-How über die Eigenschaften eines speziellen Systems nötig ist. Außerdem muss unterschieden werden zwischen den Tätigkeiten bei der Modellierung neuer Szenarien und der Anpassung vorhandener Szenarien an geänderte Anforderungen. Im letzteren Fall fällt die geforderte Qualifikation geringer aus, außerdem sind Werkzeuge entwickelbar, die den Benutzer (das könnte in diesem Fall dann auch durchaus ein Autor sein) bei der Anpassung unterstützen und so die Anforderungen an die Qualifikation weiter reduzieren.

Gleichzeitig sollte die Qualität einer solchermaßen realisierten Publikationsanwendung höher liegen als beim konventionellen Vorgehen (bei dem oft ein komplettes Publikationssystem neu entwickelt wird). Dies liegt am Einsatz erprobter Techniken, die im Laufe der Zeit weiter verbessert und an die speziellen Anforderungen von Publikationssystemen angepasst werden können. Daher erscheint es durchaus lohnenswert, über diese Arbeit hinaus solche Werkzeuge zu entwickeln und auf den Markt zu bringen.

5.4.1 Statisches Modell

Basierend auf dem oben dargestellten Meta-Modell für Publikationsanwendungen wird in den folgenden Abschnitten ein Modell vorgestellt, das das Szenario der Wikis beschreibt. Dieses Modell ist aufgeteilt in ein statisches und ein dynamisches Modell. Zunächst werden die Diagramme beschrieben, aus denen das statische Modell besteht. Die identifizierten Klassen stützen sich dabei auf die vom Meta-Modell zur Verfügung gestellten Basis-Klassen, die im Diagramm über Stereotypen angegeben sind.

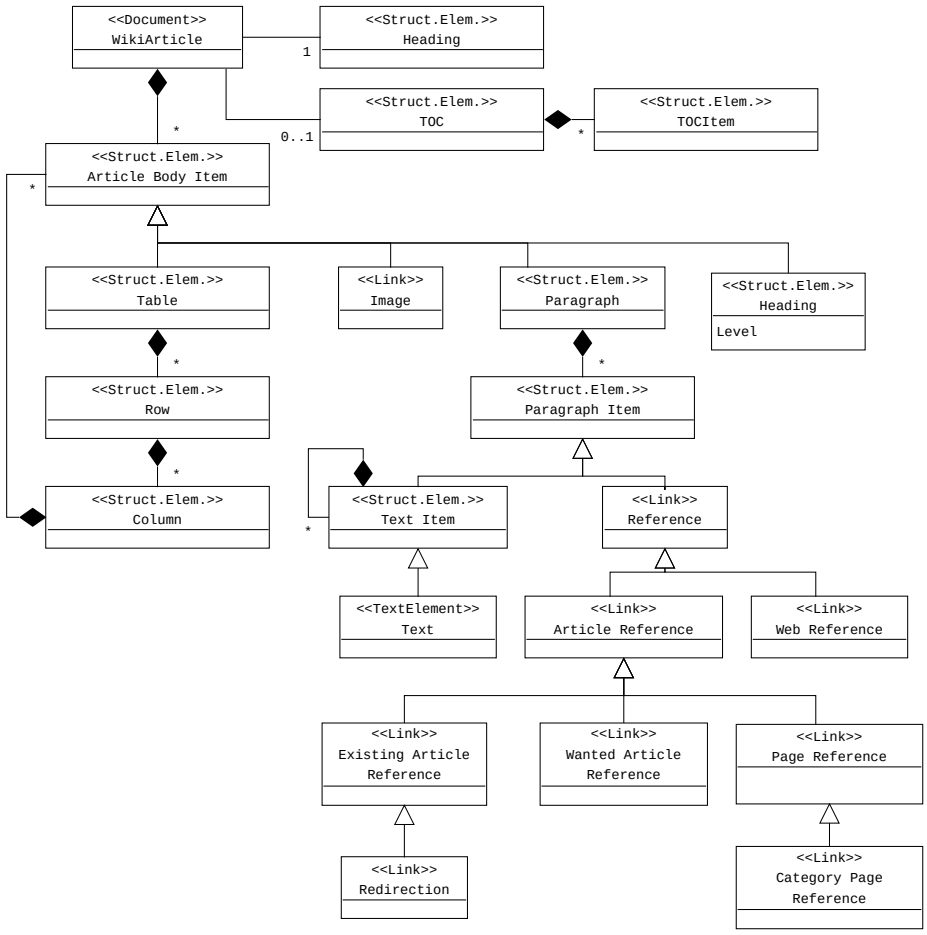

Abbildung 5.6: Wiki Dokumentenmodell

Als Erstes ist in der Abbildung 5.6 der Aufbau der im Wiki-Szenario verwalteten Dokumente beschrieben. Die Dokumente werden im Wiki-Szenario WikiArticle genannt und entsprechen diesem Dokumentenmodell. Das Dokumentenmodell wiederum entspricht dem Meta-Modell für strukturierte Dokumente (siehe Abbildung 5.4 auf Seite 142). Dies bedeutet, dass seine Bestandteile und die möglichen Beziehungen zwischen diesen Bestandteilen im Meta-Modell festgelegt sind. So ist die Klasse WikiArticle vom Typ Document und besteht dementsprechend aus einer Reihe von weiteren StructuralElements, wobei im Dokumentenmodell für Wikis diese Beziehung genauer spezifiziert wird: Ein WikiArticle besteht so aus genau einem Heading, maximal einem TOC (Table of Contents) und einer beliebigen Anzahl an ArticleBodyItems.

Die ArticleBodyItems werden durch Ableitung spezialisiert. Dadurch werden die weiteren, speziellen Bestandteile von WikiArticles beschrieben, die jeweils Instanzen von Klassen des Meta-Modells sind und abhängig davon eine weiter unterteilte Struktur besitzen können.

Die Klassen im Klassendiagramm lassen sich in drei wesentliche Gruppen aufteilen: Die erste Gruppe besteht nur aus der einen Klasse „WikiArticle“ mit dem Stereotypen „Document“. Sie

bildet den Ausgangspunkt für die weitere Strukturierung des Dokuments. Zur zweiten Gruppe gehören die Klassen, die als Stereotypen „StructuralElement“ besitzen. Sie bilden eine Art Mittelschicht und dienen zur feineren Strukturierbarkeit der Dokumente. Die dritte Gruppe schließlich besteht aus den Klassen mit den Stereotypen „Link“ und „Textelement“. Sie bilden die Blätter in dem Baum, der von einem Dokument darstellt wird.

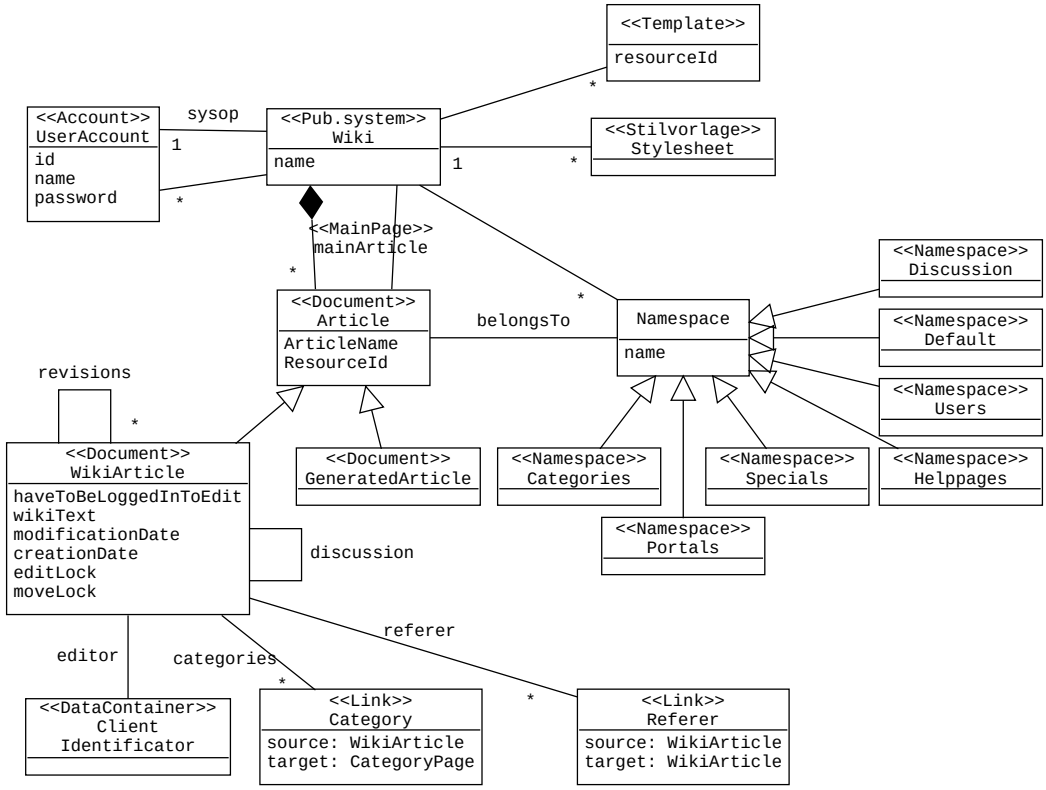

Abbildung 5.7: Wiki Domänenmodell

In der Abbildung 5.7 ist nun der wesentliche Teil des Domänenmodells von Wiki-Systemen dargestellt. Es folgen noch weitere Diagramme, die zusätzliche Teile des Domänenmodells enthalten, die aus Gründen der Übersichtlichkeit nicht in der Abbildung 5.7 enthalten sind.

Der zentrale Ansatzpunkt ist die Klasse „Wiki“, die mit dem Stereotypen „Pub.system“ gekennzeichnet ist. Dem Wiki sind UserAccounts zugeordnet, von denen einer die Sonderrolle des Systemoperators einnimmt, der das Wiki verwaltet und mehr Rechte besitzt, als normale Benutzer. Dem Wiki sind außerdem Stilvorlagen und Templates zugeordnet, die die Art der Präsentation der verwalteten Dokumente festlegen. Die verwalteten Dokumente teilen sich in zwei Gruppen auf, die beide Unterklassen zur Klasse „Article“ sind. Die erste Gruppe, „WikiArticle“, steht für die von Benutzern angelegten und bearbeiteten Artikeln. Die zweite Gruppe, „GeneratedArticle“, steht für jene Dokumente, die vom Wiki-System automatisch generiert werden. Das sind z. B. alphabetische Listen aller verwalteten WikiArticle, die von Benutzern angelegt wurden. WikiArticle Objekte besitzen eine Versionshistorie (verfügbar über „revisions“), haben jeweils

einen Bearbeiter, einen zum Artikel gehörenden Diskussionsartikel und können verschiedenen Kategorien angehören. Jeder Artikel gehört zu einem Namespace, wodurch verschiedene, logisch in sich geschlossene Gruppen gebildet werden.

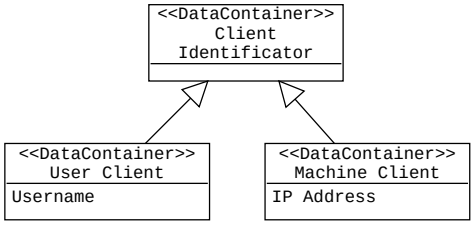

Abbildung 5.8: Wiki Domänenmodell, Client Identifier

In der Abbildung 5.8 sind die Klassen modelliert, die in einem Wiki-System Daten über die Client-Seite verwalten. Als Basis dient die Meta-Klasse „DataContainer“. Konkret wird unterschieden zwischen Clients, die als Benutzer bekannt sind (weil sie als Benutzer eingeloggt sind) und den anderen Fällen, in denen nur der Netzwerkname des Client-Rechners zur Verfügung steht.

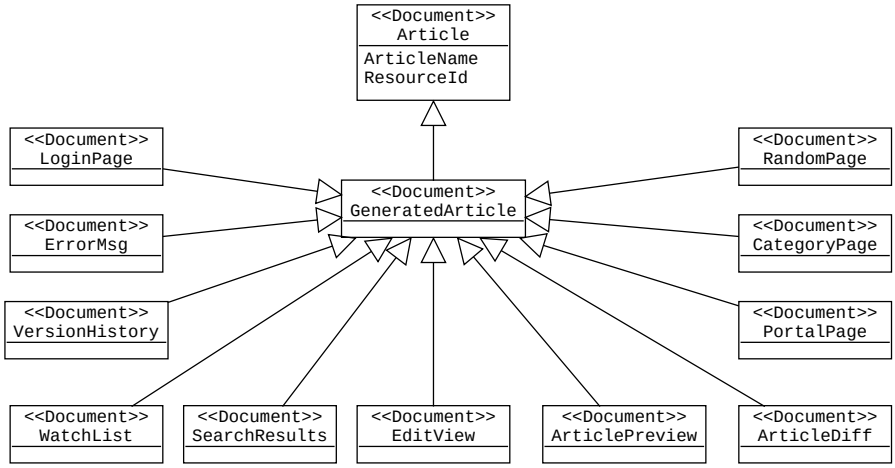

Abbildung 5.9: Wiki Domänenmodell, Generated Article

In der Abbildung 5.9 wird die Klasse der „GeneratedArticle“ weiter unterteilt, was in der Abbildung 5.7 zur Verbesserung der Übersichtlichkeit weggelassen wurde.

In der Abbildung 5.10 wird eine besondere Komponente beschrieben, die nicht naheliegend war: die Komponente „IdMapper“. IdMapper hat die Aufgabe, eine Abbildung zwischen extern ansprechbaren Ressourcennamen und intern existierenden Ressourcen (und damit Dokumenten) zu leisten. Die eigentliche Abbildung ist in einer IdentifierMap gespeichert. Durch die Abbildung kann bei Beibehaltung der externen Ressourcennamen flexibel auf ein anderes Dokument verwiesen werden, wann immer dies erforderlich ist.

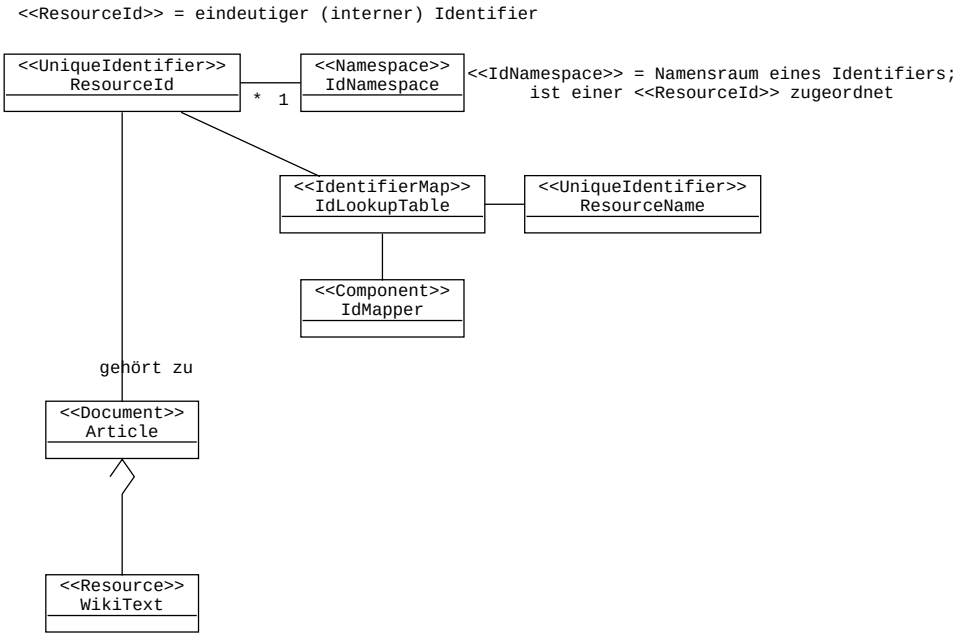

Abbildung 5.10: Wiki Domänenmodell, IdMapper

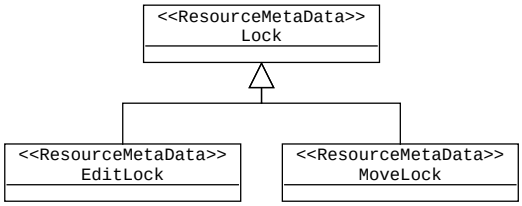

Abbildung 5.11: Wiki Domänenmodell, Locks

In der Abbildung 5.11 sind die Arten von Sperren modelliert, die ein Wiki-System üblicherweise kennt. Die Modellierung verwendet den Stereotypen „ResourceMetaData“, da Sperren als Metadaten aufgefasst werden können, die zu einer Ressource gehören (ein Dokument ist auch eine Ressource).

In der Abbildung 5.12 ist die Datenstruktur modelliert, die beim Anmelden von Benutzern am Wiki-System verwendet wird. Eine DataContainer-Klasse fasst die KeyValueParameter zusammen, die den Benutzernamen und das Passwort enthalten.

In der Abbildung 5.13 sind die Klassen dargestellt, die in Requests verwendet werden. ArticleName wird extern verwendet, um in einem Request einen Artikel zu identifizieren. Intern wird nach Ausführung der Aktion LookupArticleId eine ArticleId-Struktur verwendet. Eine ArticleId ist ein UniqueIdentifier und verbindet einen eindeutigen Identifikator mit einem Namensraum innerhalb dessen der Identifikator eindeutig ist.

In der Abbildung 5.14 sind die verschiedenen Typen von Ressourcen dargestellt, die von Wiki-

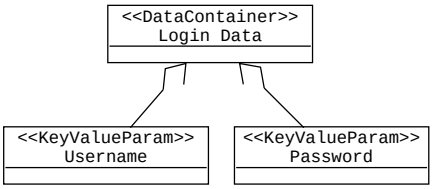

Abbildung 5.12: Wiki Domänenmodell, Login Data

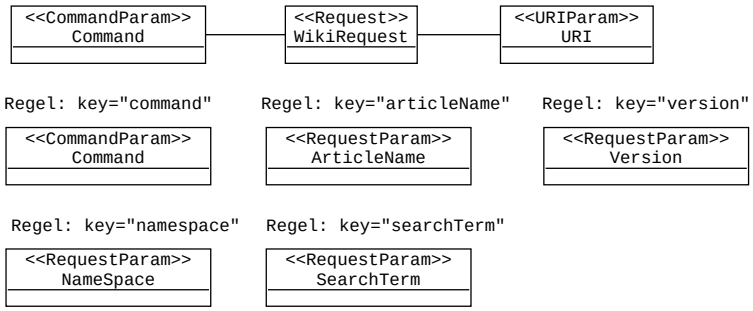

Abbildung 5.13: Wiki Domänenmodell, Request

Systemen verwaltet werden. Es wird dabei unterschieden zwischen (verwalteten oder generierten) Dokumenten (Klasse Document), Stylesheets und (von Benutzern) hochgeladenen Dateien wie Bildern, Video- und Audio-Dateien. Den Datei-Ressourcen sind dabei Metadaten in Form der FileMetaData-Klasse zugeordnet.

5.4.2 Funktionales und dynamisches Modell

Das dynamische Modell ist ausführlich in [Sch09] beschrieben. An dieser Stelle werden daher nur einzelne, exemplarisch ausgewählte Activity Diagramme beschrieben.

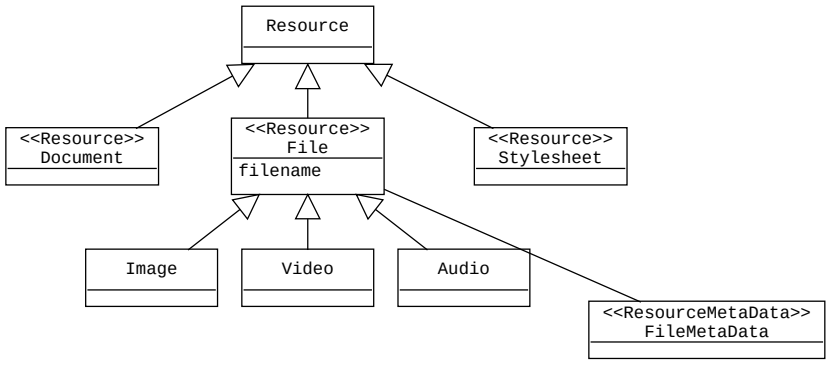

Abbildung 5.14: Wiki Domänenmodell, Resource

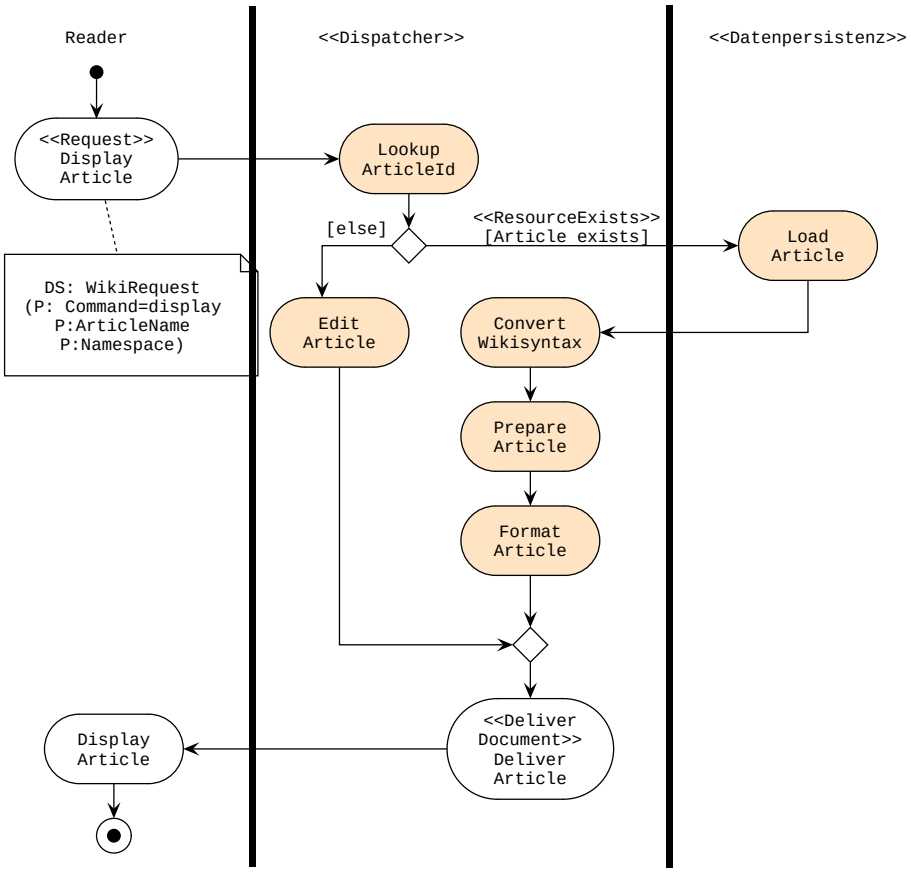

Abbildung 5.15: Wiki, dynamisches Modell, Display Article

Die Abbildung 5.15 zeigt das Activity Diagramm für den Use Case „Display Article“. Die Beziehung zur funktionalen Modellierung ist sichtbar in der ersten Aktion des Diagramms, die mit dem Stereotypen „Request“ annotiert ist. Dieser Request hat als Parameter eine Datenstruktur vom Typ „WikiRequest“, die weitere Parameter wie den Befehlsidentifikator, den gewünschten Artikelnamen und den Namespace zusammenfasst.

Dieser Request wird zunächst an die Aktion „Lookup ArticleId“ weitergereicht. Dabei handelt es sich um eine zusammengesetzte Aktion, die selbst in weitere Aktionen aufgeteilt und in einem getrennten Activity Diagramm beschrieben ist. Solche weiter unterteilten Aktionen sind in den Diagrammen farblich gekennzeichnet.

Die Aktion “Lookup ArticleId“ ist in der Abbildung 5.16 im Detail dargestellt. Die Aufgabe der Aktion besteht darin, aus der Request-Datenstruktur die ResourceId und Namespace zu extrahieren, die einen Artikel eindeutig identifiziert und ihn auf einen eindeutigen internen Identifikator abzubilden.

Anschließend wird mit Hilfe der Verzweigung „ResourceExists“ überprüft, ob der gewünschte Artikel bereits existiert. Falls das nicht der Fall ist, wird zur Aktion „Edit Article“ (Abbildung

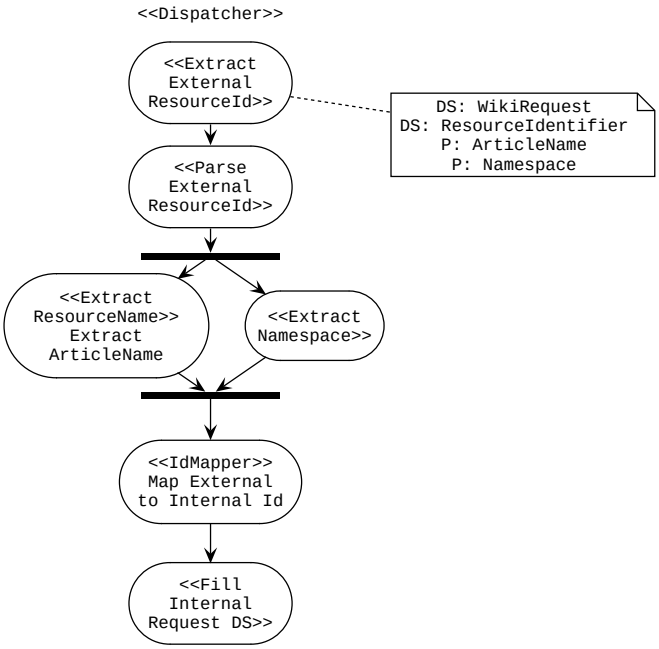

Abbildung 5.16: Wiki, dynamisches Modell, Lookup ArticleId

5.18) verzweigt, die dafür zuständig ist, dem Benutzer eine Editierungsansicht zu liefern, mit deren Hilfe der noch nicht existierende Artikel angelegt werden kann. Falls der Artikel schon existiert, wird er über die Aktion „Load Article“, die in der Abbildung 5.17 gezeigt ist, geladen. Diese Aktion überprüft zunächst, ob der Artikel im Cache zur Verfügung steht und lädt ihn entsprechend entweder aus dem Cache oder aus der Datenhaltung.

Die Aktion „Convert Wikisyntax“ (siehe Abbildung 5.19) ist spezifisch für das Wiki-Szenario und wandelt die spezielle Wikisyntax im Artikel in allgemeine Konstrukte um (z. B. in normale XML-Elemente). Diese Aktion ruft eine spezielle Wiki-Komponente auf, in der diese Funktionalität, die nur für Wikis relevant ist, von den restlichen, allgemeinen Komponenten abgetrennt ist.

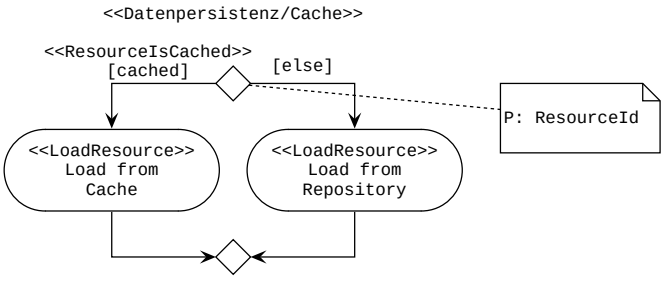

Abbildung 5.17: Wiki, dynamisches Modell, Load Article

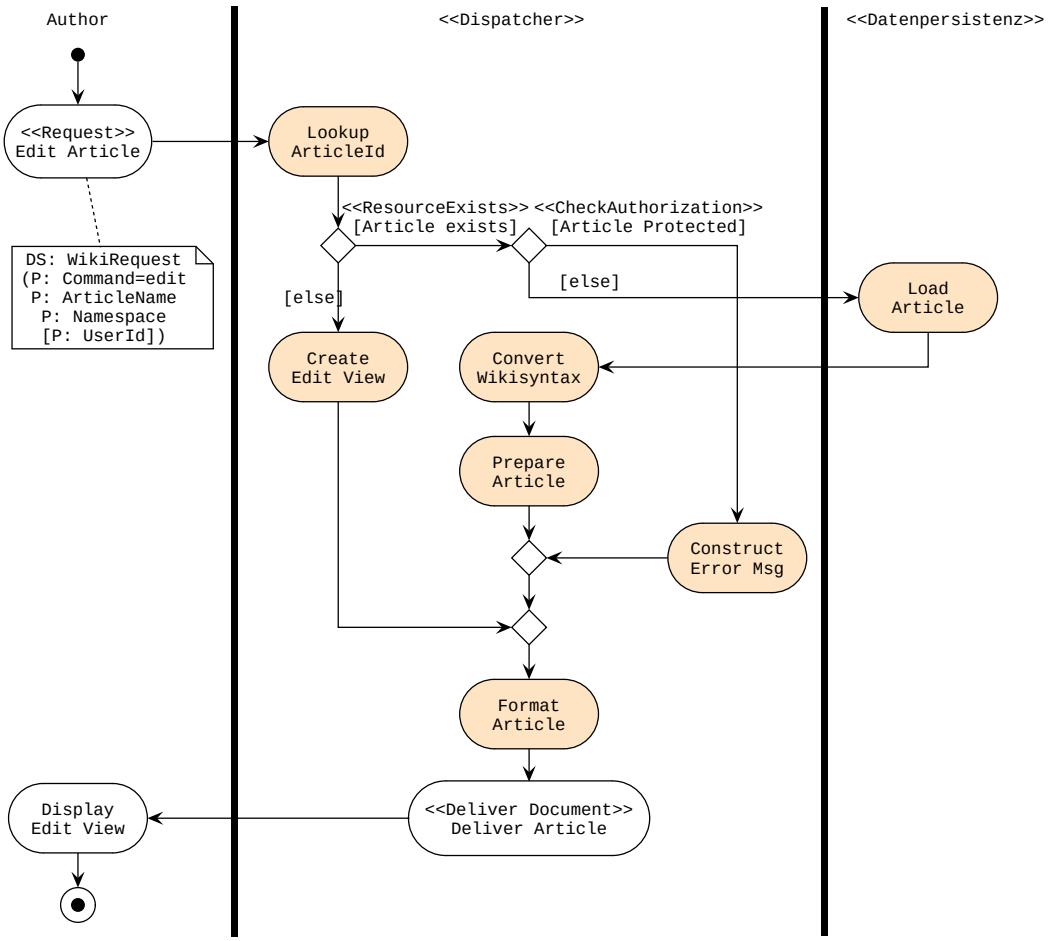

Abbildung 5.18: Wiki, dynamisches Modell, Edit Article

Die folgende Aktion „Prepare Article“ (Abbildung 5.20) nimmt als automatische Bearbeitung die Generierung eines Inhaltsverzeichnisses für den Artikel vor und gibt das Ergebnis weiter an die nächste Stufe, die Aktion „Format Article“ (Abbildung 5.21). In dieser Aktion wird der Artikel vom allgemeinen Zwischenformat, in dem er intern vorliegt, in das benötigte Ausgabeformat umgewandelt. Die letzte Aktion – „Deliver Article“ – liefert das so generierte fertige Dokument an den Benutzer aus, dem das Dokument dann schlussendlich angezeigt wird.

5.4.3 Komponentenbeziehungen

Hier werden die Beziehungen zwischen den Komponenten beschrieben, die im Abschnitt 4.3.2 noch offen gelassen wurden. Die Abbildung 5.22 auf Seite 169 zeigt die Beziehungen für das Wiki-Szenario. Die Wiki-Spezialfunktionen sind in einer eigenen Komponente zusammengefasst. Hier wird z. B. die Behandlung der Wikisyntax erledigt.

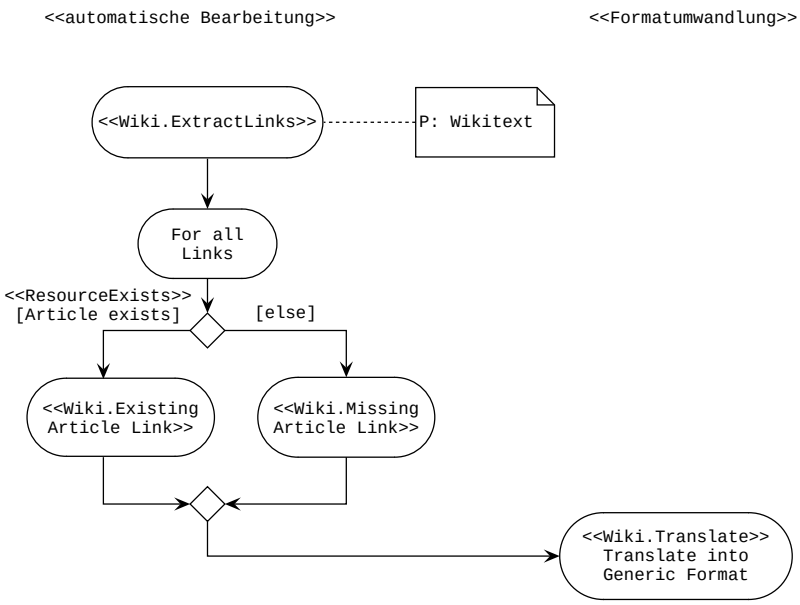

Abbildung 5.19: Wiki, dynamisches Modell, Convert Wikisyntax

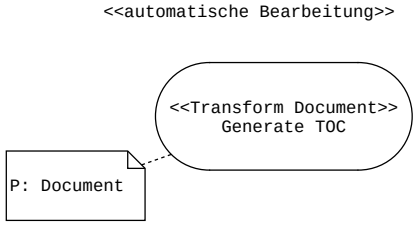

Abbildung 5.20: Wiki, dynamisches Modell, Prepare Article

Die beiden Komponenten „Trennung Präsentation“ und „Formatumwandlung“ basieren ganz wesentlich auf der Funktionalität der Komponente „Automatische Bearbeitung“. Die Validierung findet an zwei unterschiedlichen Orten statt: zum einen in der Wiki-Komponente bei der Überprüfung der Wikisyntax, zum anderen bei der allgemeinen Validierung bei der Formatierung der Dokumente.

5.5 Unterstützung des Meta-Modells durch Publikationssysteme

Das Publikationssystem muss in der Lage sein, die mit Hilfe des statischen Modells beschriebenen Datenstrukturen anzulegen und diese Datenstrukturen auch zur Laufzeit verwalten zu können. Das Publikationssystem muss auch die dynamischen Abläufe, d. h. das Zusammenspiel

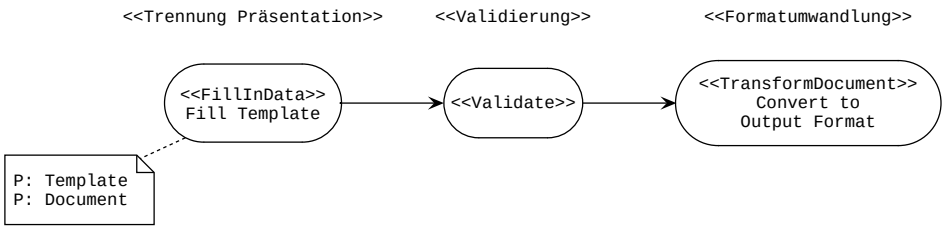

Abbildung 5.21: Wiki, dynamisches Modell, Format Article

der Komponenten, anhand von Activity Diagrammen steuern können. Die Startpunkte der dynamischen Abläufe sind dabei mit dem Request-Stereotypen markiert. So kann der Dispatcher zur Laufzeit anhand der Request-Parameter erkennen, welcher Ablauf ausgewählt werden soll. Über die Fähigkeit hinaus, das Modell zu lesen, muss das Publikationssystem natürlich auch über alle erforderlichen Komponenten verfügen. Welche das sind, kann aus dem Modell der Publikationsanwendung herausgelesen werden. Das Publikationssystem muss über einen bestimmten Baukastenbestand an Funktionen verfügen, die von den Komponenten angeboten werden. Welchen Umfang dieser Baukastenbestand haben muss, wird vom Meta-Modell bestimmt.

Das Publikationssystem muss also offensichtlich in die Lage versetzt werden, Modelle zu verstehen, die in der Sprache des Meta-Modells verfasst sind. Auf der technischen Lösungsebene können die Modelle für die Publikationssysteme beispielsweise mit Hilfe von XMI festgehalten werden. Sowohl Code-Generatoren als auch die Konfiguration zur Laufzeit können auf dieses Beschreibungsformat zurückgreifen. Dadurch, dass es auf etablierten XML-Technologien aufbaut, können außerdem bereits vorhandene und bewährte XML-Parser eingesetzt werden.

In Abschnitt 5.1 sind eine Reihe von Varianten beschrieben, wie das Meta-Modell verwendet werden kann. Zwei Varianten sind z. B. Generierung und Interpretation. Diese beiden Varianten können in einem gemischten Ansatz auch gleichzeitig eingesetzt werden: So kann Generierung im Zusammenhang mit dem statischen Meta-Modell verwendet werden und Interpretation im Zusammenhang mit dem dynamischen Meta-Modell. Die beiden Bereiche des Meta-Modells sind soweit unabhängig voneinander, so dass dies problemlos möglich ist. Dieser kombinierte Ansatz ist vorteilhaft, da er Generierung und Interpretation für die Bereiche verwendet, in denen jeweils ihre Stärken liegen (vgl. [Haa06]): Generatoren spielen vor allem bei strukturellen Aspekten ihre Stärken aus, während Interpreter bei der Beschreibung des Verhaltens eines Systems vorteilhaft sind.

Diese Kombination passt auch zu den jeweils charakteristischen Eigenschaften des jeweiligen Bereichs des Meta-Modells:

- Verhalten ist dynamisch und flüchtig, d. h. es braucht nicht in einer Datenhaltung gespeichert zu werden. Es soll jedoch schnell an veränderte Anforderungen angepasst werden können.
- Datenstrukturen sind normalerweise recht stabil. Wenn sie sich ändern, dann müssen bestehende Datenstrukturen in Datenbanken eventuell transformiert werden. Dies kann er-

heblichen Aufwand (für Planung sowie Durchführung) bedeuten. Gerade auch in Umgebungen mit hohen Anforderungen an die Verfügbarkeit und Performanz (man denke auch an cluster-basierte Systeme) ist der Generatoransatz vorteilhafter und weniger risikoträchtig. Das Problem, bestehende Datenbanken an veränderte Datenstrukturen anzupassen, wird in dieser Arbeit nicht näher betrachtet.

5.5.1 Cocoon

Im Abschnitt 4.4.2 wurde Cocoon bereits auf die Erfüllung der Anforderungen an Publikationssysteme untersucht. An dieser Stelle wird nun beschrieben, welche genaueren Eigenschaften Cocoon mitbringt, um das vorgestellte Meta-Modell zu nutzen und an welchen Stellen noch Defizite bestehen. Im Abschnitt 4.4.2.1 wurde Cocoon bereits darauf untersucht, welche Komponenten noch grundsätzlich weiterentwickelt werden müssen oder ganz fehlen. Es wird also geklärt, wie das Meta-Modell mit Cocoon implementiert werden kann.

5.5.1.1 Unterstützung des statischen Meta-Modells

Wie in Abschnitt 5.3.1 beschrieben, ist der statische Teil des Meta-Modells für Publikationsanwendungen zur besseren Verständlichkeit aufgeteilt in ein Dokumentenmodell und ein weiteres Daten-Meta-Modell. Entsprechend dieser Aufteilung wird hier auch vorgegangen, um die Unterstützung des statischen Meta-Modells durch Cocoon zu untersuchen.

Das Meta-Modell für Dokumente wird von der Komponente „Validierung“ verwendet, um die Struktur von zu erstellenden Dokumenten auf ihre Konformität mit diesen Strukturvorgaben zu überprüfen. Cocoon setzt intern einen Strom von XML Elementen ein, um die zu verarbeitenden Daten zu darzustellen. Damit ist sichergestellt, dass alle Dokumente zumindest eine XML-konforme Syntax besitzen, da sie sonst nicht verarbeitet werden können. Ob sie auch darüber hinaus auch bestimmten Strukturvorschriften entsprechen (also nach XML Terminologie „valide“ sind), kann mit Hilfe des ValidationTransformer von Cocoon mit geringem Aufwand anhand von XML Schema, RelaxNG und weiteren formalen Strukturbeschreibungssprachen geprüft werden: In [BKST07] werden Prinzipien, Muster und Vorgehensweisen beschrieben, um von einem Dokumentenmodell zu einem XML Schema zu gelangen. Wenn es nötig ist, die Pipeline (vgl. Abbildung 4.6 auf Seite 133) abhängig von diesem Validierungsergebnis feiner zu verzweigen als nur die Pipeline mit einem Fehler abzubrechen, dann ist zusätzlicher Aufwand erforderlich: Es muss eine Cocoon Action programmiert werden, die als Parameter eine XML Schema Definition erhält und den verarbeiteten SAX Strom überprüft. Das Ergebnis dieser Überprüfung kann den nachfolgenden Stationen in der Pipeline einfach als Umgebungsparameter bereitgestellt werden und dort für weitere Verzweigungen verwendet werden.

Cocoon kann Daten per SQL speichern. Dazu bindet man einen SQL-Transformer in die Pipeline ein, der über entsprechende SQL Abfragen mit der Datenbank kommuniziert. Dies ist aber nicht in allen Fällen die optimale Lösung: v. a. da Cocoon mit Hilfe objektorientierter Programmierung in Java erweitert werden kann, bieten es sich an, die verwalteten Objekte durch ein

objekt-relationales Mapping (O/R-Mapping) in Datenbanken zu persistieren. Dies stellt einen moderneren Ansatz dar und passt deutlich besser zur Objektorientierung als dies die direkte Verwendung von SQL tut. Cocoon bringt zwar von sich aus keine O/R-Mapping Unterstützung mit, allerdings gibt es folgende Möglichkeiten, dieses nachzurüsten:

- Hibernate[4]: ist eine weit verbreitete und gut entwickelte O/R-Mapping Unterstützung, das auch in vielen großen kommerziellen Anwendungen eingesetzt wird. Es kann derzeit als Industriestandard angesehen werden.
- Apache OJB[5]: hat das Ziel, mit verschiedenen Standards konform zu sein (ODMG 3.0[6], JDO[7]), befindet sich aber noch vor der Fertigstellung einer vollständigen Unterstützung.

Das grundlegende Prinzip ist bei beiden Lösungen dasselbe: Objekte und Strukturen von Objekten werden in einer Datenbank (in der Regel eine SQL-Datenbank) persistiert und können auf umgekehrtem Weg wieder als Objekte und Strukturen von Objekten in den Speicher des Java-Programms geladen werden. Welche Objekte wie genau persistiert und wieder geladen werden, kann umfangreich konfiguriert werden. Für eine Implementierung in Cocoon empfiehlt sich die Verwendung von Hibernate, schon aus dem Grund, dass sich die Apache OJB noch in Entwicklung befindet und auch weil Hibernate sich bereits vielfach bewährt hat.

Der Einsatz eines O/R-Mappings in Cocoon kann folgendermaßen aussehen: Durch das statische Meta-Modell ist das O/R-Mapping festgelegt, d. h. hiermit ist spezifiziert, welche Objekte in welchen Strukturen existieren können. Diese Objekte und Strukturen werden zur Laufzeit durch Java Code in Cocoon aufgebaut und bei Bedarf in einer Datenbank persistiert und wieder geladen.

Um diese Objekte auch in einer Cocoon Pipeline verwenden zu können, muss unterschieden werden zwischen den Objekten, die Dokumente beschreiben, und den restlichen Objekten aus dem statischen Meta-Modell. Der Grund liegt darin, dass die Dokumente in der Pipeline als SAX-Strom verarbeitet werden und die restlichen Objekte als Umgebungsparameter der Pipeline bereitgestellt werden.

Das Umwandeln in einen SAX-Strom kann durch einen eigenen Generator erreicht werden, der zunächst die Objekte aus der Datenbank lädt und diese dann in einem SAX-Strom bereitstellt. Cocoon kann durch Java um eigene Pipeline-Komponenten erweitert werden.

Für den zweiten Weg (das Bereitstellen von Umgebungsparametern) stehen in Cocoon sogenannte Actions zur Verfügung. Diese sind in der Lage, Umgebungsparameter zu verändern und neue zu setzen. Wie beim ersten Weg ist auch hier noch Entwicklungsaufwand nötig, da Cocoon von sich aus keine passenden Actions mitbringt.

[4] Cocoon and Hibernate Tutorial: `http://wiki.apache.org/cocoon/CocoonAndHibernateTutorial`
[5] Apache ObjectRelationalBridge: `http://db.apache.org/ojb/`
[6] ODMG 3.0 Object Persistence API: `http://www.odmg.org/`
[7] Java Data Objects: `http://java.sun.com/jdo/index.jsp`

5.5.1.2 Unterstützung des dynamischen Meta-Modells

Bei der Unterstützung des dynamischen Meta-Modells ist die wesentliche Aufgabe, den Kontroll- und Datenfluss, der durch Activity Diagramme beschrieben ist, in Cocoon zu integrieren. Hierfür sind zwei Dinge erforderlich:

- eine computerlesbare Beschreibung der Activity Diagramme. Hier bietet sich wieder XMI als Beschreibungssprache an.
- ein in Cocoon integrierter Interpreter, der die Steuerung des Kontroll- und Datenflusses übernimmt.

Ein großer Vorteil von Cocoon liegt darin, dass es bereits einen eigenen Interpreter für die Kontroll- und Datenflusssteuerung mitbringt: hierzu dient das Konzept der Pipelines, welche über eine Sitemap konfiguriert und zur Laufzeit interpretiert werden. Um das Rad nicht neu zu erfinden, bietet es sich hier also an, die vorhandenen Konzepte und Bestandteile der Sitemap soweit wie möglich zu nutzen. Dies erfordert dann ein Mapping der Bestandteile des Meta-Modells auf die Bestandteile der Cocoon Sitemap. Für Bestandteile, die eventuell noch keine Entsprechung in Cocoon besitzen, muss zusätzlicher Aufwand betrieben werden, um diese nachzurüsten.

Ein möglicher Kritikpunkt an dieser Herangehensweise ist, dass es sich nicht mehr um eine reine Interpretation handelt: Es ist ein Zwischenschritt erforderlich, um vom Modell, das in einer anderen Beschreibungssprache vorliegt, in eine für Cocoon verarbeitbare Sitemap zu transformieren. Damit wird eine Selbstveränderlichkeit des Verhaltens des Systems zur Laufzeit erschwert. Es wäre erforschenswert, in welchen Szenarien eine solche Selbstveränderlichkeit wirklich erforderlich ist. Für Szenarien, in denen es wichtig ist, die Verhaltensregeln eines Systems nachvollziehen zu können, ist eine Selbstveränderlichkeit möglicherweise sogar schädlich. Für die Integration in Cocoon würde es bedeuten, einen wesentlichen Funktionsgrundstock nicht zu verwenden, sondern komplett neu zu implementieren. Ein Kompromiss besteht darin, das ursprüngliche dynamische Modell in XMI zur Laufzeit zu verändern und möglichst zeitnah die Sitemap neu zu generieren bzw. die Änderungen in die Sitemap zu übertragen. Da sowohl XMI als auch die Sitemap in XML beschrieben sind, kann eine solche Transformation mit XSL durchgeführt werden.

Die Beschreibung, wie das dynamische Meta-Modell von Cocoon unterstützt werden kann, ist in unterschiedliche Schritte aufgeteilt:

1. einer Beschreibung der Daten- und Kontrollflusssteuerung. Hier geht es um die Erkennung eines Requests, den generellen Fluss mit Verzweigungen und Schleifen und die grundsätzliche Einbindung von generischen und zusammengesetzten Aktionen.
2. einer Beschreibung, wie die generischen Aktionen und Verzweigungen konkret implementiert werden können.

Es gibt bei der Benutzerinteraktion zwei unterschiedliche Varianten: Bei der ersten Variante reicht ein einfacher Request-Response-Zyklus zur vollständigen Bearbeitung aus. Bei der zweiten Varianten sind mehrere solcher Zyklen hintereinander erforderlich, die logisch miteinander

zusammenhängen. In den Activity Diagrammen liegt die zweite Variante dann vor, wenn ein Activity Diagramm als Aktion ein anderes Activity Diagramm einbindet, dessen erste Aktion mit dem Stereotypen «Request» gekennzeichnet ist. Dieser Fall ist aufgrund der Zustandslosigkeit von HTTP etwas aufwändiger handzuhaben. Es wird daher zunächst auf die einfachere erste Variante eingegangen werden und anschließend auf die zweite Variante.

Jede Interaktion vom Benutzer mit Cocoon läuft erst einmal über eine Pipeline. Daher erfolgt die Requesterkennung, welche Funktion nun aufgerufen wurde, ebenfalls über die Pipeline-Definitionen. Cocoon bringt bereits einen RequestParameter-Selektor mit, der hierfür sehr gut geeignet ist. Dabei wird die gewünschte Funktion als Parameter im HTTP-Request festgelegt und auf dieser Basis die passende Pipeline ausgewählt. Die durch die einzelnen Activity Diagramme beschriebenen Abläufe erhalten also jeweils einen Pipeline-Definitionseintrag in der Sitemap.

Die wichtigsten Bausteine für den weiteren Ablauf in den Activity Diagrammen sind generische und zusammengesetzte Aktionen. Sie können in mehreren Activity Diagrammen wiederverwendet werden. Daher sollten sie in Cocoon jeweils als einzelne Ressources in der Sitemap definiert werden. Auf diese Weise können sie in beliebigen anderen Pipelines eingesetzt werden. Diese Ressourcen stellen somit eine Bibliothek wiederverwendbarer Funktionsblöcke dar.

Verzweigungen können in Cocoon als Actions, Matcher und Selektoren implementiert werden. Actions sind sehr allgemein gehalten und sind eigentlich gar nicht für Verzweigungen gedacht, auch wenn sie dafür verwendet werden können. Matcher dienen dazu, einfache Ja-oder-Nein-Entscheidungen zu treffen und einen Unterablauf nur dann auszuführen, wenn eine Bedingung erfüllt ist. Selektoren sind ähnlich wie Matcher, aber etwas flexibler, da sie auch spezifische Fälle prüfen können und auch in mehr als zwei unterschiedliche Unterabläufe verzweigen können. Da Matcher jedoch schon für einfache „if-then-else“ Fälle zu wenig mächtig sind, sind Selektoren die erste Wahl für die Unterstützung von Verzweigungen.

Z. B. im Activity Diagramm in Abbildung 5.16 kommt ein paralleler Ablauf vor. Der Grundgedanke hinter Cocoons Pipeline Ansatz ist jedoch ein sequentieller Ablauf. Die parallelen Aktionen müssen also sequentiell ausgeführt werden, was technisch keine Schwierigkeit darstellt. Falls die Parallelität erhalten bleiben soll, muss Cocoon um diese Möglichkeit erweitert werden. Dabei muss überlegt werden, wie mit dem SAX Strom umgegangen werden soll, d. h. ob er ebenfalls parallel aufgeteilt und anschließend wieder zusammengeführt werden soll, oder ob ein einfacheres Vorgehen ausreichend ist. Cocoon bietet hierfür eingeschränkte Unterstützung durch einen sogenannten Aggregator. Die Einschränkungen bestehen darin, dass ein Aggregator nur am Anfang der Pipeline an Stelle eines Generators vorkommen darf und er nicht in der Lage ist, Ressourcen einzubinden.

Bei der zweiten Variante mit mehrstufiger Benutzerinteraktion bietet Cocoon das sogenannte Cocoon Flow[8] an, das auch Schleifen ermöglicht, d. h. den Ablauf erst fortsetzt, wenn eine bestimmte Bedingung erfüllt ist und sonst wieder zu einem vorherigen Startpunkt zurückkehrt. Für den Aufbau von Schleifen und mehrstufigen Interaktionen wird ein Control Flow Script erstellt, das folgende Parameter entgegen nimmt:

[8] Manchmal auch Flow Script genannt

- eine Ressource bzw. Pipeline, über die iteriert werden soll bzw. die aufgerufen werden soll.
- eine Abbruchbedingung für die Entscheidung, ob die Schleife beendet werden soll. Bei mehrstufigen Interaktionen ist dies nicht erforderlich.

Die Abbruchbedingungen können im Control Flow Script gut dargestellt werden. Dagegen ist für die Aufruf der Ressource noch Entwicklungsarbeit nötig: Zur Zeit können aus Control Flow Script keine Ressourcen aufgerufen werden. Der Aufruf der Ressourcen muss so erfolgen, dass er in den aktuellen SAX Strom der Pipeline integriert ist. Dies gilt nicht für den Aufruf von anderen Pipelines, da diese hier eine mehrstufige Interaktion darstellen und im Gegensatz zum Aufruf von Ressourcen einen eigenen SAX Strom aufbauen. Für mehrstufige Interaktionen wird jeder Aufruf einer mit dem «Request» Stereotypen gekennzeichneten Aktion aus dem Activity Diagramm in ein Flow Script verpackt, das die Pipeline ausführt und anschließend mit der Hilfe von Continuations wieder an die aufrufende Stelle zurückkehrt und die Pipeline fortsetzt.

An dieser Stelle soll nun genauer beschrieben werden, wie die konkreten generischen Aktionen und Verzweigungen in Cocoon implementiert werden können. Hier werden alle Aktionen aus Abschnitt 5.3.2 aufgegriffen.

Request: (Dispatcher)
Jede mit diesem Stereotypen gekennzeichnete Aktion taucht als eigene Pipeline in der Sitemap auf. Durch Angabe des gewünschten Befehls als Parameter kann mit Hilfe des RequestParameter Selectors von Cocoon entschieden werden, welcher Request ausgeführt werden soll.

Extract external ResourceId: (Dispatcher)
Diese Aktion muss in Cocoon nicht extra implementiert werden. Alle Parameter, die mit einem Request mitgeschickt werden, sind bereits beim Eintritt in die Pipeline zugreifbar.

Parse external ResourceId: (Dispatcher)
Für diese Aktion muss eine neue Action implementiert werden, die die entsprechenden Daten aus dem Request zerlegt und der Pipeline als Umgebungsparameter zur Verfügung stellt.

Extract ResourceName: (Dispatcher)
Diese Aktion muss in Cocoon nicht extra implementiert werden. Alle Parameter, die mit einem Request mitgeschickt werden, sind bereits beim Eintritt in die Pipeline zugreifbar.

Extract Namespace: (Dispatcher)
Diese Aktion muss in Cocoon nicht extra implementiert werden. Alle Parameter, die mit einem Request mitgeschickt werden, sind bereits beim Eintritt in die Pipeline zugreifbar.

Deliver Document: (Dispatcher)
Diese Aktion muss in Cocoon nicht extra implementiert werden. Cocoon bringt bereits eine Viezahl von Serializern für unterschiedliche Formate mit, die hier verwendet werden können.

ExtractRequestParameter: (Dispatcher)
Diese Aktion muss in Cocoon nicht extra implementiert werden. Alle Parameter, die mit einem Request mitgeschickt werden, sind bereits beim Eintritt in die Pipeline zugreifbar.

UpdateIndex: (Multimedia)
Diese Aktion muss als Action noch in Cocoon implementiert werden. Die Action erzeugt ein neues Metadaten-Objekt für die angegebene Ressource in der Datenstruktur für Metadaten. Für die persistente Speicherung sorgt Hibernate.

StoreBinary: (Datenpersistenz)
Diese Aktion muss als Action noch in Cocoon implementiert werden. Die Action erzeugt ein neues Daten-Objekt für die angegebene Ressource in der Datenstruktur für den ebenfalls angegebenen Namespace. Für die persistente Speicherung sorgt Hibernate.

CreateNewId: (Datenpersistenz)
Diese Aktion muss als Action noch in Cocoon implementiert werden. Die Action kann dazu entweder in der Datenhaltung nachschauen, welcher Identifikator noch nicht belegt ist, oder sie erzeugt besser gleich einen neuen, global eindeutigen Identifikator[9].

GetMultimediaData: (Multimedia)
Das Gegenstück zu UpdateIndex muss ebenfalls erst noch als Action implementiert werden. Sie lädt über Hibernate die Metadaten wieder in das System und stellt sie der Pipeline zur Verfügung.

GetMetadata: (Datenpersistenz)
Diese Aktion baut auf GetMultimediaData auf, ist jedoch nicht beschränkt auf Metadaten zu Multimedia Ressourcen, sondern lädt allgemeine Metadaten zu nicht genauer bestimmten Ressourcen.

SetRequestParameter: (Dispatcher)
Diese Aktion muss in Cocoon nicht extra implementiert werden. Innerhalb der Pipeline kann jederzeit auf den Request zugegriffen werden, um Parameter zu setzen.

Search: (Suchfunktion)
Cocoon besitzt bereits eine interne Suchfunktion in Form eines Generators, der eine Suche auf einer Apache Lucene Datenbank (siehe A2.15 in Abschnitt 4.4.2.1) ausführt. Zum Implementieren ist dann eine Verbindung zu Lucene, so dass bei neu erzeugten, gelöschten und veränderten Dokumenten der Suchindex angepasst wird.

Search Index: (Datenpersistenz)
Muss in Cocoon nicht eigens implementiert werden, da dies über Apache Lucene bereits enthalten ist.

ListResources: (Datenpersistenz)
Diese Aktion muss noch als eigene Action implementiert werden. Sie kann sich dabei auf Hibernate stützen und ist mit geringem Aufwand realisierbar.

ListLinkingDocuments: (Navigation)
Diese Aktion muss noch als eigene Action implementiert werden. Sie kann sich dabei auf Hibernate stützen und ist mit geringem Aufwand realisierbar.

ListResources: (Datenpersistenz)
Diese Aktion muss noch als eigene Action implementiert werden. Sie kann sich dabei auf Hibernate stützen und ist mit geringem Aufwand realisierbar.

[9] Hier kann beispielsweise ein Standard wie der SMPTE 330M-200X verwendet werden

FindDeadLinks: (Navigation)
Diese Aktion muss noch als eigene Action implementiert werden. Sie kann sich dabei auf Hibernate stützen und ist mit geringem Aufwand realisierbar.

ListDocumentsWithoutReferers: (Navigation)
Diese Aktion muss noch als eigene Action implementiert werden. Sie kann sich dabei auf Hibernate stützen und ist mit geringem Aufwand realisierbar.

GetRandomId: (Datenpersistenz)
Diese Aktion muss noch als eigene Action implementiert werden. Sie kann sich dabei auf Hibernate stützen und ist mit geringem Aufwand realisierbar.

GetStatistics: (Datenpersistenz)
Diese Aktion muss noch als eigene Action implementiert werden. Sie kann sich dabei weitgehend auf Hibernate stützen und ist daher mit relativ geringem Aufwand implementierbar.

CreateAccount: (Mehrbenutzer)
Diese Aktion muss noch als eigene Action implementiert werden. Sie legt ein neues Benutzerobjekt an und befüllt es mit Werten. Über Hibernate wird es in der Datenbank gespeichert.

LoadUserData: (Mehrbenutzer)
Diese Aktion muss noch als eigene Action implementiert werden. Sie kann sich dabei weitgehend auf Hibernate stützen und ist daher mit relativ geringem Aufwand implementierbar.

StoreUserData: (Mehrbenutzer)
Für diese Aktion ist lediglich eine Action erforderlich, die über Hibernate das Speichern des Benutzerobjekts in der Datenbank veranlasst.

SaveUserSpecific: (Mehrbenutzer)
Diese Aktion muss noch als eigene Action implementiert werden. Sie muss dazu eine Datenstruktur anlegen, die Namespace-abhängig Daten für einen bestimmten Benutzer aufnehmen kann. Die Persistierung erfolgt wieder über Hibernate.

LoadUserSpecific: (Mehrbenutzer)
Diese Aktion muss noch als eigene Action implementiert werden. Sie kann sich dabei weitgehend auf Hibernate stützen und ist daher mit geringem Aufwand implementierbar.

DeleteUserSpecific: (Mehrbenutzer)
Diese Aktion muss noch als eigene Action implementiert werden. Sie löscht die Datenstrukturen wieder, die in SaveUserSpecific erzeugt wurden. Um das Laden und Löschen aus der Datenbank kümmert sich wieder Hibernate.

FilterResourceList: (Mehrbenutzer)
Diese Aktion muss noch als eigene Action implementiert werden. Sie erhält als Parameter eine Liste von Ressourcen-Identifikatoren und ein Prädikat, nach dem die Liste gefiltert werden soll.

FillInData: (Automatische Bearbeitung)
Hier bietet sich die Verwendung des in Cocoon vorhandenen XSLT Transformers an. Er

kann bereits auf Request-Parameter zurückgreifen und kann beliebige XML Dokumente mit Daten befüllen. Benötigte Parameter können z. B. über eine passende Action in den Request geschrieben werden und stehen dann ebenfalls dem Transformer zur Verfügung.

LookupUserId: (Benutzerauthentifizierung)
Diese Aktion muss nicht eigens implementiert werden. In Cocoon kann jederzeit auf Request-Parameter zugegriffen werden.

VerifyLogin: (Benutzerauthentifizierung)
Für diese Aktion muss noch eine neue Action implementiert werden, die das Benutzerobjekt aus der Datenhaltung liest und damit den Benutzernamen und das Passwort aus dem Request überprüft. Der Implementierungsaufwand hierfür ist ebenfalls gering.

LoadDataContainers: (Datenpersistenz)
Für diese Aktion muss noch eine neue Action implementiert werden, die sich jedoch stark auf Hibernate stützen kann und somit nur wenig Aufwand erfordert.

StoreResourceMetadata: (Datenpersistenz)
Das Gegenstück zu GetMetadata. Diese Aktion muss als Action noch in Cocoon implementiert werden. Die Action speichert mit Hilfe von Hibernate ein Metadaten-Objekt in der Datenhaltung und verknüpft es mit dem eindeutigen Identifikator der Ressource.

StoreResource: (Datenpersistenz)
Diese Aktion muss implementiert werden. Sie erzeugt ein neues Objekt für die Ressource und speichert es über Hibernate in der Datenhaltung.

DeleteResource: (Datenpersistenz)
Diese Aktion muss implementiert werden. Über den eindeutigen Identifikator ist die Ressource bestimmt, die gelöscht werden soll. Das eigentliche Löschen übernimmt dann Hibernate.

LoadResource: (Datenpersistenz)
Diese Aktion muss implementiert werden. Über den eindeutigen Identifikator ist die Ressource bestimmt, die geladen werden soll. Das eigentliche Laden übernimmt Hibernate.

TransformDocument: (Automatische Bearbeitung)
Hier kann wieder der von Cocoon bereitgestellte XSLT Transformator eingesetzt werden. Er erhält als Parameter das XML Dokument und die Transformationsvorschriften als XSL und führt die Transformation durch.

Validate: (Validierung)
Mit Hilfe des ValidationTransformers kann die Struktur von Dokumenten auf ihre Konformität mit Strukturvorschriften geprüft werden. Dieser mitgelieferte Transformer bricht die Pipeline bei einem Fehler jedoch einfach ab. Falls also eine eigene Fehlerbehandlung gewünscht wird, so muss Cocoon hier erweitert werden. Dies ist jedoch mit geringem Aufwand möglich.

AddLatestVersion: (Versionierung)
Diese Aktion muss noch implementiert werden. Die entsprechende Action erkennt anhand des eindeutigen Identifikators, zu welcher Ressource eine neue Version erzeugt werden soll. Sie muss dann intern die Datenstrukturen ändern, d. h. die momentan aktuelle Version

als historische Version markieren und die neue Version als neue aktuelle Version in die Kette der Versionen einhängen. Für die Implementierung ist mit einem mittleren Aufwand zu rechnen.

ListAllVersions: (Versionierung)
Diese Aktion muss noch implementiert werden, ist aber im Vergleich zu AddLatestVersion weniger aufwändig zu leisten. Die Action muss anhand des angegebenen eindeutigen Identifikators zur Kette aller Versionen gelangen und eine Liste dieser Versionen zurückliefern.

UpdateIdMapping: (IdMapper)
Diese Aktion muss noch implementiert werden. Der Aufwand hierfür ist jedoch gering. Die Action pflegt eine Datenstruktur, die eine Liste von Paaren von eindeutigen Identifikatoren pflegt. Um das Speichern und Lesen aus der Datenbank kann sich wieder Hibernate kümmern, so dass hier lediglich als Implementierungsaufwand bleibt, die gewünschten Mappingänderungen durchzuführen.

Neben diesen generischen Aktionen gibt es in den Activity Diagrammen auch sogenannte Verzweigungen:

ResourceExists: (Dispatcher)
Diese Verzweigung muss noch als eigener Selektor implementiert werden. Er nimmt einen UniqueIdentifier als Parameter entgegen und prüft in der internen Datenstruktur nach, ob dazu eine Ressource existiert.

ResourceIsProtected: (Dispatcher)
Diese Verzweigung muss noch als eigener Selektor implementiert werden. Er nimmt einen UniqueIdentifier als Parameter entgegen und prüft in der internen Datenstruktur nach, ob der Zugriff auf die dazugehörige Ressource durch Zugriffsrechte eingeschränkt ist.

CheckAuthorization: (Rechteverwaltung)
Diese Verzweigung muss noch als eigener Selektor implementiert werden. Er nimmt als Parameter eine UserId, einen UniqueIdentifier für die Ressource und die gewünschte Zugriffsart entgegen und prüft in der internen Datenstruktur nach, ob der gewünschte Zugriff erlaubt ist.

ResourceIsCached: (Datenpersistenz)
Diese Verzweigung muss noch als eigener Selektor implementiert werden. Er nimmt als Parameter einen UniqueIdentifier entgegen und prüft in der internen Datenstruktur nach, ob die angegebene Ressource in einer aktuellen Version im Cache gespeichert ist.

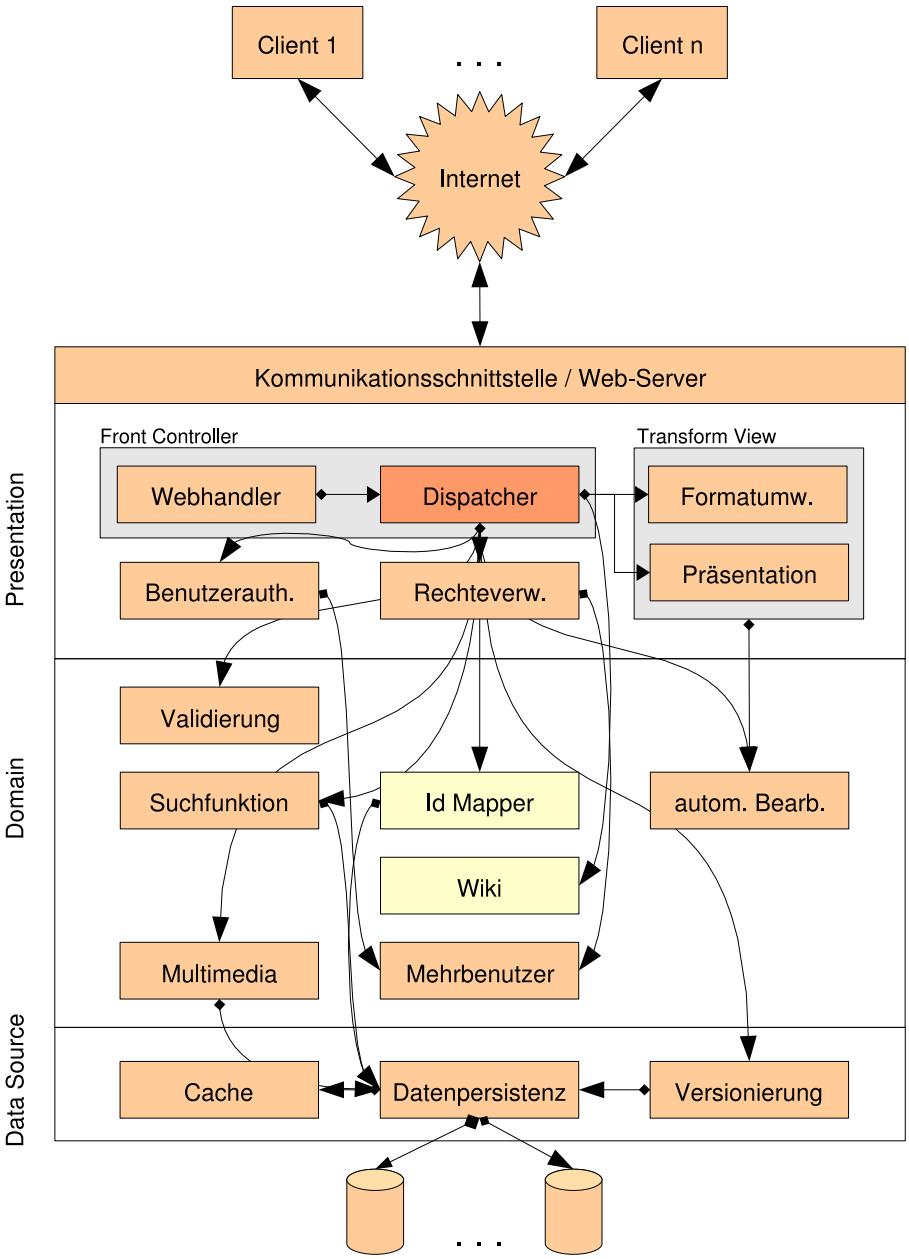

Abbildung 5.22: Wiki, Komponentenmodell

6 Schluss

Das Leben kann nur rückblickend verstanden werden.
Es muss aber vorausschauend gelebt werden.
– SÖREN KIERKEGAARD

Zum Schluss dieser Arbeit werden die Ergebnisse zusammengefasst und durch einen Rückblick eine Bewertung des Erreichten vorgenommen (Abschnitt 6.1) sowie ein darauf aufbauender Ausblick auf offen gebliebene und im Laufe der Arbeit erschienene neue Fragen gegeben (Abschnitt 6.2).

6.1 Ergebnisse

In dieser Arbeit wurde eine Software-Architektur für Publikationssysteme erarbeitet, die auf einer umfassenden Aufbereitung des Fachgebiets des elektronischen Publizierens basiert. Das hierbei entwickelte und ins Fachgebiet eingeführte Meta-Modell ermöglicht eine Höherhebung des Fachwissens: Es zeigt den Pfad weg von spezialisierten, spezifischen Publikationssystemen hin zu einer allgemeinen Verwendbarkeit und der Bewahrung und Weiterentwicklung des erworbenen Wissens.

Es wurde zunächst festgestellt, dass bislang im Fachgebiet des elektronischen Publizierens keine einheitlichen Begriffsdefinitionen verwendet wurden. Daher wurden die wichtigen Begriffe, Konzepte und Prozesse definiert. Es wurde das Konzept der Publikationskette herausgearbeitet, die stark von zwei wesentlichen Vorteilen von strukturierten Dokumenten profitiert: Separation of Concerns und automatisierte Verarbeitbarkeit.

Auf der Basis der Publikationskette sowie der allgemeinen Ziele des elektronischen Publizierens wie Zeitgewinn, Kostenreduktion, Qualitätsverbesserung und besserer Mediendurchdringung wurden anschließend umfangreiche Anforderungen an Publikationssysteme zusammengetragen.

Zur genaueren Untersuchung von Publikationssystemen und Publikationsanwendungen wurden zunächst ähnliche Anforderungen zu größeren Funktionsclustern zusammengefasst. Danach wurde eine große Anzahl von Einsatzszenarien für Publikationssysteme genauer beschrieben und die jeweilige Signifikanz der Funktionscluster für diese Szenarien dargestellt. Die große Anzahl belegt auch das breite Spektrum, das von Publikationsanwendungen abgedeckt werden kann, von digitalen Zeitungen über webbasierte Ticketingsystem bis hin zu Internet Banking.

Als Ergebnis zeigten sich ebenso einige Funktionscluster mit hoher Signifikanz in sehr vielen Szenarien wie auch solche Funktionscluster mit geringer Signifikanz in den meisten Szenarien. Mit der wichtigste Funktionscluster überhaupt war die Bereitstellung von Suchmöglichkeiten, wohingegen die Validierung kaum Bedeutung besitzt. Dies war zunächst überraschend und liegt darin begründet, dass Validierung nur selten einen direkten Nutzen für ein Szenario besitzt. Ebenso war die Dynamik relativ unwichtig und hing meist ab von der Signifikanz der Interaktivität.

Durch die Bewertung der Signifikanzen der Funktionscluster für einzelne Szenarien wurde es möglich, einander ähnliche Szenarien zu finden und gemeinsame Oberklassen von Szenarien zu bilden. Überraschend war dabei, dass eine Oberklasse von den Szenarien für Gerichtsportale, digitale Bibliotheken, autonome Suchdienste und persönliche digitale Bibliotheken gebildet wurde.

Um zum Ziel zu gelangen, eine Software-Architektur für Publikationssysteme vorzuschlagen, wurde anschließend ein Überblick über das Fachgebiet der Software-Architektur gegeben. Dies war vor allem wichtig zur Herausstellung der relevanten Qualitätsziele für die Software-Architektur sowie zur Beschreibung, wie diese Qualitätsziele erreicht werden können.

Als Ergebnis wurde eine Software-Architektur für Publikationssysteme entwickelt. Es wurde begründet, welche Qualitätsziele hierbei besonders berücksichtigt wurden und wie diese erreicht wurden. Anschließend wurden zwei wichtige aktuelle Publikationssysteme ausgewählt und auf ihre Übereinstimmung mit der vorgeschlagenen Software-Architektur untersucht.

Im letzten Kapitel wurde die Erreichung des wesentlichsten Ziels der vorgeschlagenen Software-Architektur – die Modifizierbarkeit – durch den Einsatz eines Meta-Modells detailliert beschrieben. Dazu wurde nach einem kurzen Überblick über Meta-Modellierung ein Meta-Modell für Publikationsanwendungen vorgestellt. Dieses Meta-Modell wurde anhand der vollständigen Modellierung der Publikationsanwendung für Wikis verifiziert. Ebenso wurde das allgemeine Vorgehen beschrieben, um andere Publikationsanwendungen mit dem Meta-Modell zu beschreiben bzw. bestehende Modelle um neue Eigenschaften und Verhaltensweisen zu erweitern. Abschließend wurde das ausgewählte Publikationssystem Apache Cocoon untersucht, wie weit es das Meta-Modell unterstützt und es wurde herausgestellt, welche Bereiche noch fehlen bzw. angepasst werden müssen.

6.2 Ausblick

6.2.1 Untersuchung des Nutzens der Selbstveränderlichkeit

In der Arbeit wurde erwähnt, dass eine dynamische Interpretation des Meta-Modells eine Selbstveränderlichkeit des Verhaltens des Publikationssystems zur Laufzeit unterstützt. Es wäre erforschenswert, in welchen Szenarien eine solche Selbstveränderlichkeit vorteilhaft oder eventuell sogar erforderlich ist.

6.2.2 Implementierung in ein Publikationssystem

Es sollte eine tatsächliche Implementierung der Unterstützung des Meta-Modells in ein Publikationssystem vorgenommen werden. Als Ausgangspunkt hierfür bietet sich das Publikationssystem Apache Cocoon an, das bereits in Bezug auf seine Unterstützung des Meta-Modells betrachtet wurde.

Eine andere Möglichkeit wäre auch die vollständig neue Entwicklung eines geeigneten Publikationssystems. Da es jedoch eines der Ziele dieser Arbeit war, für verschiedene Publikationsanwendungen nicht verschiedene Publikationssysteme zu erfordern, ist die Erweiterung eines bestehenden Publikationssystems naheliegender.

6.2.3 Entwicklung eines Editors für die Modelle

Es wäre hilfreich, einen Editor zu haben, mit dessen Hilfe man auf Basis des Meta-Modells gültige Modelle für Publikationsanwendungen erstellen kann. Für die Entwicklung kann z. B. auf Arbeiten des Eclipse Projekts zurückgegriffen werden (siehe [BKS05, KEVH06]): Es gibt bereits eine Erweiterung für Eclipse, mit deren Hilfe ein grafischer Editor definiert werden kann, der selbst-definierte Meta-Modelle editiert. Die Meta-Modelle, die diese Modelle definieren, basieren auf Ecore, das die Meta-Meta-Modellsprache von Eclipse ist und damit der MOF in der Modellierungshierarchie der OMG entspricht.

Für die Definition grafischer Editoren gibt es in Eclipse bereits das „Graphical Editing Framework“ (GEF), dessen Einsatz etwas umständlich ist, so dass auf dessen Basis und unter Zuhilfenahme des Eclipse Modeling Framework (EMF), eine Erweiterung namens „Graphical Modeling Framework“ (GMF) entwickelt wurde, die einfach zu benutzen sein soll. Dabei dient das EMF dazu, die Meta-Modelle zu definieren. Das GEF erstellt die grafischen Editoren.

Für die Verwendung mit dem Meta-Modell für Publikationsanwendungen sind dabei folgende Punkte zu beachten:

- Der Editor benötigt eine Darstellung des Meta-Modells im Ecore Modell. Diese Darstellung wird in dieser Arbeit offen gelassen, sie stellt aber kein gravierendes Problem dar.
- Das Ergebnis (eine modellierte Publikationsanwendung) liegt in einem Format vor, das gegebenenfalls erneut konvertiert werden muss, um von einem Publikationssystem für eine konkrete Publikationsanwendung einsetzbar zu sein.

Ein Markt für ein solches Werkzeug scheint durchaus vorhanden zu sein, dies zeigt schon die Integration einer solchen grundsätzlichen Editorenfunktionalität in Eclipse.

6.2.4 Untersuchung und Modellierung weiterer Szenarien

In jüngerer Zeit kamen noch weitere Einsatzszenarien für Publikationsanwendungen auf, die eventuell neue Anforderungen stellen. Dies sind Anwendungen des sogenannten Web 2.0 wobei hier vor allem Social Software zu nennen ist. Diese Szenarien wurden in dieser Arbeit nicht mehr untersucht.

7 Glossar

Content: Als Content werden in Content Management Systemen Daten innerhalb eines Dokuments bezeichnet, die nicht lediglich zur Beschreibung von Daten des Dokuments dienen. Wie die Daten genau aussehen, ist dabei nicht von Interesse. Der Begriff dient vor allem zur Abgrenzung zwischen den in den Dokumenten verwertbaren Daten und Metadaten, die eher zu deren Verwaltung dienen. Beispiele sind etwa Texte, Bilder, Grafiken, Audio- und Videosequenzen. Content Management Systeme sind Software-Systeme, die (ähnlich wie Publikationssysteme) elektronische Dokumente publizieren. Die Dokumente sind jedoch nicht oder weit weniger strukturiert als bei Publikationssystemen.

Dokument: Unter einem Dokument ist eine endliche Datenabfolge zu verstehen, die in computerverarbeitbarer Form und effektiv zugreifbar nicht-flüchtig auf einem Datenträger vorliegt.

Elektronisches Publizieren: Der Begriff „elektronisches Publizieren“ besitzt zwei verschiedene Bedeutungen. Als Erstes versteht man darunter den Prozess, in dem Dokumente auf elektronischem Weg von ihrer Quelle zu ihrer Endnutzung gelangen. Als Quelle der Dokumente dienen dabei nicht ausschließlich menschliche Autoren; auch computergestützte Prozesse können Dokumente generieren. Die Dokumente können dabei nach ihrer Erzeugung eine Reihe von Zwischenstationen entlang der sogenannten Publikationskette durchlaufen.

Die zweite Bedeutung des Begriffs „elektronisches Publizieren“ steht für das wissenschaftliche Fachgebiet, das Modelle, Vorgehensweisen und Werkzeuge zur Verfügung stellt, die den oben erwähnten Prozess des elektronischen Publizierens ermöglichen und unterstützen. Es beschäftigt sich mit den Grundlagen, der theoretischen Analyse und Konzeption von Systemen.

Format: Das Format ist eine Festlegung, wie ein Dokument als Folge von Symbolen oder Zeichen zu kodieren ist. Bsp. für Formate sind: XML, HTML, WML, PostScript, PDF, ZIP, WAV, MPEG, JPEG. Vgl. auch den Begriff „Dateiformat“.

Inhalte: Die Inhalte stellen in einem strukturierten Dokument den wesentlichen Datengehalt dar, welcher in Form von natürlichsprachlich codierter Information vorliegt. Er ist weder den Strukturelementen noch den Präsentationsanweisungen zuzurechnen. Diese Inhalte können auch in einem nicht-strukturierten Dokument enthalten sein, dann jedoch ohne die Metadaten aus den Strukturelementen und ohne getrennte Präsentationsanweisungen. Die Inhalte sind von den Strukturelementen und den Präsentationsanweisungen unterscheidbar kodiert.

Markup: Wenn bei einem strukturierten Dokument die Strukturelemente im selben Zeichenstrom enthalten sind wie auch die eigentlichen Inhalte des Dokuments, dann bezeichnet

Markup diejenigen Teile des Zeichenstroms, die zu den Strukturelementen gehören. Eine andere Bezeichnung für Markup ist „Textauszeichnung“.

Modell der strukturierten Dokumente: beschreibt die Bestandteile von strukturierten Dokumenten und ihre Beziehungen. Ein strukturiertes Dokument entspricht dem Modell der strukturierten Dokumente, wenn die Strukturelemente die Inhalte in eine Hierarchie unterteilen. Diese kann als Baumstruktur dargestellt werden. Die Strukturelemente bestehen aus einem Namen und Attributen, die jeweils in den Knoten des Strukturbaums gespeichert sind.

Präsentation: legt fest, wie ein Dokument dem Benutzer dargeboten wird. Dies kann durch eine grafische Anzeige geschehen, aber auch auf eine andere Weise wie z. B. durch akustische Signale.

Die Präsentation kann grafische Attribute (Schriftart, Größe, Farbe, Rahmen), Sichtbarkeit, Positionierung (Abstände, Absolut/Relativ, ...) beeinhalten. Es existieren aber auch andere Präsentationsmedien als das visuelle: akustisch/auditiv (Stimmhöhe, Lautstärke, Geschwindigkeit), haptisch, . . .

Präsentationsanweisungen: Vorschriften zur Generierung der Präsentation (grafisch, akustisch, . . .) eines Dokuments. Eine Präsentationsanweisung beschreibt die Details der Präsentation für einen bestimmten Teil des Dokuments.

Publikation: bezeichnet ein verfügbar gemachtes Dokument oder eine Sammlung verfügbar gemachter Dokumente.

Publikationsanwendung: bezeichnet ein Software-System, das Dokumente im Zuge des elektronischen Publizierens verfügbar macht und so ein spezifisches Szenario implementiert. Eine Publikationsanwendung basiert in der Regel auf einem Publikationssystem.

Publikationskette: bezeichnet in der klassischen Sicht des Publikationswesens die Abfolge aller Stationen (Datenquellen, Bearbeitungsstationen, Ausgabeziele), die von einem Dokument während des Prozesses des elektronischen Publizierens durchlaufen werden. In der modernen Sicht ist dieser streng lineare Ablauf aufgelöst.

Publikationssystem: bezeichnet ein Softwaresystem, das die Möglichkeiten der Konzepte einmal des Modells der strukturierten Dokumente und zum anderen der Publikationskette durch eine konkrete Realisierung dieser Konzepte nutzbar macht.

Stilvorlage: maschinenverarbeitbare, in einer bestimmten Syntax beschriebene Sammlung von Präsentationsanweisungen für ein strukturiertes Dokument. Die Stilvorlage kann vom Dokument getrennt gespeichert sein.

Strukturiertes Dokument: Ein strukturiertes Dokument ist ein Dokument, bei dem zwischen den eigentlichen Inhalten, Strukturelementen und Präsentationsanweisungen unterschieden wird.

Strukturelemente: beschreiben den Aufbau eines strukturierten Dokuments. Sie stellen Metadaten über die eigentlichen Inhalte eines strukturierten Dokuments zur Verfügung und sind in einer Baumstruktur angeordnet.

Syntaktische Realisierung des Modells der strukturierten Dokumente: legt fest, wie die Kodierung von strukturierten Dokumenten als Zeichenkette gestaltet wird, d. h. sie beschreibt ein Format für strukturierte Dokumente. XML ist z. B. eine solche syntaktische Realisierung.

Szenario: ist ein Einsatzbeispiel, in dem ein Publikationssystem grundsätzlich verwendet werden kann.

Literaturverzeichnis

[ACM02] ALUR, DEEPAK, JOHN CRUPI und DAN MALKS: *Core J2EE Patterns: Best Practices and Design Strategies*. Prentice Hall, 2002.

[AII06] AIIM: *The Enterprise Content Management Association*, 4.2.2006. http://www.aiim.org.

[Arb03] ARBEITSKREIS ELEKTRONISCHES PUBLIZIEREN – BÖRSENVEREIN DES DEUTSCHEN BUCHHANDELS: *10 Jahre Elektronisches Publizieren – Rückblick und Vorschau*, Januar 2003. http://www.boersenverein.de/sixcms/media.php/686/AKEP_branchenumfrage2002.pdf.

[Arb04a] ARBEITSKREIS ELEKTRONISCHES PUBLIZIEREN – BÖRSENVEREIN DES DEUTSCHEN BUCHHANDELS: *Branchenbarometer Elektronisches Publizieren 2004*, Oktober 2004. http://www.boersenverein.de/sixcms/media.php/686/AKEP_Branchenbarometer2004.pdf.

[Arb04b] ARBEITSKREIS ELEKTRONISCHES PUBLIZIEREN – BÖRSENVEREIN DES DEUTSCHEN BUCHHANDELS: *Studie zur XML-Nutzung in Verlagen*, 2004. http://www.boersenverein.de/sixcms/detail.php?id=66572.

[Bal01] BALZERT, HELMUT: *Lehrbuch der Software-Technik*. Spektrum Akademischer Verlag, 2001.

[BBRS03] BROY, MANFRED, KLAUS BERGNER, ANDREAS RAUSCH und MARC SIHLING: *Vorlesung Softwarearchitektur verteilter Systeme*, 2002/2003. http://www4.in.tum.de/~sihling/savorles/.

[BCH+01] BROWN, KYLE, GARY CRAIG, GREG HESTER, JAIME NISWONGER, DAVID PITT und RUSSEL STINEHOUR: *Enterprise Java Programming with IBM WebSphere*. Addison-Wesley, 2001.

[BCK03] BASS, LEN, PAUL CLEMENTS und RICK KAZMAN: *Software Architecture in Practice*. Addison-Wesley, 2003.

[BHMH03] BAUMGARTNER, PETER, HARTMUT HÄFELE und KORNELIA MAIER-HÄFELE: *Evaluationsverfahren für den Vergleich virtueller Lernplattformen*, 2003. http://www.peter.baumgartner.name/articles/evaluation-lms.

[BK03] BRÜGGEMANN-KLEIN, ANNE: *Document Engineering im World-Wide Web*, 2002/2003. http://sunschlichter0.informatik.tu-muenchen.de/~brueggem/Vorlesungen/epWS2003/Skript/alle.pdf.

[BK04] BRÜGGEMANN-KLEIN, ANNE: *Elektronisches Publizieren – Document Engineering im World-Wide Web*, 2003/2004. Folien zur Vorlesung.

[BKS05] BOYETTE, NEIL, VIKAS KRISHNA und SAVITHA SRINIVASAN: *Eclipse Modeling Framework For Document Management*. In: *Proceedings of the 2005 ACM Symposium on Document Engineering*, Seiten 220–222, 2005.

[BKST07] BRÜGGEMANN-KLEIN, ANNE, THOMAS SCHÖPF und KARLHEINZ TONI: *Principles, Patterns and Procedures of XML Schema Design – Reporting from the XBlog Project*. In: *Proceedings of Extreme Markup Languages*, 2007. http://www.idealliance.org/papers/extreme/proceedings/html/2007/BruggemannKlein01/EML2007BruggemannKlein01.html.

[BMR+96] BUSCHMANN, FRANK, REGINE MEUNIER, HANS ROHNERT, PETER SOMMERLAD und MICHAEL STAL: *Pattern-Oriented Software Architecture – A System of Patterns*, Band 1. John Wiley & Sons, 1996.

[Bos97] BOSAK, JON: *XML, Java, and the Future of the Web*, 1997. http://www.xml.com/pub/a/w3j/s3.bosak.html.

[BRJ99] BOOCH, GRADY, JAMES RUMBAUGH und IVAR JACOBSON: *The Unified Modeling Language User Guide*. Addison-Wesley, 1999.

[Bro75] BROOKS, FREDERICK P.: *The Mythical Man-Month*. Addison-Wesley, 1975.

[BS00] BRÖSSLER, PETER und JOHANNES SIEDERSLEBEN (Herausgeber): *Softwaretechnik: Praxiswissen für Software-Ingenieure*. Carl Hanser Verlag, 2000.

[Buc05] BUCHBERGER, ROBERT: *Wenn es persönlich wird... – Webpersonalisierung*, 2005. http://www.contentmanager.de/magazin/artikel_47_wenn_es_persoenlich_wird_-_webpersonalisierung.html.

[Bun02] BUNDESMINISTERIUM FÜR BILDUNG UND FORSCHUNG: *Information vernetzen – Wissen aktivieren, Strategisches Positionspapier*, 2002. http://www.bmbf.de/pub/information_vernetzen-wissen_aktivieren.pdf.

[Byr04] BYRNE, TONY: *The CMS Report*, Winter 2004. Sample Only.

[Car05] CARNEGIE MELLON UNIVERSITY SOFTWARE ENGINEERING INSTITUTE: *How Do You Define Software Architecture?*, 13. Dezember 2005. http://www.sei.cmu.edu/architecture/definitions.html.

[CESW06] CLARK, TONY, ANDY EVANS, PAUL SAMMUT und JAMES WILLANS: *Applied Metamodelling – A Foundation for Language Driven Development*, Januar 2006.

[Cla97] CLARKE, ROGER: *Electronic Publishing: A Specialised Form of Elektronic Commerce*. 10th International Electronic Commerce Conference, 1997. http://www.anu.edu.au/people/Roger.Clarke/EC/Bled97.html.

[Cla99] CLARKE, ROGER: *Key Issues in Electronic Commerce and Electronic Publishing*. In: *Proceedings of the Ninth Australasian Information Online & On Disc Conference and Exhibition*, Sydney, 1999. http://www.csu.edu.au/special/online99/proceedings99/204b.htm.

[Coc01] COCKBURN, ALISTAIR: *Writing Effective Use Cases*. Addison-Wesley, 2001.

[Con95] CONNOLLY, DAN: *Overview of SGML Resources*, 1995. http://www.w3.org/MarkUp/SGML/.

[Die04] DIEPOLD, PETER: *Elektronisches Publizieren*. Zeitschrift für Erziehungswissenschaften, 7. Jg, Seiten 85–96, 04/2004. http://www.diepold.de/elektronisches_publizieren.pdf.

[DKGS04] DAVIS, MARC, SIMON KING, NATHAN GOOD und RISTO SARVAS: *From context to content: leveraging context to infer media metadata*. In: *Proceedings of the 12th annual ACM international conference on Multimedia*, Seiten 188–195. ACM, 2004.

[FGK01] FUHR, NORBERT, NORBERT GÖVERT und CLAUS-PETER KLAS: *PADDLE - Personal Adaptive Digital Library Environment*, 2001. http://www.is.informatik.uni-duisburg.de/courses/dortmund/pg/pg381/.

[Fow03a] FOWLER, MARTIN: *Patterns of Enterprise Application Architecture*. Addison-Wesley, 2003.

[Fow03b] FOWLER, MARTIN: *UML Distilled – A Brief Guide to the Standard Object Modeling Language*. Addison-Wesley, 2003.

[GHJV95] GAMMA, ERICH, RICHARD HELM, RALPH JOHNSON und JOHN VLISSIDES: *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[GHK00] GRUHN, VOLKER, DANIEL HEYMANN und MARC KLEINE: *Eine Architektur für Content-Management-Systeme auf Basis von XML*. In: *Proceedings der 1. Deutschen Tagung XML Heidelberg*, Seiten 149–167, 2000.

[GI 06] GI - GESELLSCHAFT FÜR INFORMATIK: *Architekturen*, 4. Januar 2006. http://www.gi-ev.de/informatiktage/die-workshop-themen/architekturen/.

[GJM91] GHEZZI, CARLO, MEHDI JAZAYERI und DINO MANDRIOLI: *Fundamentals of Software Engineering*. Prentice-Hall, 1991.

[Gr"05] GRÄBE, HANS-GERT: *Vorlesung Software aus Komponenten*, 2004/2005. http://ais.informatik.uni-leipzig.de/studium/vorlesungen/2004_ws/cw/2004w_cw_v_04.pdf.

[Haa06] HAASE, ARNO: *MDSD, Teil 1: Model Driven Software Development in der Praxis*. Java Magazin, Seiten 34–37, 2/2006.

[Hes04] HESS, THOMAS: *XML in Verlagen: Modewelle oder mehr? Vorstellung der Ergebnisse aus Phase 1-3 der Gemeinschaftsstudie von AKEP und dem Institut für Wirtschaftsinformatik und Neue Medien (LMU München)*. Arbeitskreis Elektronisches Publizieren – Börsenverein des Deutschen Buchhandels, 6. Oktober 2004. http://www.boersenverein.de/sixcms/media.php/686/XMLStudie.pdf.

[Hof79] HOFSTADTER, DOUGLAS R.: *Gödel, Escher, Bach, an Eternal Golden Braid*. Vintage Books, 1979.

[HS03] HENDERSON-SELLERS, BRIAN: *Unterstanding Metamodeling*. In: *22nd International Conference on Conceptual Modeling*, 2003. `http://www.er.byu.edu/er2003/slides/ER2003T1HendersonSellers.pdf`.

[Jec04] JECKLE, MARIO: *Scriptum zur Vorlesung XML*, 2004. `http://www.jeckle.de/vorlesung/xml/script.html`.

[Jes90] JESSEN, EIKE: *Vorlesung Systemprogrammierung*, 1990.

[Kam03] KAMPFFMEYER, ULRICH: *Wohin geht die Reise? Die Bedeutung von Dokumenten-Technologien für Wirtschaft und Gesellschaft*. In: *IIR Interflow Kongress*, 2003.

[Kay03a] KAY, MICHAEL: *XML Five Years On: A Review of the Achievements So Far and the Challenges Ahead*. In: *Proceedings of the 2003 ACM Symposium on Document Engineering*, Seiten 29–31, 2003.

[Kay03b] KAY, MICHAEL: *XSLT programmer's reference*. Wrox Press, 2003.

[Kep02] KEPSER, STEPHAN: *Ein Beweis zur Turing-Vollständigkeit von XSLT*, 2002. `http://tcl.sfs.uni-tuebingen.de/~kepser/slides/ws-herrsching-xslt.pdf`.

[KEVH06] KOLB, BERND, SVEN EFFTINGE, MARKUS VÖLTER und ARNO HAASE: *Graphical Modeling Framework: Sich selbst definieren*. iX, Seiten 152–156, 12/2006.

[KP02] KHOSLA, VINOD und MURUGAN PAL: *Real Time Enterprises. A Continuous Migration Approach.*, 2002. `http://www.kpcb.com/files/bios/RTEWHITEPAPER.pdf`.

[KR04] KHARE, ROHIT und ADAM RIFKIN: *The Origin of (Document) Species*, 2004. `http://www.ifindkarma.com/attic/papers/www/origin-of-species.html`.

[Kro97] KROHA, PETR: *Softwaretechnologie*. Prentice-Hall, 1997.

[Kru95] KRUCHTEN, PHILIPPE: *The 4+1 View Model of Architecture*. IEEE Software, 12(6):42–50, November 1995.

[Kru99] KRUCHTEN, PHILIPPE: *Der Rational Unified Prozess*. Addison-Wesley, 1999.

[KS05] KLINGER, CLAUDIA und RALPH SEGERT: *Das Online ABC*, 3.2.2005. `http://www.webwunder.de/asp/abc.asp?abfrage=Content`.

[LC01] LEUF, BO und WARD CUNNINGHAM: *The Wiki Way – Quick Collaboration on the Web*. Addison-Wesley, 2001.

[MA96] MALER, EVE und JEANNE EL ANDALOUSSI: *Developing SGML DTDs*. Prentice-Hall, 1996.

[MA03] MOCZAR, LAJOS und JEREMY ASTON: *Cocoon Developer's Handbook*. Sams Publishing, 2003.

[Miz02] MIZZARO, STEFANO: *Workshop on Personalization Techniques in Electronic Publishing on the Web: Trends and Perspectives*, 2002. `http://www.dimi.uniud.it/~mizzaro/AH2002/`.

[Mö03] MÖLLER, ERIK: *Das Wiki-Prinzip*, 2003. `http://www.heise.de/tp/r4/artikel/14/14736/1.html`.

[Nie04] NIEDERMEIER, STEPHAN: *Cocoon 2 und Tomcat – XML-Publishing mit dem Open-Source-Framework*. Galileo Press, 2004.

[NP] NUESCHELER, DAVID und PEETER PIEGAZE: *JSR 170: Content Repository for Java technology API*. `http://www.jcp.org/en/jsr/detail?id=170`.

[NT95] NONAKA, IKUJIRO und HIROTAKA TAKEUCHI: *The Knowledge-Creating Company*. Oxford University Press, New York, Oxford, 1995.

[Par72] PARNAS, DAVID L.: *On the Criteria To Be Used in Decomposing Systems into Modules*. In: *Communications of the ACM*, Band 15, Nummer 12, Dezember 1972.

[PBG04] POSCH, TORSTEN, KLAUS BIRKEN und MICHAEL GERDOM: *Basiswissen Softwarearchitektur*. dpunkt.verlag, Heidelberg, 2004.

[Pem05] PEMBERTON, STEVEN: *The ACELA Project: Aims and Plans*, 2005. `http://homepages.cwi.nl/~steven/acela/`.

[Pid03] PIDCOCK, WOODY: *What are the differences between a vocabulary, a taxonomy, a thesaurus, and ontology, and a meta-model?*, 2003. `http://www.metamodel.com/article.php?story=20030115211223271`.

[PRR99] PROBST, GILBERT, STEFFEN RAUB und KAI ROMHARDT: *Wissen managen*. Gabler-Verlag, Wiesbaden, 1999.

[Ren06] REN, ZIHENG: *Design und Implementierung eines Weblog-Hosting-Systems auf Basis von Apache Lenya/Cocoon*. Diplomarbeit, Technische Universität, Juni 2006.

[RM02] ROSENFELD, LOUIS und PETER MORVILLE: *Information Architecture for the World Wide Web*. O'Reilly, Beijing, Cambridge, Farnham, Köln, Paris, Sebastopol, Taipei, Tokyo, 2002.

[Roq01] ROQUES, PASCAL: *UML in Practice*. John Wiley & Sons, 2001.

[RR01] ROTHFUSS, GUNTHER und CHRISTIAN RIED: *Content Management mit XML*. Springer-Verlag, 2001.

[RW05] ROZANSKI, NICK und EOIN WOODS: *Software Systems Architecture – Working With Stakeholders Using Viewpoints and Perspectives*. Addison-Wesley, 2005.

[Sch01] SCHRAMM, DIRK: *Wie verwaltet man Inhalte? Anforderungen an XML-basierte Content Management Systeme im Electronic Publishing*. WiSt – Wirtschaftswissenschaftliches Studium, 11/2001. `http://wiim.wiwi.tu-dresden.de/wist/hefte/0111/artikel.html`.

[Sch09] SCHÖPF, THOMAS: *Modelle von Publikationsanwendungen*, 2009. `http://sunschlichter0.informatik.tu-muenchen.de/~schoepf/Model-PA.pdf`.

[SFB99] SÜSS, CHRISTIAN, BURKHARD FREITAG und PETER BRÖSSLER: *Meta-modeling for Web-Based Teachware Management*. In: *Proceedings of the Workshop on the World Wide Web and Conceptual Modeling*, Seiten 360–373, 1999.

[SG96] SHAW, MARY und DAVID GARLAN: *Software Architecture – Perspectives on an Emerging Discipline*. Prentice Hall, 1996.

[SG00] SOLEY, RICHARD und OMG STAFF STRATEGY GROUP: *Model Driven Architecture*, 2000. `ftp://ftp.omg.org/pub/docs/omg/00-11-05.pdf`.

[Sie04] SIEDERSLEBEN, JOHANNES: *Moderne Softwarearchitektur – Umsichtig planen, robust bauen mit Quasar*. dpunkt.verlag, 2004.

[Skå] SKÅRHØJ, KASPER: *Sites made with TYPO3*. `http://typo3.org/about/sites-made-with-typo3/`.

[Skå05a] SKÅRHØJ, KASPER: *TYPO3 CMS: History*, 2005. `http://www.typo3.com/History.1268.0.html`.

[Skå05b] SKÅRHØJ, KASPER: *TYPO3.org: Getting Started*, 2005. `http://typo3.org/documentation/document-library/doc_tut_quickstart/`.

[Skå05c] SKÅRHØJ, KASPER: *TYPO3.org: Modern Template Building, Part 1*, 2005. `http://typo3.org/documentation/document-library/doc_tut_templselect/Page_Tree_and_Templa/`.

[Skå05d] SKÅRHØJ, KASPER: *TYPO3.org: What are extensions?*, 2005. `http://typo3.org/extensions/what-are-extensions/`.

[SMB94] SPERBERG-MCQUEEN, C.M. und LOU BURNARD: *A Gentle Introduction to SGML*, 1994. `http://www.isgmlug.org/sgmlhelp/g-index.htm`.

[VGB01] VIDGEN, RICHARD, STEVE GOODWIN und STUART BARNES: *Web Content Management*. In: *Proceedings of the 14th Bled Electronic Commerce Conference*, 2001.

[Vir06] VIRTUELLES SOFTWARE ENGINEERING KOMPETENZZENTRUM: *Glossareintrag: Softwarearchitektur*, 4. Januar 2006. `http://www.software-kompetenz.de/?10753`.

[Weg02] WEGNER, HOLM: *Analyse und objektorientierter Entwurf eines integrierten Portalsystems für das Wissensmanagement*. Doktorarbeit, Technische Universität Hamburg-Harburg, 2002.

[Wei72] WEINBERG, GERALD M.: *The Psychology of Computer Programming*. Van Nost. Reinhold, 1972.

[Zus99] ZUSE, HORST: *Geschichte der Programmiersprachen*, 1999. Forschungsbericht des Fachbereichs Informatik der Technischen Universität Berlin Nr. 1999-1. ISSN 1436-9915.