Publikationsprozesse in digitalen Bibliotheken - Mittel und Methoden der Autorenunterstützung

Diplomarbeit

Universität Rostock
Fakultät für Informatik und Elektrotechnik
Institut für Informatik

vorgelegt von	:	Sebastian Schick
Matrikelnummer	:	201667
geboren am	:	11.07.1980 in Rostock
Erstgutachter	:	Prof. Dr. Andreas Heuer
Zweitgutachter	:	Prof. Dr. Peter Forbrig
Betreuer	:	Dr. Holger Meyer
		Dipl.-Inf. Anja Schaar
		Dipl.-Inf. Ilvio Bruder
Abgabedatum	:	24. Oktober 2005

Zusammenfassung

In dieser Arbeit wurde ein Publikationsprozess für die Modellierung von Multimediadokumenten in digitalen Bibliotheken entwickelt. Grundlage für die Betrachtungen waren die Multimedia Dokumentenmodelle *MPEG-7* und *METS* . Ausserdem wurde der Content-Life-Cycle von Content-Management-Systemen untersucht.

Das Ergebnis der Arbeit ist ein Java Framework für den Publikationsprozess. Die Service orientierte Architektur des Frameworks verwendet das Meta-Framework *Spring* und baut auf dem Framework *x18p* auf. Das Framework wird prototypisch in der digitalen Bibliothek der Universitätsbibliothek Rostock *@libri* umgesetzt, welches auf dem Content-Repository *MyCoRe* basiert.

Abstract

In this thesis a publication process for the modeling of multimedia documents in digital libraries was generated. The basis of the view where the multimedia document models *MPEG-7* and *METS* . Moreover the content life cycle of content management systems was investigated.

The result of this thesis is a java framework for the publication process. The service orientated architecture of the framework uses the meta-framework *Spring* and is based on the framework *x18p* . A prototype of the framework is implemented in the digital library *@libri* , which is part of the library of the university of Rostock. *@libri* based on the content repository *MyCoRe* .

CR-Klassifikation

H.3.3 Information Search and Retrieval
H.3.5 Online Information Services
H.3.6 Library Automation
H.5.1 Multimedia Information Systems

Key Words

digital library, multimedia document, publication process, multimedia authoring

Danksagung

An dieser Stelle bedanke ich mich bei Herrn Dr. Holger Meyer für die Überlassung des Themas und die ausgezeichnete Betreuung dieser Arbeit.

Weiterhin bedanke ich mich bei der Fachreferentin für Informatik der Universitätsbibliothek Rostock, Frau Dipl. Inf. Anja Schaar. Insbesondere für ihre Unterstützung und für wichtige Hinweise bei der Arbeit mit der digitalen Bibliothek *@libri* .

Ebenfalls möchte ich mich bei Herrn Dipl. Inf. Ilvio Bruder für seine Unterstützung bei der Anfertigung der Arbeit bedanken.

Inhaltsverzeichnis

1 Einleitung

Digitale Bibliotheken enthalten neben einfachen Textdokumenten zunehmend auch komplex strukturierte multimediale Dokumente. Multimedia Dokumente können neben Text-, Audio-, Bild-, 2D-, 3D-Daten auch Beziehungen zwischen einzelnen Objekten und weitergehende Anforderungen enthalten. Anforderungen an Multimedia Dokumente sind z.B. eine hohe Wiederverwendbarkeit oder die darstellungsneutrale Beschreibung. Objekte in Multimedia Dokumenten können z.B. in räumliche oder zeitliche Beziehung gesetzt werden. Bei der räumlichen Beziehung geben Positionen an, wo die Objekte auf der Anzeige angezeigt werden. Zeitliche Beziehungen geben an, wann und wie lange Objekte angezeigt werden. Vernetzungsstrukturen zwischen den Objekten werden beispielsweise mit Hyperlinks dargestellt. Für die Beschreibung dieser Zusammenhänge existieren verschiedene Dokumentenmodelle. Viele Modelle sind auf spezielle Anwendungsgebiete optimiert. Bei der Erstellung von Multimediadokumenten werden Autoren oftmals nur bei der Erstellung der Teildokumente unterstützt. Für das Zusammenfassen der Teildokumente in Multimediadokumenten fehlt oft die Unterstützung durch entsprechende Werkzeuge. Außerdem werden die unterschiedlichen Medientypen häufig mit unterschiedlichen Systemen erstellt.

Um diese Lücke zu schließen, wird ein Publikationsprozess für komplex strukturierte Multimediadokumente eingeführt. Dafür muss der allgemeine Erstellungsprozess, in digitalen Bibliotheken an die speziellen Anforderungen für Multimediadokumente, angepasst werden. Zentrale Punkte sind dabei die Integration existierender Werkzeuge, die [semi-] automatische Ableitung von Metadaten und die Aufarbeitung der Dokumenteninhalte für eine medienspezifische Deskribierung. Eine flexible Sicht auf die zu publizierenden Inhalte ist genauso von Interesse wie passende Suchmöglichkeiten.

Ausgehend von den Anforderungen an die Funktionalität einer digitalen Bibliothek, soll in dieser Arbeit ein Publikationsprozess für die Erstellung von komplex strukturierten Multimediadokumenten entwickelt werden. Auf der Basis von geeigneten Multimedia Dokumentenmodellen und Betrachtungen des Content-Life-Cycles in Content-Management-Systemen, wird dazu eine Konzeption erstellt. Der Publikationsprozess ist die Grundlage für die Entwicklung eines Java Frameworks. Durch die Definition eines Frameworks, soll ein hohes Maß an Wiederverwendbarkeit geboten werden.

In Form einer Proof-of-Concept Implementierung soll das Framework in die digitale Bibliothek der Universitätsbibliothek Rostock integriert werden. Die Bibliothek basiert auf dem Open-Source Content-Repository *MyCoRe* . Die in der Arbeit entwickelten Ansätze und Konzepte sollen anhand eines Dokumentenmodells für Bildserien demonstriert werden. Das Dokumentenmodell wird mit Hilfe des *MyCoRe* Datenmodells umgesetzt.

2 Multimedia Dokumentenmodelle

Komplex strukturierte Multimediadokumente können heute mit verschiedenen Multimedia Dokumentenmodellen beschrieben werden. Die Modelle unterscheiden sich teilweise erheblich in ihren Möglichkeiten, Beziehungen, Strukturen, Hierarchien etc. zu realisieren.
Um die Modellierungsmöglichkeiten dieser Multimedia Dokumentenmodelle zu beschreiben und eine Übersicht darüber zu geben, wird ein Katalog von Anforderungen vorgestellt. Die Anforderungen beschreiben notwendige Eigenschaften für Multimedia Dokumentenmodelle . Anhand dieser Eigenschaften werden die aktuellen Modelle *MPEG-7* und *METS* für Multimediadokumente untersucht. Die Untersuchung soll einen Überblick geben, wie die Modelle bestimmte Anforderungen umsetzen.

MPEG-7 wurde für diese Arbeit ausgewählt, weil das Dokumentenmodell sehr umfangreich in seinen Beschreibungsmöglichkeiten ist. Es stellt einen aktuellen Standard für die Ergänzung von Multimedia Daten mit Metadaten dar. *METS* ist ein Standard, der von der Library of Congress speziell für digitale Bibliotheken entwickelt wurde. Er wurde ausgewählt, weil er einen aktuellen Standard im Bereich digitaler Bibliotheken darstellt.

Anhand der genannten Anforderungen werden die vorgestellten Dokumentenmodelle analysiert. Hierbei soll untersucht werden, inwieweit eine Unterstützung der geforderten Anforderungen vorhanden ist und wie sie umgesetzt wird. Zusätzlich werden spezielle Möglichkeiten der jeweiligen Modelle vorgestellt.
Die gewonnen Erkenntnisse über die Umsetzung der Multimedia Dokumentenmodelle soll bei der Konzeption des Publikationsprozesses für digitale Bibliotheken umgesetzt werden.
Die Definition für Multimediadokumente wird in dieser Arbeit aus [Bol01] übernommen.

> A multimedia document is a media element that forms the composition of continuous and discrete media elements into a logically coherent multimedia unit.
>
> A multimedia document is an instantiation of a multimedia document model that provides the primitives to capture all aspects of a multimedia document.
>
> A multimedia document format determines the serialized representation of a multimedia document for the document's exchange.

2.1 Metadaten

2.1.1 Definition Metadaten

Als Metadaten (*"Daten über Daten "*) werden allgemein strukturierte Daten bezeichnet, die andere Daten beschreiben. Sie geben Grundinformationen über Dokumente wie z.B. die Angaben über den Autor, den Titel oder den Zeitpunkt der Veröffentlichung an. Der effektive Einsatz von Metadaten setzt, genau wie bei bibliothekarischen Regelwerken, eine Standardisierung voraus.

In Tabelle 2.1 wird ein Überblick über eine mögliche Klassifizierung von Metadaten gegeben.

Klassifikation	Metadaten	Beispiele
inhaltsbeschreibend (interpretierend)	kontextbeschreibend	Indexvokabular, Ontologien, Thesaurus
	kontextbezogen	Identifikation, Raum- und Zeitdaten
	objektbeschreibend nicht textuell	Gegenstände, Personen, Eindrücke, Aktivitäten, Titel
	objektbeschreibend textuell	Annotation, Drehbuch, Untertitel
inhaltsbezogen (nicht interpretierend)	Feature	Farbverteilung, Textur, Klangdynamik, Form
	Segmentspezifikation	Anfang und Ende einer Videoszene, Umriss eines Bildausschnitts
inhaltsunabhängig	präsentationsbezogen	QoS, Auflösung, Layout
	aufnahmebezogen	Urheber, Aufnahmegerät
	speicherbezogen	Medientyp, Speicherformat, Speicherort

Tabelle 2.1: Klassifikation von Metadaten [Sch02]

Die Merkmale für die Klassifikation von Metadaten aus Tabelle 2.1 werden im Folgenden detaillierter beschrieben:

- **Inhaltsbeschreibende** Metadaten beschreiben die Semantik von Objekten. Dabei wird der Inhalt interpretiert. Sie können verwendet werden, um direkt nach Objekten zu suchen.
- **Inhaltsbezogen** Metadaten beschreiben den Inhalt von Objekten auf einer semantisch niedrigen Stufe. Sie lassen sich in einer Anfrage indirekt für eine Ähnlichkeitssuche verwenden. Eine Unterscheidung wird in Features und Segmentspezifikation vorgenommen.

- **Inhaltsunabhängige** Metadaten sind für die Verwaltung und korrekte Identifizierung von Objekten notwendig. Dies kann z.B. das Speicherformat eines Objektes sein.

Inhaltbeschreibenden Metadaten werden in weitere Klassifikationen unterteilt:

- **Kontextbeschreibende** Metadaten sind allgemeine Daten, die für die Objektbeschreibung notwendig sind. Sie beschreiben immer eine Menge von Objekten. Häufig für sie für den Abgleich von Objekten verwendet und erlauben zusätzlich inhaltliche Vergleiche von Objektbeschreibungen.

- **Kontextbezogene** Metadaten ergeben sich erst aus dem Kontext einer Kollektion von Objekten. Als Beispiel können Beziehungen zwischen Objekten existieren, die eine zeitliche oder räumliche Anordnung darstellen.

- **Objektbeschreibende** Metadaten beschreiben den Inhalt von Objekten. Ein Beispiel ist ein Foto mit Personen, deren Namen und Addressen zusätzlich verwaltet werden.

Die Objekte können unterschiedlich stark strukturiert sein. Wichtig ist der Zusammenhang zwischen dem Grad der Strukturiertheit von Objekten und dem Umfang des Strukturteils der Metadaten. Um so tiefer strukturiert ein Objekt ist, um so komplexer wird der Strukturbereich der Metadaten. Strukturen für das Unterstützen von Suche, Manipulation und für Beziehungen zwischen Objekten werden dadurch umfangreicher. Bei der Definition von Metadaten muss deshalb immer nach neuen Ansätzen für die Beschreibung gesucht werden, da die Dokumente immer komplexer werden. Nach [Göt] wird dieses Thema immer wichtiger für Bibliotheken, die sich nicht mehr nur noch auf ihre herkömmlichen Regelwerke berufen können.

2.1.2 Gewinnung von Metadaten

Metadaten können auf verschiedenen Weise erzeugt werden. In Abhängigkeit des verwendeten Dokumentenmodells und den technischen Möglichkeiten, werden drei Verfahren unterschieden:

- automatisch
- semiautomatisch mit manueller Nachbesserung
- manuell

Bei dem *automatischen* Erzeugen von Metadaten, werden inhaltsbezogene Metadaten extrahiert. Dazu gehören z.B. Farbhistogramme, Auflösung oder die Größe eines Bildes. Auf einer sehr spezialisierten Anwendungsebene besteht die Möglichkeit komplexe Merkmale zu erzeugen.

Die *manuelle* Extraktion von inhaltsbeschreibenden Metadaten wird häufig durch spezielle Personen vorgenommen. Es ist technisch schwer umsetzbar, diese Daten automatisch zu extrahieren. In Bibliotheken kann die Vergabe z.B. durch entsprechendes Personal durchgeführt werden. Es werden oft Stichworte vergeben, die vorgegebenen Attribut und Anwendungs-Vokabularien entsprechen. Wichtig für die spätere Indizierung der Metadaten ist die einheitliche Verwendung von Begriffen. Metadaten die das gleiche beschreiben, müssen auch gleich lauten. Als Beispiel können die Begriffe oval und abgerundet genannt werden. Folgende Nachteile ergeben sich bei einer manuellen Extraktion:

- hoher Aufwand
- hohe Fehleranfälligkeit, dadurch ist eine Inkonsistenz nicht vermeidbar
- es kann nur auf vorher erstellten Metadaten eine Anfrage gestellt werden

Die *semiautomatische* Extraktion verbindet die automatischen und manuellen Techniken. Die automatischen Verfahren bieten oft keine hundertprozentige Sicherheit und müssen manuell korrigiert werden. Diese Vorgehensweise ermöglicht es, schnell Metadaten zu erzeugen, die durch die nachträgliche manuelle Verarbeitung eine hohe Qualität haben.[Heu]

2.2 Anforderung an ein Multimedia Dokumentenmodell

2.2.1 Überblick über Anforderungen

Ein Dokumentenmodell beschreibt nach [EF00] die logische Struktur eines Dokumentes. Für die Modellierung von Multimediadokumenten müssen entsprechende Dokumentenmodelle bestimmte Eigenschaften erfüllen.
Im Folgenden werden alle wichtigen Eigenschaften aufgezählt und erläutert, die für diese Arbeit von interesse sind. Die genannten Anforderungen sollen ein Grundgerüst für die Analyse der Modelle darstellen.

In der Dissertation von S. Boll zu dem Thema *ZyX – Towards flexible multimedia document models for reuse and adaptation* wird das Multimedia Dokumentenmodell *ZyX* vorgestellt [Bol01]. Im Zuge der Dissertation wurden verschiedene Multimedia Dokumentenmodelle untersucht. Für diese Untersuchungen wurden verschiedene Kriterien erarbeitet, die Multimedia Dokumentenmodelle erfüllen müssen. Diese Kriterien werden in dieser Arbeit weiter verwendet, da sie ein gute Ausgangsbasis bieten.

Die in [Bol01] aufgeführten Anforderungen für Multimedia Dokumentenmodelle werden grundsätzlich in einfach und erweitert unterteilt.
Für die Unterstützung des Publikationsprozesses in einer digitalen Bibliothek werden in

dieser Arbeit zusätzlich Anforderungen an die Modelle gestellt, die entsprechend angegeben werden.

Einfache Anforderungen sind unterteilt in die Möglichkeiten der *zeitlichen*, *räumlichen* und *interaktiven* Modellierung. Die erweiterten Eigenschaften der Modelle werden mit

- Wiederverwendbarkeit
- Anpassbarkeit an Nutzer Bedürfnisse
- und die darstellungsneutrale Beschreibung

der Multimediadokumente formuliert.

Ein Multimedia Dokumentenmodell sollte zusätzlich zu den in [Bol01] angegebenen einfachen Anforderungen auch die Modellierung von Bildern, Texten, Audio und Videos unterstützen. Hierbei sollen Elemente beschrieben werden, die nur einem Medientyp zuzuordnen sind. Die genaue Struktur und der Inhalt ist von der Applikation abhängig und wird daher hier nicht weiter betrachtet.
Für den Einsatz in der digitalen Bibliothek Rostock werden außerdem folgende Anforderungen für Dokumentenmodelle untersucht:

- Beschreibung von komplexen Beziehungen zwischen Objekten,
- Vererbung von Informationen,
- Unterstützung von bestehenden Metadaten Formaten wie z.B. Dublin Core (DC),
- einfaches Erstellen durch lokale Anwendungen
- die Möglichkeit der Suche in den Metadaten

2.2.2 Anforderungen im Detail

In diesem Abschnitt wird eine detaillierte Beschreibung der in Abschnitt 2.2.1 genannten Anforderungen an Multimedia Dokumentenmodelle vorgenommen. Die Definition der einfachen und erweiterten Anforderungen ist dabei aus [Bol01] entnommen.
Die einfachen Anforderungen werden dort als minimale Voraussetzung für ein Multimedia Dokumentenmodelle gesehen. Diese Voraussetzungen müssen erfüllt sein, um Multimedia Applikationen grundsätzlich zu unterstützen. Für eine Multimedia Unterstützung sind zusätzlich die erweiterten Anforderungen notwendig. Diese Unterteilung baut auf den in ([Bol01], Kapitel 2) gezeigten Erfahrungen auf.

Einfache Anforderungen

Zeitliches Modell

Eine wesentliche Anforderung an Multimediadokumente stellt die Möglichkeit der zeitlichen Modellierung dar. Es wird zwischen vier verschieden Typen unterschieden:

- Das **Zeitpunkt-Modell** gibt die zeitliche Darstellung mit Hilfe von Punkten an. Die Punkte definieren auf der Zeitachse den Beginn und das Ende der Präsentation eines Elementes. Zeitpunkte können hier $<$ (kleiner), $>$ (größer) oder $=$ (gleich) sein.
- Das **Zeitintervall-Modell** definiert, ähnlich wie beim Zeitpunkt Modell, zeitliche Beziehungen durch Zeitpunkte, die aber in Intervallen der Form $[t_i, t_j]$ angegeben sind. Hierbei können Beziehungen wie *zwischen* oder *gleich* ausgedrückt werden. In [All83] werden 13 verschiedene Beziehungen unterschieden, auf die hier aber nicht weiter eingegangen werden soll. Zu erwähnen ist, dass die Zeitintervalle nicht variabel sein können, und somit nicht auf unbekannte Intervalle anwendbar sind.
- Im **ereignisgesteuerten Modell** wird der zeitliche Verlauf der Anzeige durch Ereignisse gesteuert. Ein Ereignis kann z.B. das Ende eines Videos oder einer Audio Wiedergabe sein. An das Ereignis ist dann wiederum der Beginn einer neuen Präsentation gekoppelt.
- Im Scriptsprachen-Modell können je nach Art der Skriptsprache, komplexe zeitliche Bedingungen und Abhängigkeiten für die Präsentation von Elementen definiert werden.

Räumliches Modell

In Multimediadokumenten ist nicht nur die zeitliche Anordnung von Elementen von Interesse. Elemente sollen auch räumlich in Bezug zueinander gestellt werden können. Diese Darstellung kann durch verschieden Modelle ausgedrückt werden:

- absolute Positionierung,
- Richtungsrelationen und
- topologische Relationen

Bei der absoluten Positionierung werden die Elemente auf einem Anzeigefeld an einer festen absoluten Position angezeigt. Diese Position lässt sich durch ein Koordinatenpaar angeben, das gegebenenfalls durch eine dritte Koordinate für die Tiefe erweitert werden kann. Es wird außerdem davon ausgegangen, dass die Position zum einen absolut in Bezug auf den Koordinatenursprung und zum anderen absolut zu einem anderen Objekt sein kann.
Die Richtungsrelationen stellen eine weitere Möglichkeit dar, Objekte räumlich anzuordnen. Es werden Beschreibungen wie *north, north-west* usw. verwendet.

Die topologische Relation kann in 8 unterschiedliche Relationen unterteilt werden: *disjoint, meet, overlap, covers, coverd-by, contains, inside* and *equal*.

Interaktion

Eine weitere wichtige Anforderung an Multimediadokumente ist die Interaktion innerhalb der Dokumente. Der Nutzer muss die Möglichkeit haben, selbstständig zu entscheiden, welchen Weg er innerhalb der Präsentation geht. Durch Hyperlinks und Menüs ihm u.a. eine navigierende Interaktion ermöglicht. Eine weitere wichtige Möglichkeit ist die Navigation in Teilbereiche von Elementen. Unterstützt ein Modell diese Navigation, können Teilbereiche aus Bildern oder Szenen in Videos referenziert werden.

Die in [Bol01] betrachtete interaktive Modellierung geht davon aus, dass entweder feste zeitliche (z.B. Start- und Endpunkt in einem Video) oder räumliche Objekte (z.B. ein Bild innerhalb der Präsentation) ausgewählt werden. Mit *anchor* oder *area* Elementen ist es so möglich, Objekte in zeitliche oder räumliche Fragmente zu unterteilen. Diese statische Auswahl von Regionen, die es nicht ermöglicht Strukturen und semantische Informationen der Objekte mit zu betrachten, reicht für heutige Multimediadokumente/Anwendungen nicht mehr aus.
Es ist zusätzlich notwendig, dass Raum-Zeit Objekte verwendet werden können. Damit sind z.B. Regionen innerhalb einer Szene in einem Video gemeint, die sich mit einem Link belegen lassen.

Erweiterte Anforderungen

Dokumentenmodelle sollen Wiederverwendbarkeit, Anpassbarkeit und eine darstellungsneutrale Beschreibung unterstützen. Diese drei Kriterien ermöglichen es modular und kontext-abhängig Multimediadokumente zu erzeugen.

Wiederverwendbarkeit

Die Wiederverwendbarkeit ist wichtig bei Autorenprozessen, um Zeit bei der Erstellung von Multimediadokumenten einsparen zu können. Ein weiterer Vorteil liegt darin, dass Elemente nicht mehrmals erzeugt werden müssen und so von anderen Nutzern wiederverwendet werden können. Für eine detailliertere Betrachtung wird die Wiederverwendbarkeit in drei Dimensionen unterteilt. Die *Granularität* der Wiederverwendung, die *Art* der Wiederverwendung und *Auswahl und Identifikation* der wiederverwendbaren Komponenten.

Die **Granularität**
beschreibt was von einem Multimediadokument-Modell wiederverwendet werden kann. Hier sind drei Ebenen zu Unterscheiden:

- Wiederverwendung von kompletten Dokumenten
- Wiederverwendung von Fragmenten eines Multimediadokumentes, wie z.B. einzelne Szenen
- Wiederverwendung von individuell atomaren Elementen

Für die drei oben genannten Granularitäten werden zwei verschiedene Arten von Wiederverwendbarkeit unterschieden. Die identische Wiederverwendbarkeit, bei der alle Komponenten wiederverwendet werden. Dies sind die zeitlichen, räumlichen, design und interaktions Beziehungen, die vom Autor angegeben wurden. Bei der strukturellen Wiederverwendung wird nur der Strukturteil der Elemente verwendet. Layout Informationen werden hier nicht mit übernommen.

Auswahl und Identifikation

Um Elemente auswählen zu können und vorher zu identifizieren, werden Metadaten über sie benötigt. Diese Metadaten müssen klassifiziert werden, um sie zu indizieren und abzufragen. Sind diese Voraussetzungen erfüllt, kann ein Autor mit Hilfe dieser Metadaten, Elemente ausfindig machen und weiter verwenden.

Anpassbarkeit

Die Anpassbarkeit kann in zwei Bereiche eingeteilt werden. Die Anpassung der Präsentation der Multimediadokumente aufgrund technischer Gegebenheiten, wie z.B. die Bandbreite eine Übertragungsleitung oder die Bildschirmgröße der Anzeige. Sowie die Anpassung in Abhängigkeit vom Kontext. Im letzteren Fall soll die Präsentation an die Interessen des Nutzers angepasst werden. Ein Professor könnte einen anderen Betrachtungswinkel als ein Student auf einen Artikel haben.

Darstellungsneutrale Beschreibung

Ein Multimediadokument sollte unabhängig von seiner späteren Darstellung gespeichert werden. Hierdurch ist es möglich, dass Multimediadokument in die zur Anzeige verwendeten Formate zu konvertieren. Datenverluste bei der Konvertierung sind nicht mehr von Interesse.
Für die Anforderung an die darstellungsneutrale Beschreibung ist es wichtig, den Grad der Unabhängigkeit der Beschreibung vom Layout festzulegen. Dieser Grad wird mit dem semantischen Level des Multimedia Dokumentenmodells beschrieben. Ein Dokumentenmodell beschreibt ein Dokument auf einem hohen semantischen Level, wenn der Inhalt mehr als die Präsentation beschrieben wird. Ein Modell, das ein hohes semantisches Level hat, unterstützt viele Spezifikations-Strukturen, die gut erreichbar sein müssen.

Anforderungen an Modelle speziell für digitale Bibliotheken

Ein Dokumentenmodell muss besondere Anforderungen erfüllen, wenn es für den Einsatz in digitalen Bibliotheken verwendet werden soll. Im Folgenden werden die oben genannten Anforderungen für digitale Bibliotheken definiert und beschrieben.

Für die Beschreibung von komplexen Beziehungen zwischen Objekten, muss ein Dokumentenmodell eine große Auswahl an Modellierungskonzepten zur Verfügung stellen. Dabei stehen das Modellieren von Objekt-Reihenfolgen oder Hierarchien im Vordergrund der Betrachtungen. Folgende Beziehungen sind ebenfalls wichtig:
Die **Attributbeziehung** ermöglicht das zuordnen von Multimedia Content zu einem Element. Ein Bild kann hier z.B. ein beschreibendes Attribut für ein Element sein.

Zusammengesetzte Objekte werden durch **Komponentenbeziehungen** dargestellt. Ein Buch kann aus verschiedenen einzelnen Kapiteln zusammengesetzt sein.

Um Informationen mit verschiedenen Medientypen darzustellen, wird die **Substitutionsbeziehung** verwendet. Informationen aus einem Text können gesprochen oder geschrieben sein.

Semantische Äquivalenz von Objekten. Objekte sind semantisch äquivalent, wenn sie den gleichen Inhalt haben, obwohl es sich um verschiedene Varianten des selben Objekts handelt. Ein Beispiel sind Bilder. Hier werden oft die Bilder in originaler Größe und als Thumbnail abgelegt. [HS]

Die **Vererbung** von Struktur und Verhalten ermöglicht den Aufbau von Hierarchien.

Metadaten Formate wie Dublin Core (DC), MetaDiss oder OAI 2.0 sollen ebenfalls unterstützt werden.

Das **Suchen** in Metadaten und Content ist ein weiterer wichtiger Faktor für digitale Bibliotheken. Die Metadaten sollten möglichst gut indizierbar sein.

Wichtig ist auch eine **einfache Erstellung** durch lokale Anwendungen.

Weiterhin müssen eine Publishing-Schnittstelle für Integration und Interaktion (Schnittstellen: WebDAV, FTP, etc.) mit dedizierten Autorenwerkzeugen (Adobe, Corel, etc.) vorhanden sein.

2.3 Multimedia Content Description Interface (MPEG-7)

2.3.1 Überblick

Der *MPEG-7* Standard beschreibt audiovisuelle Inhalte. Er wurde ausgewählt, da er nicht nur auf einen bestimmten Anwendungsbereich ausgelegt ist, vielmehr besteht die Möglichkeit Inhalte aus verschiedenen Bereichen auf gleiche Art und Weise zu beschreiben. Es werden eine Vielzahl von Möglichkeiten für das Beschreiben unterschiedlicher multimedialer Anwendungen zur Verfügung gestellt. Der Standard soll stellvertretend für Multimedia Dokumentenmodelle vorgestellt werden. Eine ausführliche Einführung in den Standard kann in [SS02b] gefunden werden, die die Grundlage dieses Abschnitts darstellt.

Der Standard wird von der *Moving Picture Expert Group* (MPEG) entwickelt, die genaue Bezeichnung lautet *Content Description Interface*. Weiterhin wurde *MPEG-7* als ISO/IEC 15938 standardisiert. Anders als MPEG-1, MPEG-2 und MPEG-4, beschreibt

MPEG-7 nicht den Inhalt an sich (die Bits), sondern die Informationen über den Inhalt.

Die folgenden Definitionen aus dem Requirements document [MPE05] sollen die Verwendenten Grundelemente kurz darstellen.

Der *descriptor* ist ein Tool, das Elemente im Standard beschreibt. Die Definition lautet:

> "A Descriptor (D) is a representation of a Feature. A Descriptor defines the syntax and the semantics of the Feature representation."

Die *Description Schemes* werden aus Deskriptoren und anderen *Description Schemes* erstellt. Sie stellen die Struktur dar und werden im Requirements document folgendermaßen angegeben:

> "A Description Scheme (DS) specifies the structure and semantics of the relationships between its components, which may be both Descriptors and Description Schemes."

Für die Beschreibung der Deskriptoren und der *Description Schemes* wird die *Description Definition Language* verwendet die im Folgenden definiert wird:

> "The Description Definition Language (DDL) is a language that allows the creation of new Description Schemes and, possibly, Descriptors. It also allows the extension and modification of existing Description Schemes."

Weiterhin ist in [SS02b] eine Definition für *System Tools* angegeben:

> "Tools related to the binarization, synchronization, transport and storage of descriptions, as well as to the management and protection of intellectual property."

Die oben vorgestellten *Description Schemes* beschreiben visuelle, audio und multimedia *Description Tools*. Die Deskriptoren beschreiben low-level features wie z.B. Texturen, Bewegung oder Farben, und high-level features wie z.B. abstrakte Konzepte oder Content-Management-Prozesse. Die low-level features sollen vorrangig automatisch erzeugt werden, im Gegensatz dazu müssen die high-level features oft weiterhin vom Menschen erstellt werden. Um komplexe Beziehungen zwischen Deskriptoren oder *Description Schemes* zu beschreiben, werden *Description Schemes* verwendet.

Der Standard ist für das bessere Verständnis und für eine bessere Übersicht in acht verschiedene Teile gegliedert. Diese Aufteilung ist nach [SS02b] aus der Standardisierungsphase entstanden. Es soll zusätzlich die Möglichkeit bestehen, die einzelnen technologischen Cluster des Standards selbständig anzuwenden.

Die Einteilung nach [SS02b] wird im Folgenden kurz vorgestellt:

- Part 1 - *Systems*: Der System Teil beschreibt die Tools die die Funktionalität des *MPEG-7* Standards bereitstellen. Diese sind das Bereitstellen, Synchronisieren und Verwalten von multimedia Beschreibungen. Außerdem wird die Darstellung binär und textuell unterstützt.

- Part 2 - *Description Definition Language*: Die DDL ist die Sprache, durch die descriptor schemes und Deskriptoren beschrieben werden. Sie wird in XML Schema Sprache definiert. Zusätzlich zu den XML Schema Struktur Komponenten und Datentypen ist die DDL mit den eigenen *MPEG-7* Datentypen Array und Matrix erweitert worden. Deswegen wird ein spezieller *MPEG-7* Parser für *MPEG-7* Dokumente benötigt, wobei ein xml Parser die beiden Datentypen einfach ignorieren kann.

- Part 3 - *Visual*: Deskriptoren (color, texture, shape und motion) und *Description Schemes* beschreiben visuelle Informationen.

- Part 4 - *Audio*: Deskriptoren und *Description Schemes* beschreiben audio Informationen, wobei low-level Tools auf alle Audiosignale angewendet werden können. Application-specific Tools sind dagegen mehr auf einen Applikationsbereich spezialisiert.

- Part 5 - *Generic Entities and Multimedia Description Schemes (MDS)*: Hier werden alle Deskiproren und *Description Schemes* beschrieben die visuelle Informationen beschreiben. Weiterhin werden Informationen für Multimedia features beschrieben.

- Part 6 - *Referece Software*: Diese software beschreibt die Software zu den einzelnen Tools.

- Part 7 - *Conformance Testing*: in diesem Teil werden die Regeln und Funktionen festgelegt, um *MPEG-7* Beschreibungen zu testen.

- Part 8 - *Extraction and use of MPEG-7 Descriptions*: Im letzten Punkt werden Informationen über die Extraktion und Verwendung von *Description Tools* gegeben...

In Abbildung 2.1 ist ein Beispiel für ein *MPEG-7* Dokument dargestellt. Es beschreibt ein Auto in einem Bild.

2.3.2 Konzepte und Besonderheiten

In diesem Abschnitt soll, in Anlehnung an [SS02b], ein kurzer Überblick über spezielle Konzepte im *MPEG-7* Standard geben werden.

Der Standard selbst bietet die Möglichkeit durch die in Abschnitt 2.3.1 erwähnten Deskriptoren individuelle Eigenschaften zu beschreiben, die mit den *Description Schemes* zu komplexen Beschreibungen zusammen gefasst werden können. Mit Hilfe der DDL

```
<Mpeg7>
    <Description xsi:type="SemanticDescriptionType">
        <Semantics>
            <Label>
                <Name> Auto </Name>
            </Label>
            <Definition>
                <FreeTextAnnotation> Ein Auto mit vier Rädern
                    </FreeTextAnnotation>
            </Definition>
            <MediaOccurrence>
                <MediaLocator>
                    <MediaUri> bild.jpg </MediaUri>
                </MediaLocator>
            </MediaOccurrence>
        </Semantics>
    </Description>
</Mpeg7>
```

Abbildung 2.1: Beispiel *MPEG-7*

ist es möglich die im Standard vorgegebenen Deskriptoren und *Description Schemes* einzuschränken oder zu erweitern.
In Abbildung 2.2 wird der allgemeine Aufbau eines *MPEG-7* Dokuments gezeigt. Die vorhandenen *Multimedia Description Schemes* (MDS) sind in verschiedenen *top-level* Elementen zusammengefasst. Das *root* Element befindet sich immer an oberster Stelle.

Für die Erstellung von *MPEG-7* Dokumenten stellt der Standard eine Reihe von *Basic Elements* zur Verfügung. In Form einer Bibliothek werden häufig verwendete Elemente für die Beschreibung von Audio, Visuellen und Multimedia *Description Schemes* zusammengefasst. In [SS02b] sind die *Basic Elements* in *basic data types*, *linking und localization tools* und *basic tools* unterteilt, die hier im Überblick dargestellt werden.

Basic Data Types

Für die Beschreibung von Multimediadokumenten wird im *MPEG-7* Standard die durch die DDL zur Verfügung gestellten Datentypen erweitert mit:

- Interger und Real Zahlen
- Vektoren und Matrizen
- Wahrscheinlichkeitsverteilung, die mit Vektoren und Matrizen beschrieben wird
- Strings

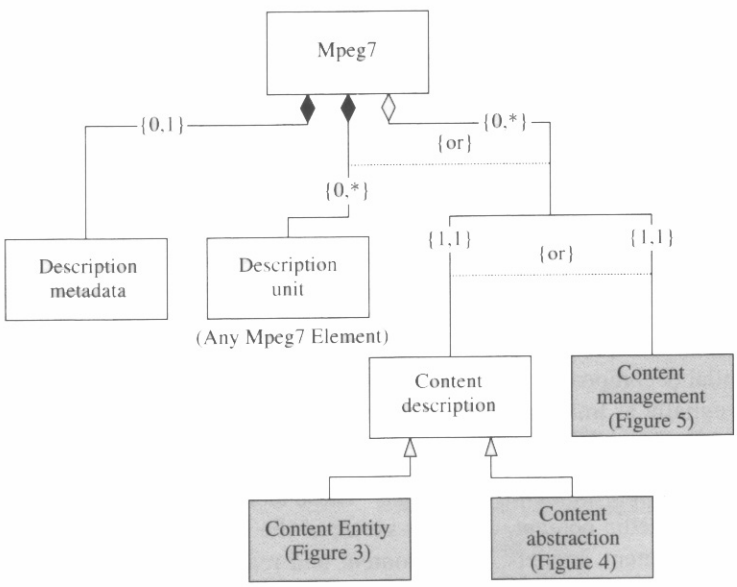

Abbildung 2.2: *MPEG-7* root und top-level Elemente [SS02a]

Linking und Localization Tools

Im Folgenden werden kurz die Möglichkeiten von Links und Lokalisierung in *MPEG-7* beschrieben.

Interne und externe Beschreibungen lassen sich ohne Einschränkung auf die XML Struktur referenzieren. Dafür gibt es drei verschiedene Typen.

- *idref*: Ein Element wird wie aus xml bekannt, durch sein ID Attribut bestimmt. Das Element muss innerhalb des Dokumentes liegen und seine ID muss eindeutig sein.
- *xpath*: Innerhalb des Dokuments kann mit Hilfe eines XPATH Ausdrucks ein bestimmter Teilbaum des Dokuments ausgewählt werden.
- *href*: Jedes beliebige Element, auch außerhalb des Dokuments, kann mit Hilfe eines Uniform Resource Identifier (URI) referenziert werden.

Um Elemente eindeutig zu identifizieren, wird der Datentyp *UniqueID* verwendet. Damit ist es möglich standardisierte Identifikatoren wie URI's oder ähnliche zu verwenden.

Medialocator werden verwendet, um Elemente zu einem bestimmten Content zuzuordnen. Mit Hilfe einer URI wird auf den Content verwiesen. Durch *InlineMedia* können Binärdaten direkt in eine *MPEG-7* Beschreibung eingefügt werden (hierfür müssen die Binärdaten z.B. in Bas64 kodiert sein). Zusätzlich kann eine *streamID* angegeben werden, um einen Stream innerhalb des Content auszuwählen. Es werden weiterhin zwei Arten unterschieden:

- Der **TemporalSegmentLocator** wählt ein Segment innerhalb eines zeitlichen Objektes aus.
- Der **ImageLocator** ermöglicht es einzelne Bilder oder Frames aus Videos zu selektieren. Zeitangaben sind auf Zeitpunkte beschränkt.

In *MPEG-7* kann Zeit auf zwei verschiedene Arten dargestellt werden. Als *media time* und als *generic time*. Die Darstellung basiert auf dem ISO 8601 Standard und ist nach [SS02b] mit verschiedenen anderen Standards wie SMIL[1] oder MPEG-4[2] kompatibel.

Basic Tools

Die *Basic Tools* stellen eine Bibliothek von *Description Schemes* und Datentypen dar, die für die Erstellung anderer *MPEG-7* Beschreibungen verwendet werden. Nach [SS02b] wird folgende Unterteilung getroffen:

- *graph und relation Tools*: Relationen stellen Beziehungen zwischen einer oder mehreren *Description Schemes* Instanzen dar.

  Der Datentyp für Graphen verwendet Listen für die Verwaltung von Knoten. Die Referenzen zwischen den Knoten werden mit URI's dargestellt.

- *textual annotations*:
  - *free text*: Es erfolgt eine Volltextbeschreibung.
  - *keyword*: Einzelne Stichworte werden angegeben.
  - *structured annotation*: Die Beschreibung wird strukturierter vorgenommen. Sie ist zwischen den beiden ersten Typen einzuordnen und enthält Strukturelemente wie "Wer", "Was", "Wo"und "Wann".
  - *dependency structure*: Grammatikalische Strukturen werden in Form von Abhängigkeiten zwischen Elementen dargestellt.

Das *OrderingKey Description Schemes* bietet die Möglichkeit, Daten nach einem bestimmten Attribut bzw. Schlüssel zu sortieren. Der Schlüssel wird durch ein *Selector* Element beschrieben, welches z.B. ein XPath Ausdruck sein kann. [WHM02]

2.3.3 Bewertung

Der Standard *MPEG-7* ermöglicht die Beschreibung einer Vielzahl von audiovisuellen Daten. Dazu bietet er ein große Menge von Deskriptoren, die eine sehr flexible Anwendung ermöglichen. Mit Hilfe der DDL kann der Standard erweitert werden. Die große Flexibilität führt aber zu einer sehr hohen Komplexität des Standards. Der hohe Strukturierungsgrad führt sehr schnell dazu, dass XML Werkzeuge zum Bearbeiten und Erstellen

[1] `http://www.w3.org/AudioVideo/`
[2] `http://www.chiariglione.org/mpeg/standards/mpeg-4/mpeg-4.htm`

der *MPEG-7* Beschreibungen benötigt werden. Die hohe Anzahl an Elementen erschwert zusätzlich die Einarbeitung, da im Normalfall nicht der gesamte Umfang genutzt wird. Hier bietet der Standard aber die Möglichkeit, die Beschreibungen einzuschränken und nur Teile zu verwenden, die auch benötigt werden. Ein weiterer Nachteil liegt darin, dass der *MPEG-7* Standard eher auf das Retrieval und Suchen von Daten ausgelegt ist. Das speichern in Realtionalen Datenbank Systemen (RDBMS) ist mit entsprechenden XML mapping Vorschriften möglich, aber aufgrund der Komplexität der Beschreibungen nicht effizient umsetzbar.

2.4 Metadata Encoding & Transmission Standard (METS)

2.4.1 Überblick

In diesem Abschnitt wird der *Metadata Encoding & Transmission Standard METS* vorgestellt, der von der *Library of Congress* (LoC) entwickelt wird. *METS* beschreibt Metadaten für das Verwalten und den Austausch von Daten in digitalen Bibliotheken. Es werden eine Reihe von Techniken zur Darstellung von beschreibenden, administrativen und strukturellen Metadaten zur Verfügung gestellt. Mit diesem Standard können auch komplexe Links zwischen Objekten dargestellt werden. Weiterhin erfüllt er die drei Rollen *Submission Information Package (SIP)*, *Archival Information Package (AIP)* und *Dissemination Information Package (DIP)* innerhalb des *Open Archival Information System (OAIS)* Referenz Modells. Im Folgenden soll der Standard stellvertretend für Modelle, die im Bibliotheksumfeld genutzt werden, vorgestellt werden.

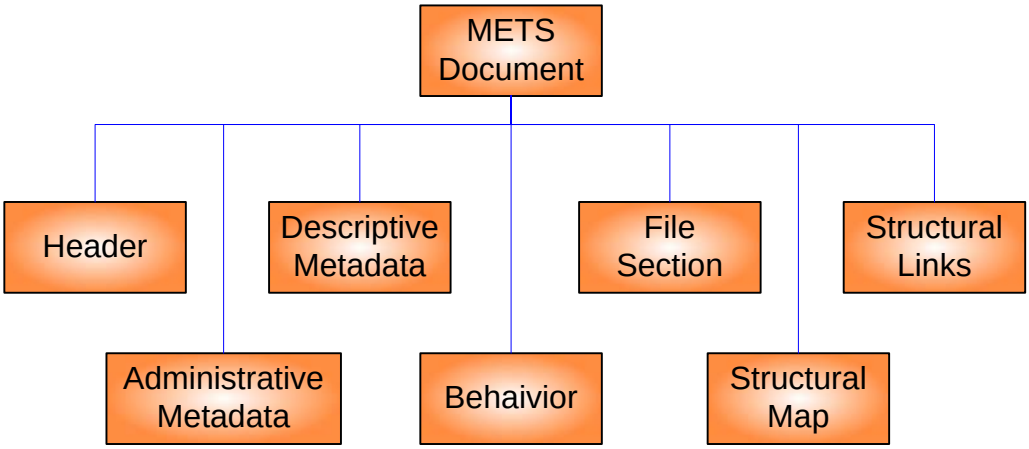

Abbildung 2.3: METS XML Schema

Abbildung 2.3 zeigt den Aufbau eines METS Dokumentes. Die 7 Hauptelemente werden nachfolgend kurz erläutert.

- Der **METS Header** beschreibt das *METS* Dokument selbst, nicht das Objekt der digitalen Bibliothek, welches durch das METS Dokument beschrieben wird. Es beinhaltet z.B. Informationen über denjenigen der das METS Dokument erstellt hat und den Autor.

- Im Abschnitt **Descriptive Metadata** können externe Metadaten eingebunden (XLink zum referenzieren) oder interne Beschreibungen zu einem *METS* Dokument (XML oder binär) angegeben wrden. Da kein bestimmtes Schema vorgeschrieben ist, hat der Benutzer die Möglichkeit eigene Bescheibungsschemata einzubinden (z.B. Dublin Core, XMetaDiss, usw.).

- Der Bereich **Administrative Metadata** speichert Informationen über die Dateien, die erstellt und gespeichert wurden. Die administrativen Metadaten sind in 4 Teile unterteilt. *Technical Metadata* wie das Format, Zeit der Erstellung usw. Den *Intellectual Property Rights Metadata* und den *Source Metadata* die das analoge Objekt beschreiben, was zu dem *METS* Objekt gehört. Als viertes die *digital Provenance Metadata*, die die Beziehungen unter den einzelnen Dateien darstellt (z.B. master/derivate Beziehungen).

- Die **File Section** beinhaltet alle Dateien des digitalen Objektes. Die Dateien können Gruppiert werden, um z.B. eine Einteilung in Bilder und Thumbnails zu ermöglichen. Zusätzlich können zu jeder Datei Metadaten hinzugefügt werden. Der Inhalt kann als base64 kodiert dirket in das *METS* Dokument eingefügt werden oder wird über XLink referenziert.

- In der **Structural Map** wird die hierachische Struktur der digitalen Objekte beschrieben. Hier werden die Objekte außerdem mit den Dateien und Metadaten verknüpft, zu denen sie gehören.

- Die **Structural Links** ermöglichen es Hyperlinks zwischen Knoten der *structural map* einzufügen. Dies wird nach [Lib] hauptsächlich für die Archivierung von Webseiten verwendet. Für die Referenzen wird der W3C Standard *XLink* verwendet.

- Im Bereich **Behavior** kann Inhalt ausführbares Verhalten zugeordnet werden. Jedes Verhalten hat immer ein Abschnitt, in dem der ausführbare Code referenziert wird.

Unbedingt erforderlich sind die *structural map* und die *file section*, alle anderen Elemente sind optional. Jedes *METS* Objekt wird als Baum dargestellt. Jeder Knoten in einem solchen Baum kann wiederum *descriptive*, *administrativ* Metadaten oder Dateien enthalten.

Wichtig für den Gebrauch von *METS* sind die *Profiles*. "METS Profiles are intended to describe a class of METS documents in sufficient detail to provide both document authors and programmers the guidance they require to create and process METS documents conforming with a particular profile."[Lib] Diese *profiles* werden auf den Internetseiten der *LoC* als XML Dateien frei zur Verfügung gesellt.

In Abbildung 2.4 ist ein Beispiel aus [Lib] für ein *METS* Dokument dargestellt.

```
<METS:dmdSec ID="DM129277">
    <METS:mdWrap MDTYPE="OTHER" OTHERMDTYPE="GDM">
        <gdm:gdm>
            <gdm:core>
                <gdm:coreDate BEGINDATE="1912" ENDDATE="1914" DATE="1912-1914"/>
                <gdm:localID LOCALIDTYPE="(MH)HOLLIS">BNI3165</gdm:localID>
                <gdm:origin>Cambridge [Mass.]</gdm:origin>
                <gdm:title>Reports of the president and treasurer
                    for...</gdm:title>
            </gdm:core>
            <gdm:creator NAMETYPE="cn" ROLE="author">Radcliffe
                College</gdm:creator>
            <gdm:creator NAMETYPE="cn" ROLE="publisher">University
                Press</gdm:creator>
        </gdm:gdm>
    </METS:mdWrap>
</METS:dmdSec>
```

Abbildung 2.4: Beispiel *METS*

2.4.2 Konzepte und Besonderheiten

Im vorherigen Abschnitt wurde die grobe Struktur des *METS* Standards vorgestellt. Wie in Abbildung 2.3 gezeigt, wird der Standard auf der obersten Ebene von 7 verschieden Elementen beschrieben, von denen nur *fileSec* und *structMap* erforderlich sind. Im Folgenden werden einzelne Elemente und Konzepte des Standards vorgestellt.

Die Descriptive Metadata Section beschreibt Metadaten mit den Elementen:

- **mdWrap** bietet die Möglichkeit externe Metadaten direkt als XML oder in binärer Form (Base64 kodiert) einzubinden.
- **mdRef** kann mit Hilfe von XLink und XPointer einen Link auf externe Metadaten setzen.

In der *administrative metadata section* stehen folgende Elemente zur Verfügung:

- **techMD** beinhaltet technische Informationen über Dateien.
- **rightsMD** beschreibt Informationen über Rechte, die einzelnen IP Addressen zugeordnet werden können.

- **sourceMD** beinhaltet Informationen über die original Resource die das *METS* Dokument beschreibt.
- **digiprovMD** zeichnet Veränderungen der digitalen Objekte und Relationen zu entsprechenden Derivaten auf.

Die oben genannten Elemente sind alle entweder vom Typ mdWrap oder mdRef.

Innerhalb der File Section stehen verschiedene Elemente zur Verfügung. Das *FLocat* Element bietet die Möglichkeit einzelne Content-Dateien auszuwählen. Dafür wird die Syntax von Xlink verwendet. Mit dem *fileGrp* Element können Dateien gruppiert werden.

Mit dem *FContent* Element können im Gegensatz zu *FLocat* Content Dateien direkt in das *METS* Dokument eingefügt werden. Auch hier besteht wieder die Möglichkeit die Datei im XML Format oder binär (Base64 kodiert) einzufügen.

Die Structural Map bietet die Möglichkeit mit dem Element *div* (division) Objekte zu strukturieren. *METS* Objekte sind immer Bäume, die Elemente vom Typ *mptr* und *fptr* enthalten. Der *mptr* ermöglicht es andere *METS* Dokumente zu referenzieren, um sie in den aktuelle Teil einzufügen. Es können außerdem *descriptive* und *administrative* Metadaten eingefügt werden. Mit *fptr* kann das *div* Element mit der Datei, die beschrieben werden soll, verknüpft werden.

Die *Structure Link* beschreibt alle Links zwischen Knoten in der *Structural Map*. Dabei wird die Syntax von Xlink und Xpointer verwendet.

2.4.3 Bewertung

Der Standard *METS* ist auf die Unterstützung für Objekte innerhalb digitaler Bibliotheken ausgerichtet. Dabei wird eine überschaubare Menge von Elementen zur Beschreibung geliefert, die eine einfache Abbildung auf Relationale Datenbank Systeme (RDBMS) ermöglicht. Ein weiterer Vorteil für die Anwendung innerhalb digitaler Bibliotheken ist das Verwalten administrativer Elemente im Standard. Ein Nachteil ist die ungenaue Trennung von beschreibenden und strukturellen Metadaten. Hier sind die Anforderungen an ein Multimedia Dokumentenmodell höher.

2.5 Umsetzung der Anforderung

2.5.1 Zeitliche Modellierung

Zeitliche Modellierung wird im ***MPEG-7*** Standard vom *TemporalRelation Classification Scheme (CS)* bereitgestellt. Damit können Elemente verwendet werden, die implizit und explizit angegebene zeitliche Attribute besitzen. Die Standard Relationen sind die binäre und die n-stufige Relation. Nach [BMRS02] baut die binäre Relation auf "*Allen's temporal interval relations*" auf. Mit der n-stufigen Relation können zwei oder mehrere

Objekte in sequentielle oder parallele Beziehung gesetzt werden.
Diese Relationen können z.B. auf Video Segmente oder Ereignisse angewendet werden. [BMRS02]

Der Metadaten Standard ***METS*** bietet verschiedene Möglichkeiten zeitliche Beziehungen zu modellieren. Dabei kann in der *Structural Map* zu jedem Bereich (div) ein *fptr* (file pointer) angegeben werden. Innerhalb dieses file pointers besteht die Möglichkeit mit den Elementen *par* für parallel und *seq* für Sequenz zeitliche Beziehungen darzustellen. Der Zeitpunkt, wann Elemente abgespielt bzw. angezeigt werden, kann mit *par* angegeben werden. Eine sequentielle Wiedergabe von Elementen ist mit dem *seq* Element möglich.

2.5.2 Räumliche Modellierung

MPEG-7 stellt mehrere Möglichkeiten für die Beschreibung von räumlichen Zusammenhängen zur Verfügung. Elemente zur Strukturierung werden im *SpatialRelation Classification Scheme (CS)* zusammengefasst. Innerhalb dieses Schemas können 2D Objekte räumlich in Relation gesetzt werden. Zur Auswahl stehen 14 verschiedene Richtungsrelationen wie z.B. *links* oder *northwest*. Diese Relationen können unter anderem auf Bildregionen und Objekte angewendet werden. Eine Besonderheit im *MPEG-7* Standard ist die inverse Relation, die Relationen invertiert.
Eine weitere Möglichkeit stellt die Verwendung von generischen Relationen dar, die durch das *GraphRelation CS* und das *BaseRelation CS* gegeben sind. Sie geben Relationen zwischen einzelnen Segmenten an.
Zusätzlich ist es ebenfalls möglich, mit Hilfe von *Segment Entities* eine Strukturierung vorzunehmen. Damit können topologische Relationen dargestellt werden. [BMRS02]

Der Metadaten Standard ***METS*** bietet keine Möglichkeiten, räumliche Beziehungen zu modellieren. Die Modellierung bezieht sich auf hierarchische Strukturen zischen den Objekten.

2.5.3 Interaktive Modellierung

Im ***MPEG-7*** Standard werden verschiedene Formen der interaktiven Modellierung unterstützt. Eine Möglichkeit stellen Tools für das Zusammenfassen dar. Mit den beiden Schemata *HierarchicalSummary* und *SequentialSummary* lassen sich hierarchische oder sequentielle Zusammenfassungen erstellen. Die beiden Schemata können auf Zeit unterschiedlichen (time-varying) audiovisuellen Daten angewendet werden.

Weiterhin können Ansichten (Views) und die Zerlegung von Ansichten (view decomposition) verwendet werden. Dabei lässt sich eine Ansicht durch Filterung, Regionen oder aus Partitionen erstellen. Die Zerlegung bietet zusätzlich die Möglichkeit, Graphen, Bäume oder Sets zu beschreiben.

Mit dem Konzept der Variationen (variations) ist es möglich, alternative oder zusätzliche Elemente wie z.B. komprimierte Bilder, Bilder in anderer Qualität oder anderen

Sprachen zur Verfügung zu stellen. [BS02]

Besonders hervorzuheben ist, dass im MPEG-7 Standard video hyperlinking angewendet werden kann. Mit Hilfe von *motion descriptors* können so Hyperlinks auf Regionen innerhalb eines Videos gelegt werden.

Der Metadaten Standard ***METS*** bietet keine Möglichkeiten, Interaktionen zu modellieren. Die Modellierung bezieht sich nur auf hierarchische Strukturen zischen den Objekten.

2.5.4 Wiederverwendbarkeit

Die Wiederverwendbarkeit von kompletten Dokumenten wird im ***MPEG-7*** Standard durch das Konzept der *MediaLocator* breit gestellt. Es wird ein generischer Mechanismus zur Lokalisierung von Multimedia Content zur Verfügung gestellt. Damit kann mit Hilfe einer URI ein bestimmtes Dokument ausgewählt werden. Zusätzlich besteht die Möglichkeit Content direkt in die Beschreibung als Binärdaten einzufügen. Besteht das Dokument aus verschiedenen bzw. mehreren Streams, so ermöglicht es die *StreamID* bestimmte Streams innerhalb eines Dokuments anzusprechen.

Mit dem *TemporalSegmentLocator* lassen sich Zeitspannen angeben. Der *ImageLocator* kann, eingeschränkt auf Zeitpunkte, einzelne Bilder oder Frames aus einem Video auswählen.

Weiterhin können *MPEG-7* Dokumente wieder verwendet werden. Hierfür steht das Konzept der *references* zur Verfügung. Beschreibungen können, wie in Abschnitt 2.3.2 beschrieben, mit idref innerhalb des aktuellen Dokumentes, mit xpath Teilbäume oder mit href ein beliebiges Dokument ausgewählt werden.
[WHM02]

Die Wiederverwendbarkeit ist im ***METS*** Standard eingeschränkt. Dabei werden die Möglichkeiten der *File Section* und der *Structural Map* zusammen verwendet. Die in der File Section angegebenen Dateien werden in der Structural Map bestimmten Bereichen (devisions) zugeordnet. Innerhalb dieser Bereiche kann mit dem *fptr/area* Element eine Position innerhalb einer Datei angegeben werden. Das *area* Element zeigt auf eindimensionale oder zweidimensionale Elemente in audio, video, text oder Bilddaten.

2.5.5 Auswahl und Identifikation

Der ***MPEG-7*** Standard bietet verschiedene Möglichkeiten, einzelne Objekte bzw. Elemente mit Informationen zu versehen.
Kollektionen lassen sich mit Hilfe der *Collection Tools* erstellen. Sie werden aus Content, Segmenten oder Deskriptor Instanzen gebildet.

Die *Model Tools* bieten die Möglichkeit, *Wahrscheinlichkeitsmodelle*, *analytische Modelle*, *Cluster Modelle* und *Klassifikationen Modelle* zu konstruieren. Speziell mit den Klassifikationen Modellen kann Inhalt mit Labeln oder semantischen Informationen versehen werden. Weiterhin kann unbekannter Inhalt klassifiziert werden. [SB02]

Der ***METS*** Standard stellt keine erweiterten Möglichkeiten für die Auswahl und Identifikation von Objekten zur Verfügung. Innerhalb des *descriptive metadata* Elementes können Metatdaten beschrieben werden. Mit dem *mdRef* Element ist es möglich, externe Metadaten Beschreibungen einzubinden. Dabei können *METS* Dokumente und verschiedene andere Typen referenziert werden. Der Typ, die Verwendung und ähnliches werden dabei mit verschieden Attributen definiert.

Zusätzlich zu den externen Beschreibungen können Metadaten intern eingebunden werden. Dafür steht das Element *mdWrap* zur Verfügung. Es stellt einen Wrapper dar, mit dem sich entweder weitere Informationen im XML Format, oder binär kodierte (Base64 kodiert) Daten einfügen lassen.

2.5.6 Anpassbarkeit

Im ***MPEG-7*** Standard können die Verwendungshistorie (usage history) und Nutzer Einstellungen (user preferences) abgelegt werden.
Die Verwendungshistorie speichert Informationen über die ausgeführten Aktionen des Nutzers. Dabei kann zwischen verschiedenen Typen von Aktionen unterschieden werden. Die Aktionen werden in Listen als nicht überlappende Zeitspannen gespeichert. Anhand dieser Daten, kann der Service Provider sein Angebot auf den jeweiligen Nutzer zuschneiden.

Ein weitere Möglichkeit den Inhalt anzupassen besteht darin, Nutzer Einstellungen zu speichern. Der Nutzer kann Informationen über den Inhalt, der gesucht, gefiltert oder angesehen werden soll, ablegen (*FilteringAndSearchPreferences*). Hier bietet der *MPEG-7* Standard ein Vielzahl von Möglichkeiten zur Einschränkung. Die *BrowsingPreferences* speichern Einstellungen über das Verhalten beim Browsen und Navigieren ab.

Nach [BYF02] besteht zusätzlich die Möglichkeit, aus der Verwendungshistorie automatisch Nutzer Einstellungen abzuleiten. [BYF02]

Der Metadaten Standard ***METS*** bietet keine Möglichkeiten den Content anzupassen. Dabei kann nicht auf technische Gegebenheiten eingegangen werden. Es ist ebenfalls nicht möglich, Kontextabhängigkeiten anzugeben. Mit Hilfe der Gruppierungsmöglichkeit für Dateien in der File Sectionen können verschieden Gruppen von Dateien angegeben werden. Eine Gruppe besteht z.B. aus einem Bild in hoher und niedriger Auflösung. Gegebenenfalls kann diese Darstellung von Anwendungen weiter verwendet werden.

2.5.7 Darstellungsneutrale Beschreibung

Der Standard ***MPEG-7*** basiert auf XML und wird durch ein erweitertes XML Schema beschrieben. Die Struktur der Daten wird von der Beschreibung getrennt. So ist eine darstellungsneutrale Beschreibung möglich.

Der Standard ***METS*** basiert auf XML und wird durch ein XML Schema beschrieben. Im Standard wird keine eindeutige Trennung von Struktur und beschreibenden Metadaten gemacht.

2.5.8 Beziehungen zwischen Objekten

Im ***MPEG-7*** Standard können Objekt-Reihenfolgen mit Hilfe des *Description Schemes OrderingKey* dargestellt werden. Attributbeziehungen werden durch Erweiterungen der jeweiligen *Description Schemes* realisiert. Durch das Prinzip der *MediaLocator* können ebenfalls Erweiterungen einzelner Schemata-Komponenten-Beziehungen erstellt werden. Substitutionsbeziehungen und semantisch äquivalente Beziehungen lassen sich in *MPEG-7* mit dem Konzept der Variationen (variations) dargestellen.

Beziehungen zwischen Objekten werden im ***METS*** Standard in der Structural Map angegeben. Es ist es außerdem möglich, bestimmte Reihenfolgen zwischen Bereichen (devisions) mit dem *ORDER* Attribut der *structMap* zu bestimmen.

2.5.9 Unterstützung von anderen Metadaten Formaten

Die Intergration anderer Formate kann im ***MPEG-7*** Standard durch Erweiterungen entsprechender *Description Schemes* erfolgen. Ein Beispiel für den Metadaten Standard Dublin Core (DC) wurde in 2.3.1 beschrieben.

Wie bereits in Abschnitt 2.5.5 erwähnt, können andere Metadaten Formate in ***METS*** mit Hilfe des *descriptive Metadata* Elements eingefügt oder verwendet werden. Dabei ist es möglich, externe Daten zu referenzieren oder Daten direkt als XML in das Dokument einzufügen. Eine weiter Möglichkeit besteht darin, die Metadaten als Binärdaten in das *METS* Dokument einzufügen.

2.5.10 Zusammenfassende Darstellung

Tabelle 2.2 gibt einen Überblick über die Betrachtungen aus Abschnitt 2.5. Es wird zusammengefasst, welche Anforderungen die Dokumentenmodelle *MPEG-7* und *METS* unterstützten.

Anforderung	***MPEG-7***	***METS***
Zeitliche Modellierung	+	+
Räumliche Modellierung	+	-
Interaktionen Modellierung	+	-
Wiederverwendbarkeit	+	o
Auswahl und Identifikation	+	+
Anpassbarkeit	+	o
Darstellungsneutrale Beschreibung	+	-
Beziehungen zwischen Objekten	+	+
Unterstützung von anderen Metadaten Formaten	+	+

wird unterstützt: + wird nicht unterstützt: - wird teilweise unterstützt: o

Tabelle 2.2: Anforderungen die *MPEG-7* und *METS* unterstützen

Wie in Tabelle 2.2 gezeigt, bietet *MPEG-7* die Möglichkeit alle Anforderungen umzusetzen. Ausgelegt als Format für den Austausch von Informationen, ist die Nutzbarkeit und die Interoperabilität des Standards ein Problem. Für den Anwender ist es relativ einfach eine *MPEG-7* konforme Beschreibung zu erstellen. Durch den hohen Freiheitsgrad bei der Verwendung von Elementen und Attributen sind diese Beschreibungen aber sehr allgemein. Dies führt häufig dazu, dass gleiche Dokumente mit unterschiedlichen *MPEG-7* Deskriptoren dargestellt werden. Außerdem können diese Deskriptoren von Dritten häufig nur schwer verstanden werden.[TMPP04]

3 Publikationsprozess für Multimedia Dokumente in digitalen Bibliotheken

Das Erstellen von komplex strukturierten Multimediadokumenten in digitalen Bibliotheken, erfolgt auf der Grundlage eines Publikationsprozesses. Die Autoren werden häufig nur bei der Erzeugung von Teildokumenten unterstützt. In diesem Kapitel soll ein Content-Life-Cycle für Multimediadokumente entwickelt werden, der den Autoren bei der Publikation hilft.

Nach Betrachtungen zu digitalen Bibliotheken und Content-Management-Systemen soll ein Content-Life-Cycle für Multimediadokumente in digitalen Bibliotheken erstellt werden.
Ausgehend von der Definition in [HD02], dass digitale Bibliothekssysteme "alle Prozesse auf einer digitalen Bibliothek [unterstützen], die auch das Content-Management-System anbietet", soll versucht werden, den Publikationsprozess von Content-Management-Systemen, als Ausgangspunkt für die weitere Betrachtung zu verwenden.

In dieser Arbeit wird der Begriff Publikationsprozess für die Beschreibung des Weges von Content zum Multimediadokument in digitalen Bibliotheken verwendet. Der Begriff Veröffentlichung stellt die Bereitstellung von Multimediadokumenten dar.

3.1 Digitale Bibliothek

Digitale Bibliotheken sind darauf ausgelegt, Dokumente mit bleibenden Wert zu sammeln, zu archivieren, recherchierbar zu machen und ihn Nutzern zur Verfügung zu stellen. Die Bibliotheken müssen sicherstellen, dass diese Dokumente langfristig zitierbar sind. Zu den Dokumenten bietet die Bibliothek beschreibende Metadaten an.
Die digitale Bibliothek soll zusätzlich zu der anbieterseitigen Content-Verwaltung auch eine kundenseitige Verwaltung beinhalten. Der Content-Life-Cycle (CLC) einer digitalen Bibliothek ist in Abbildung 3.1 dargestellt. Ein *Content-Life-Cycle* beschreibt die Stufen, die ein (digitales) Objekt in seinem Leben durchläuft.

Die Verwendung von digitalen Bibliotheken wird durch die elektronische Unterstützung beim

- Schreiben
- Produzieren
- Verteilen

- Finden
- Zusammenstellen
- Archivieren
- Lesen/Nutzen

von verschiedenen Dokumenttypen motiviert.

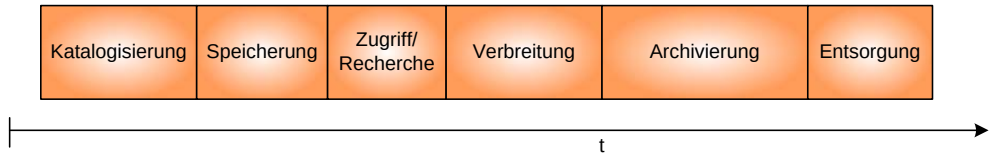

Abbildung 3.1: Dokument Content-Life-Cycle in digitalen Bibliotheken [HM05]

Die Vorteile von digitalen Dokumenten liegen z.B. in der schnellen weltweiten Verfügbarkeit, in der gleichzeitigen Nutzung eines Exemplars, in der automatischen Erschließbarkeit oder der Integration Verschiedener Medien.
Die Datentypen von digitale Dokumenten werden in diskrete und kontinuierliche Typen unterteilt. Diskrete Typen sind:

- Strukturierte Daten
- Texte
- Zeichnungen
- Bilder

Zu den kontinuierlichen Typen gehören:

- Audio
- Video
- Animation

Eine Einteilung von Dokumenten in verschiedene Typen, kann nach Art ihres Inhaltes vorgenommen werden. [EF00] nimmt dazu folgende Unterteilung vor:

- **Referenzwerke** sind Dokumente, deren Nutzung wiederkehrend ist und die nie im ganzen verwendet werden. Dazu gehören z.B. Produkt-Kataloge, Lexika oder Gesetzestexte.
- **Primäre Text-Veröffentlichungen** enthalten neues Wissen. Die Texte werden selektiv bezüglich einzelner Dokumente ausgewählt, dann aber vollständig genutzt. Forschungsberichte und Fachzeitschriften sind Beispiele für diesen Dokumenttyp.

- **Druckbare Nicht-Text-Dokumente** werden vollständig genutzt. Dazu gehören z.B. Bilder, Karten oder technische Zeichnungen.
- **Nicht-Druckbare Dokumente** werden vollständig genutzt. Audio, Video, Animation und Software sind Beispiele zu diesem Dokumenttyp. Multimediadokumente werden auch zu diesem Typ gezählt.

3.1.1 Speichern und Archivieren von digitalen Dokumenten

Die Speicherung von digitalen Dokumenten befasst sich mit der logischen Speicherorganisation und Datenverwaltung. Daten werden unterteilt in:

- **Strukturierte Daten** (z.B. Metadaten), die gut mit Datenbankmodellen dargestellt werden können.
- **Unstrukturierte Daten**, die gut mit Dokumentenmodellen dargestellt werden können.
- **Semi-strukturierte Daten**, die gut durch Markup-Sprachen wie SGML oder XML dargestellt werden können.

Dokument-zentrierte Anwendungen, wie digitale Bibliotheken, die tief strukturierte Dokument-Hierarchien besitzen, verwenden häufig XML zum speichern. Dies bietet die Möglichkeit beliebige Datentypen zu modellieren. Weiterführende Informationen über das Thema *XML und Datenbanken* gibt es unter
`http://www.xml-und-datenbanken.de/`.

Für Bibliotheken ist es sehr wichtig, Dokumente identifizieren zu können. In konventionellen Bibliotheken werden Bücher z.B. anhand ihrer ISBN (*International Standard Book Number*) oder der ISSN (*International Standard Serial Number*) eindeutig identifiziert. Liegen von einem Buch mehrere Exemplare vor, müssen diese mit verschiedenen Signaturen unterschieden werden.
In digitalen Bibliotheken kommen drei verschiedene Identifikatoren zum Einsatz:

- Die **URL** (*Uniform Resource Locator*) bestimmt eine Datei, welche zu einem bestimmen Zeitpunkt auf einem Rechner zu finden ist. Das Problem bei der Verwendung von URL's besteht darin, dass sich die URL verändert, wenn sich z.B. der physikalische Ort der Resource ändert.
- **DOI** (*Digital Object Identifier*) ist eine eindeutige Nummer, die sich aus der Nummer einer Registrierungsstelle, der Nummer des Verlages und einer beliebigen Dokumentnummer zusammensetzt[1]. Eine DOI wird durch die Verlage zertifiziert.
- Die **URN** (*Uniform Resource Name*) ist genau so aufgebaut wie die DOI, wird aber durch Universitätsverbünde oder einzelne Universitäten zertifiziert[2].

[1] `http://www.doi.org/`
[2] `http://www.persistent-identifier.de/`

Die Archivierung von digitalen Dokumenten beschreibt eine langfristige Aufbewahrung der Dokumente, die über die aktive Nutzung deutlich hinaus geht. Wichtig bei der Speicherung von Dokumenten ist die Verwendung von anerkannten Datenformaten. Die Datenformate müssen für eine langfristige Archivierung geeignet sein.
Für die Speicherung von Textdokumenten können die Formate XML, SGML, oder PDF verwendet werden. Bei der Archivierung von Nicht-Textdokumenten, gibt es noch keine Empfehlungen.

3.1.2 Suchen und Gewinnen von Informationen

Das Suchen und die Gewinnung von Informationen in digitalen Bibliotheken ist eine Kombination von Anfragen und Suchen.

- Anfragen werden an Metadaten gestellt, die strukturiert vorliegen.
- Eine Suche wird in Metadaten vorgenommen, die als unstrukturierte Informationen oder in Volltexten/Multimedia-Inhalten vorliegen.

Um ein Dokument zu finden, kann der Nutzer Anfragen an Metadaten stellen, oder in den Metadaten suchen. Weitere Hilfsmittel, die eine digitale Bibliothek zur Verfügung stellen sollte, sind:

- Thesaurus, z.B. für Synonyme oder Unter- und Oberbegriffe
- Klassifikationssysteme, z.B. ACM-Klassifikation, oder DDC (Dewey Decimal Classification)
- Lexikon, z.B. Mehrsprachigkeit

Zusätzlich zu den genannten Hilfsmitteln, können Alternativen zum Suchen, wie die Navigation über ein Klassifikationssystem oder das Browsing über Web-Browser angeboten werden.
Um die Möglichkeiten von Suche und Anfragen nutzen zu können, benötigt die digitale Bibliothek Metadaten. Der Vorgang für die Erzeugung von Metadaten wird Deskribierung genannt. Die Deskribierung erzeugt Deskriptoren, die durch automatische oder manuelle Techniken generiert werden können.
Manuell wird z.B. eine Klassifizierung oder eine Vergabe von Schlagworten vorgenommen. Ein Beispiel für ein automatisches Verfahren der Deskribierung ist die morphologische Reduktion, bei der die Deskriptoren nur in ihren Grundformen verwendet werden.

Um die vorhandenen Suchverfahren effektiv einsetzten zu können, sollten folgende Suchhilfen nach [EF00] in jeder digitalen Bibliothek enthalten sein:

- Bestandskataloge (z.B. OPAC)
- lokale und entfernte Nachweis-Datenbanken (z.B. DBLP)

- Suchmaschinen
- Internet-Kataloge, die fachlich relevant sind

Die Bereitstellung dieser Dienste sollte dem Nutzer über eine einheitliche Schnittstelle zur Verfügung gestellt werden. Dabei ist oft eine Transformation der Anfragen, auf die jeweiligen Systeme zugeschnitten, notwendig.
Überschreitet das Ergebniss einer Anfrage eine bestimmte Anzahl, erwartet der Nutzer eine sinnvolle Reihenfolge bei der Ausgabe. Die Treffer müssen für die Präsentation sortiert und gemischt werden. Üblich ist die Ausgabe der Dokumente in Folge ihrer Relevanz. Dafür werden Ranking Techniken verwendet. Um eine einfache Selektion der Suchergebnisse bei einem Überangebot vornehmen zu können, sollten weiterhin Leseproben, Kommentare und Gutachten oder Browsen in der Umgebung von Suchergebnissen zur Verfügung stehen.

Alternativen zum Suchen sind das Navigieren und Browsing. Das Navigieren setzt das Vorhandensein von Klassifikationen voraus. Die Klassifikationen sind oft als Baumstrukturen organisiert, über dessen Stufen der Nutzer den Suchraum einschränken kann.

Das Browsing ermöglicht, oft mit Hilfe von Hypertext, das Verfolgen von Verweisen die im Dokument angegeben sind. Die Verweise können in das Dokument oder außerhalb des Dokumentes zeigen. Dabei können Verweise nur Informationen innerhalb des Suchraums verwenden.

3.1.3 Verbreitung

Diese Stufe beschreibt die Verteilung der Dokumente. Der Transport von Dokumenten zum Nutzer kann auf verschiedenen Weise statt finden. Es stehen eine Reihe von Protokollen zur Verfügung, die kurz genannt werden.

- Internet Protokolle wie TCP/IP, FTP oder HTTP bieten die Möglichkeit über entsprechende Browser oder Clients die Dokumente zu betrachten.
- Spezielle Bibliotheks-Protokolle wie der Z39.50[3] Standard erlauben es, Recherchen in Nachweis-Datenbanken über das Internet durchzuführen. Der Standard definiert eine eigene Abfragesprache (RPN Query) und mehrere eigene Ergebnisformate.

Verschiedene Darstellungsformate für Dokumente erlauben es die Darstellung an Nutzerbedürfnisse anzupassen. Formate für Texte sind z.B. PDF, XML oder PostScript.

3.2 Content-Management

Im Folgenden soll ein Überblick über die einzelnen Komponenten eines Content-Management-Systemen gegeben werden (siehe [Boi01]).

[3] `http://www.loc.gov/z3950/agency/`

Um den Begriff des Content-Management zu definieren, wird zunächst der Begriff Content erklärt. Nach [HM05] stellt Content Inhalte wie Daten, Dokumente, Multimedia-Objekte zur Bereitstellung in Rechnernetzen dar. Der Content wird durch Autoren oder Redakteure erfasst und danach publiziert.

CM steht weiterhin für das Erzeugen und Präsentieren von Content, der je nach Anwendung langlebig oder sehr kurzlebig sein kann. Im CM ist aber im Gegensatz zur digitalen Bibliothek keine kundenseitige Content-Verwaltung vorgesehen. Der Content Life Cycle des CM ist in Abbildung 3.2 dargestellt.

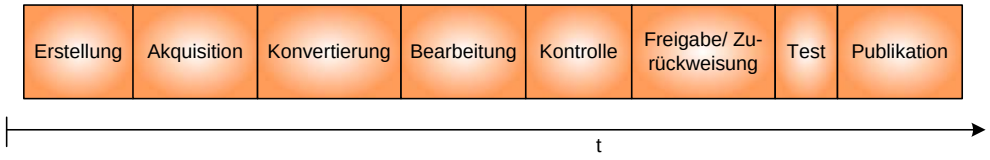

Abbildung 3.2: Dokument Content-Life-Cycle im Content-Management [HM05]

Im Bereich der digitalen Bibliotheken muss der Begriff des Content nach [HM05] dahingehend verändert werden, dass Inhalte wie Daten, Dokumente, Multimedia-Objekte von bleibenden Wert sind. Weiterhin wird er den Benutzern zur Verfügung gestellt.

Der Begriff Content-Management (CM) wird in [HM05] als "*Verwaltung multimedialer Dokumente(Content) zur Bereitstellung im LAN / WAN*" definiert.

In [Boi01] wird ein detaillierter Überblick über CM gegeben. Der Autor beschreibt den Aufbau von Content-Management-Systemen (Einzahl: CMS, Mehrzahl: CMSe oder CMSen), wobei er die einzelnen Komponenten erläutert. Weiterhin geht er auf die Planung der einzelnen Komponenten ein und gibt dabei einen Produkt unabhängigen Einblick.

Ein CMS kann grob in drei Teilbereiche unterteilt werden:

- *Collection-System*
- *Management-System*
- *Veröffentlichungs-System*

Das Collection-System sammelt beliebigen Content, der mit einem CMS verwaltet werden soll. Einzelne Schritte, wie das Segmentieren und die Angabe von Metadaten, sind Teil dieser Komponente. Durch das Segmentieren wird der Content in seine elementaren Bestandteile zerlegt. Ein elementarer Bestandteil ist hier Content entsprechend oben genannter Definition. Das Management-System fast die Bereiche Repository, Workflow und Administration zusammen. Der Content wird in einem Repository abgelegt, in dem er verwaltet werden kann. Als letzte Komponente ist das Veröffentlichungs-System zu nennen. Hier wird der Content aus dem Repository in entsprechenden Publikationen

ausgegeben. Im Folgenden werden die einzelnen Komponenten im Detail beschrieben, um einen Überblick über Teilaufgaben zu geben.

In [HD02] werden die drei Stufen des Content-Management-Systeme unterschieden:

- Integrationssystem
- Redaktionssystem
- Repository-System

Das Integrationsstem entspricht dem oben vorgestellte Collection-System mit Ausnahme der Aggregation. Die Aggregation ist gleichbedeutend mit dem Redaktionssystem in [HD02]. Das Repository-System ist mit dem Management-System zu vergleichen.

3.2.1 Das Collection-System

Das Collection-System bereitet den Content auf, der in das CMS eingestellt werden soll. Dabei werden die Phasen

- Erstellen
- Erfassen
- Konvertieren
- Aggregieren

unterschieden. Der Content wird in seine Bestandteile zerlegt und so aufbereitet, dass er in die Struktur des CMS passt.

Das Erstellen von Content

Das Erstellen von Content bezieht sich auf das Erstellen neuen Contents mittels bestimmter Autorenwerkzeuge. Hierbei ist es wichtig, dass ein CMS dem Autor die Möglichkeit bietet, Autorenwerkzeuge zu verwenden. Dabei kann zum einen eine komplett in das CMS integrierte Autoren-Umgebung zur Verfügung gestellt werden (z.B. ein Texteditor), zum anderen kann auch eine Schnittstelle zu einem externen Autorenwerkzeug eingebunden werden. Diese Lösungen können aber nur schwer den Funktionsumfang von Autorenwerkzeugen bieten. Der Autor kann nicht in seiner gewohnten Entwicklungsumgebung arbeiten. Eigene Entwicklungen können den von Autoren gewohnten Funktionsumfang häufig nicht bieten.

Ein besserer Ansatz ist hier die Schnittstelle zu einem Autorenwerkzeug. Der Vorteil liegt darin, dass der Autor sein gewohnten Editor und dessen meist großen Funktionsumfang nutzen kann. Weiterhin kann so Content erstellt werden, der den Anforderungen des CMS an Struktur und binärem Format entspricht.

Um den Prozess des Erstellens weiter zu unterstützen, können Templates zur Verfügung gestellt werden. Zusätzlich sollten Meta Informationen entsprechend des Content automatisch hinzugefügt werden. Dies können z.B. Informationen über Autor oder Datum der Bearbeitung sein.

Das Erfassen von Content

Das Erfassen von Content wird immer dann notwendig, wenn er nicht selbstständig erstellt wird oder wenn Content nicht direkt aus den Autoren Werkzeugen in das Publikationssystem geladen werden kann. Dann muss eine Upload Möglichkeit bereitgestellt werden. Beim Erfassen von Content unterscheidet man grundsätzlich zwei Typen:

- Sourcen, die für Wiederverwendung ausgelegt sind. Sie sind bereits mit Metadaten versehen und in ihre einzelnen Komponenten zerlegt. Außerdem liegen sie in einem Standardformat vor, welches ohne weitere Probleme weiter verarbeitet werden kann.
- Dateien, die den oben genannten Bedingungen nicht genügen. Sie liegen in jeder möglichen Form vor. Dies können z.B. Dateien in Datenbanksystemen sein, aber auch analoge Video Signale, die vor der Bearbeitung erst digitalisiert werden müssen.

Um Datenverluste zu verhindern, sollte der erfasste Content komplett in das eigene Speichersystem überführt werden.

Das Konvertieren von Content

Content, der beim Erfassen nicht in dem Format des CMS vorliegt, oder nicht entsprechenden Strukturen genügt, muss konvertiert werden.
Das Konvertieren wird in drei Stufen unterteilt:

- nicht benötigter Inhalt wird entfernt
- mapping der Binärdaten in das Standard Format des CMS
- mapping der Struktur des Content auf die Standard Struktur des CMS

Das Konvertieren des Content ist notwendig, damit die Daten in einem für das Publikationssystem einheitlichen Format vorliegen. Folgende Schritte wie z.B. das Segmentieren können dann auf bestimmte Formate ausgerichtet werden.

Das Aggregieren von Content

Nachdem der Content erfasst und gegebenenfalls konvertiert ist, muss er aggregiert werden. Durch das Aggregieren werden alle Quellen in eine einheitliche Struktur überführt.

Der Prozess der Aggregation wird in drei Bereiche gegliedert:

Die *redaktionelle Verarbeitung* stellt ein Framework zur Unterstützung des Redaktionsprozesses zur Verfügung. Dabei werden bestimmte Regeln überprüft, die sicherstellen,

dass vorgegebene Standards eingehalten werden. Dazu gehört das Überprüfen, ob der Inhalt entsprechenden Richtlinien genügt (z.B. Rechtschreibung, Wortwahl, Zeitformen usw.). Weiterhin können Konsistenzregeln überprüft werden. Die Regeln sollen einen einheitlich hohen Standard der Publikationen sicherstellen.

Bei der *Segmentierung des Content* wird gegebenenfalls zusammengesetzter Content in elementare Teile zerlegt, die wiederum Content Elemente darstellen. Hierfür ist es wichtig, dass vorher festgelegt wurde, welche Art von Content erstellt werden soll. Die Segmentierung kann häufig als Teil der Konvertierung ausgeführt werden. Es ist aber auch möglich Content einzeln zu erstellen und danach zu erfassen. Wichtig ist, dass bei der Segmentierung die Struktur der zusammengesetzten Objekte geeignet abgespeichert wird.

Alle Content Elemente, die aus der Segmentierung entstanden sind, müssen mit Metadaten versehen werden. Das ist notwendig, damit der Content später wieder auffindbar ist und die Möglichkeit der Wiederverwendung besteht. Metadaten können automatisch, halbautomatisch oder manuell erstellt werden. Die Daten können über ein Formular oder eine Anwendung im System gespeichert werden. Außerdem können sie über Listen und andere Schnittstellen importiert werden. Das ermöglicht dem Autor, bereits existierende Metadaten weiter zu verwenden. Weitere Informationen dazu in Abschnitt 2.1.

Collection Service

Mit Hilfe der Collection Services werden die Daten der Autoren in das Repository geladen. Es wird unterschieden, ob der Content direkt beim Erstellen in das System geladen werden kann oder im Nachhinein einzeln oder in Gruppen geladen werden muss.

Am häufigsten wird Content per Webinterface in das System überführt, wobei Metadaten und die eigentlichen Dateien über Formulare an das System geschickt werden.

Eine weitere Möglichkeit besteht darin, einzelne Tools wie z.B. einen Word Prozessor mit Templates zu versorgen, welche Autoren benutzen können. Das spätere Erfassen wird durch die vorgegebene Struktur erleichtert. Zusätzlich können Dateien, die in Gruppen organisiert sind, in das System geladen werden. Hierfür werden im Normalfall spezielle Tools benötigt, die diese Aufgabe erfüllen. CMS stellen hierfür normalerweise Schnittstellen zur Verfügung, aber keine eigenen Tools.

3.2.2 Das Management-System

Innerhalb des Management-Systems sind die Bereiche Repository, Workflow und Administration zu unterscheiden:

Repository

Das Repository speichert Content, der aus dem Collection-System gewonnen wurde. Für das Speichern stehen meist mehrere Möglichkeiten zur Verfügung. Am häufigsten werden Kombinationen aus relationalen Datenbanken und Dateisystemen eingesetzt. Bei dieser Variante wird der Content (z.B. pdf Dateien) in einem Dateisystem abgelegt. Innerhalb der Datenbank muss ein Verweis auf diese Datei verwaltet werden. Das hat den Vorteil,

dass Dateien schnell verfügbar sind. Es ist aber genauso möglich, jeden Content Typ in Datenbanken abzulegen.

Informationen zur Kontrolle und Konfiguration des Systems werden in Datenbanken abgelegt. Dies können Templates, Konfigurationsdateien, Metadaten und ähnliches sein.

Im Repository werden Index Informationen über den Content angelegt. Die vorhandenen Metadaten werden indiziert und andere Indizes wie z.B. Volltextindizes werden erstellt.

Die Zugriffsrechte auf das Repository werden im Administrations System verwaltet und festgelegt.

Workflow

Das Workflow-System kontrolliert Aktivitäten innerhalb des CMS. Hier steht es offen, wo die Kontrolle des Workflow einsetzt. Sie kann bei der Erfassung oder Erstellung des Content beginnen und bis zur Publikation reichen. Die Vorteile eines Workflow System liegen darin, dass Abläufe kontrolliert bearbeitet werden können. Der Weg des Content wird durch einen Workflow genau beschrieben. Es ist weiterhin möglich, Aufgaben effektiv an bestimmte Resource (z.B. Bearbeiter) zu verteilen.

Administration

Die Adminstration verwaltet die Nutzer, die mit dem CMS arbeiten. Für die Verwaltung muss die Möglichkeit bestehen, Nutzer und Gruppen anzulegen. Nutzer können entweder Rollen oder Gruppen zugeordnet werden, denen entsprechende Rechte zugewiesen sind. Anhand der Rechte kann entschieden werden, ob der Nutzer z.B. berechtigt ist, eine Komponente im System abzulegen.

3.2.3 Das Veröffentlichungs-System

Das Veröffentlichungs-System stellt Content aus dem Repository zur Verfügung. Der Content wird mit Hilfe von Templates zu Publikationen zusammengefügt und veröffentlicht. Templates können statisch oder dynamisch sein. Statische Templates setzen den Content anhand von festen Regeln. Bei dynamischen Templates hingegen werden z.B. Informationen zur Personalisierung ausgewertet. Dazu müssen ebenfalls Metadaten des Content verwendet werden.

3.3 Der Content Life Cycle für den Publikationsprozess in digitalen Bibliotheken

3.3.1 Allgemein

In Abbildung 3.3 werden der Content-Life-Cycle aus Abschnitt 3.1 und aus Abschnitt 3.2 gezeigt.

Um den Publikationsprozess von Multimediadokumenten in einer digitalen Bibliothek zu unterstützen, wird im Folgenden der CLC aus dem Bereich des CM mit dem CLC

aus dem Bereich der digitalen Bibliotheken zusammengeführt.
Dabei wird der CLC aus dem CM dem CLC aus der digitalen Bibliothek voran gestellt. Die Abbildung 3.4 zeigt den neuen CLC für Objekte einer digitale Bibliothek.

Wie bereits erwähnt, ist der CLC an die in Abschnitt 3.2 vorgestellten Komponenten und deren Funktionalität angelehnt.

Innerhalb des Collection-Systems (Abs. 3.2.1) wurden die Phasen Erstellen, Erfassen, Konvertieren und Aggregieren vorgestellt. Diese Schritte wurden in den CLC entsprechend übernommen.

Im Folgenden wird der Content zu Beginn seines Lebenszyklus durch einen Autor erstellt. Das Erstellen des Content wird normalerweise auf das spätere Publizieren ausgerichtet. Daher muss zwischen zwei Arten von Autoren unterschieden werden. Autoren, die den Content speziell für das hier vorgestellte Publikationssystem erzeugen und Autoren deren Content auch für andere Systeme erstellt wurde. Je nach Art des Content muss nach dem Erstellen der Content erfasst werden. Das Erfassen von Content kann z.B. durch das Scannen von Texten oder durch das Senden einer Email geschehen. Abhängig vom binären Format und der Struktur, muss gegebenenfalls eine Konvertierung durchgeführt werden. In der Stufe des Aggregierens wird der Content zum einen auf seine Korrektheit in Bezug auf bestimmte redaktionelle Richtlinien überprüft, zum anderem wird er segmentiert und mit Metadaten versehen.

Die Stufen bearbeiten, prüfen, genehmigen/ablehnen und testen wurden in Abschnitt 3.2 nicht betrachtet. Sie sind aber für den redaktionellen Teil des Publikationsprozesses von großer Bedeutung.

In der Stufe bearbeiten werden verschiedene Content Elemente zusammengefasst und in Beziehung zueinander gesetzt. Die eigentliche Publikation bzw. das Multimediadokument wird hier aus Content erstellt. Nach dem Bearbeiten muss der Content überprüft werden. Wichtig ist, dass kein Fehler beim Bearbeiten entstanden ist. Danach sollte in Abhängigkeit des Dokumententyps noch eine Überprüfung durch eine berechtigte Person stattfinden, ob eine Veröffentlichung genehmigt wird. Bei positiver Entscheidung erfolgt eine letzte Fehlersuche.

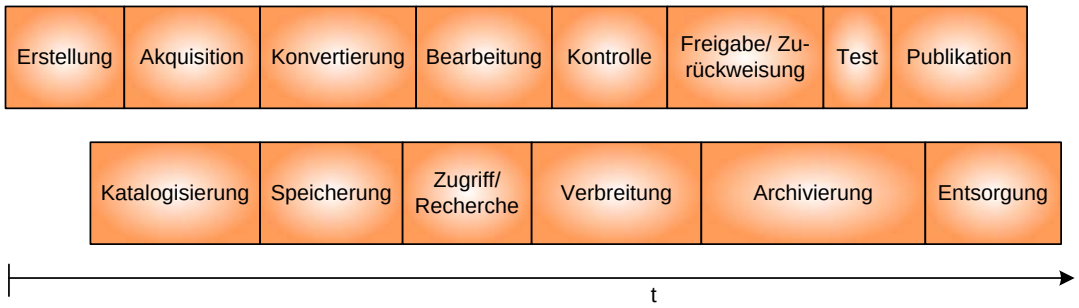

Abbildung 3.3: Content Life Cycle im Content-Management und digitalen Bibliotheken [HM05]

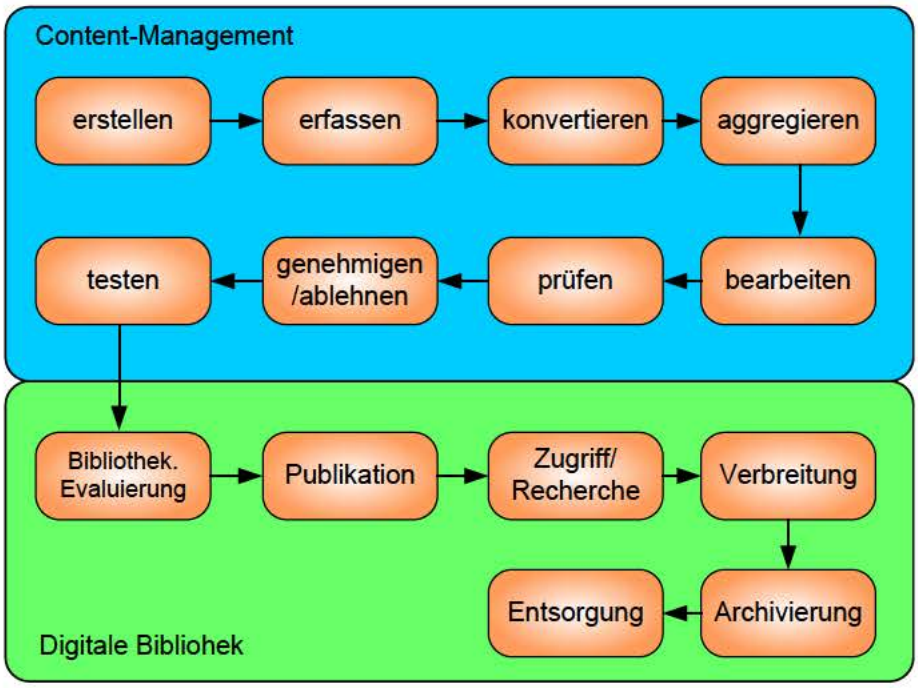

Abbildung 3.4: Stufen des *Content Life Cycle*

Anders als beim CM wird das Dokument nicht sofort publiziert. Entsprechend den Anforderungen der digitalen Bibliothek muss gegebenenfalls eine bibliothekarische Evaluierung erfolgen. Das weitere Vorgehen in diesem Fall, wird zu einem späteren Zeitpunkt betrachtet.

Nach der Evaluierung durch die Bibliothek, wird der Content publiziert.
Der Zugriff und die Recherche bieten dann die Möglichkeit den gespeicherten Content zu suchen und zu verwenden.

Die Schritte Verbreitung, Archivierung und Entsorgung sind Funktionen einer digitalen Bibliothek und werden hier nicht weiter betrachtet.

3.3.2 Stufen des Content Life Cycle im Detail

Die einzelnen Stufen des CLC wurden im vorherigen Abschnitt 3.3.1 vorgestellt. Im Folgenden werden die Stufen des Publikationsprozesses für Multimediadokumente im Detail erläutert. Dabei wird versucht, die Funktionen der einzelnen Stufen sehr genau zu beschreiben um im weiteren Verlauf der Arbeit den Publikationsprozess auf einzelne Funktions-Module abbilden zu können. Die einzelnen Schritte müssen so konzipiert werden, dass Multimediadokumente damit erzeugt werden können, die den Anforderungen aus Abschnitt 2.2 entsprechen.

Das Erstellen von Content

Autoren erstellen ihre Dokumente/Content mit Hilfe von Autorenwerkzeugen. In Bezug auf den Publikationsprozess der digitalen Bibliothek stellen sich die Fragen, ob Autorenwerkzeuge erweitert oder eigene Autorenwerkzeuge zur Verfügung gestellt werden.

In Abschnitt 3.2.1 wurden zwei Varianten vorgestellt, um Content zu erfassen.

- eigene Entwicklungen, die integriert werden
- Erweiterung vorhandener Autorenwerkzeuge

Für den Publikationsprozess in digitalen Bibliotheken ist die Bereitstellung von integrierten Lösungen für die Erstellung von Content nicht sinnvoll.

Die bessere Variante besteht darin, vorhandene Autorenwerkzeuge mit neuen Funktionen zu erweitern. Die bereitgestellte Funktionalität bietet die Möglichkeit, den Content direkt im hier entwickelten System zu speichern. Dabei müssen die Daten in Struktur und binärem Format den Vorgaben des Systems entsprechen. Dieser Ansatz hat den Vorteil, dass der Autor mit seinem gewohnten Werkzeugen arbeiten kann. Zusätzlich können gegebenenfalls auch Konvertierungen des Content entfallen. Im Rahmen dieser Arbeit soll die Möglichkeit geschaffen werden, dass eine Schnittstelle für das hier entwickelte System zur Verfügung steht, die es ermöglicht, dass für Werkzeuge eine solche Erweiterung bereitgestellt werden kann. *Eine Erweiterung an sich ist nicht Ergebnis dieser Arbeit.* Diese Vorgehensweise verfolgt das Ziel, gegenüber Autorenwerkzeugen offen zu bleiben. Autoren Werkzeuge verwalten oft bestimmte Metadaten über den erzeugten Content und stellen diese gegebenenfalls auch zur Verfügung. Die Einbindung dieser Daten ist von Interesse, da der Autor in diesem Fall Metadaten nur einmal eingeben muss.
Zusammenfassend besteht die Stufe *Erstellen* aus den Möglichkeiten:

- vorhandene Werkzeuge mit Funktionen erweitern, die das direkte Speichen im Publikationssystem ermöglichen.
- Content zu erstellen und dann durch den Schritt *Erfassen* in das Publikationssystem zu laden.

die wichtig sind.

Kann das Autoren Werkzeug nicht erweitert werden, weil z.B. keine Programmierschnittstellen zur Verfügung stehen, muss der Content auf eine andere Art in das System geladen werden.Im Folgenden Abschnitt werden weitere Methoden vorgestellt.

Das Erfassen von Content

Für den Publikationsprozess in digitalen Bibliotheken muss eine allgemeine Anbindung für den Upload von Content geboten werden.

Das Erfassen kann auf verschieden Weise vorgenommen werden. Ein Weg ist das Laden per Web Formular. Dabei stellt das Publikationssystem Formulare zur Verfügung, die zum einen Möglichkeiten des Dateiupload bieten und zum Anderen Felder für die Beschreibung des Inhalts. Mit Inhalt ist in diesem Fall der Content Typ gemeint, der für die weiter Verarbeitung wichtig ist.

Spezielle Tools können ebenfalls eingesetzt werden, um Content in das Publikationssystem zu laden. Diese Tools sind von Spezialisten direkt auf einen Anwendungsfall und meist einen Content Typ ausgelegt. Zusätzlich unterstützen sie die Möglichkeit des *bulk loading*.

Eine weitere Möglichkeit bieten z.B. Schnittstellen wie Webdav[4] oder FTP[5]. Der Content kann hier auf einem Server in dafür bereitgestellten Bereichen abgelegt werden. Diese Bereiche werden vom Publikationssystem entweder automatisch oder manuell nach Material durchsucht. Um den Content Typ zu bestimmen, kann der abgelegte Content z.B. in einem offenen Format wie z.B. zip gewrappt sein. Die zip Datei enthält in einer zusätzlichen XML Datei Informationen über den Inhalt und die Anzahl der enthalten Dateien. Ein ähnliches Verfahren kann bei Emails angewendet werden. Den Bereich auf dem Server ersetzt ein Email Server und eine entsprechende Email Adresse.

Anders als bei CMS, muss ein Publikationssystem für digitale Bibliotheken *content harvesting* nicht bereit stellen. Die in Abschnitt 3.2.1 vorgestellte Stufe Erfassen betrachtet auch die Möglichkeit, dass Content von bestimmten Quellen automatisch bezogen wird. Dabei soll im Begriff Erfassen das *harvesting* ausdrücklich nicht enthalten sein. Content soll aktiv bzw. nicht automatisch, auf den oben beschriebenen Wegen, in das System eingestellt werden. Dies basiert auf den folgenden Entscheidungen.

1. Wenn Content automatisch bezogen wird, ist die Quantität nur schwer zu kontrollieren.

2. Die weiteren Stufen des CLC müssten automatisch ausgeführt werden. Damit würde die Kontrolle der Qualität nicht mehr gesichert sein.

3. Werden die weiteren Stufen nicht automatisch ausgeführt, sind mehr Mitarbeiter zur Ausführung einzuplanen.

In Abschnitt 3.2.1 wurde bereits erwähnt, dass der erfasste Content komplett in das eigene Speichersystem überführt werden sollte, um einen Datenverlust zu verhindern. Zusammenfassend werden folgende Punkte der Stufe Erfassen für die weitere Arbeit als wichtig erachtet.

- das Laden mit Hilfe eines Web Interfaces bzw. eines Web Formulars

- Bereitstellung von speziellen Upload Tools, die auf bestimmte Content Typen spezialisiert sind

[4] `http://www.webdav.org/`
[5] `http://www.scit.wlv.ac.uk/rfc/rfc9xx/RFC959.html`

- Upload über Schnittstellen wie Webdav, FTP oder Email auf einen Server
- das Zwischenspeichern der erfassten Dateien

Das Konvertieren von Content

Das Konvertieren von Content beinhaltet die in Abschnitt 3.2.1 beschriebenen Schritte:

- entfernen von Inhalt
- mapping der Binärdaten
- mapping der Struktur

Im Publikationsprozess muss es möglich sein, verschiedene Content Typen zu konvertieren. Die Implementierung ist abhängig vom Content Typ und muss für jeden einzelnen Typ bereitgestellt werden.

Das Aggregieren von Content

Nach dem Content in das Publikationssystem durch oben genannte Methoden geladen und konvertiert wurde, muss er aggregiert werden. In Abschnitt 3.2.1 wurde das Aggregieren in drei Schritte unterteilt:

- redaktionelle Verarbeitung
- Segmentierung
- Angabe von Metadaten

Im Publikationsprozess soll davon ausgegangen werden, dass Content nicht automatisch aus (fremden) Quellen bezogen wird. Darum wird festgelegt, dass Autoren schon bei der Erstellung des Content dafür verantwortlich sind, den Content nach bestimmten Richtlinien zu erzeugen. Daraus folgt, das keine inhaltliche Bearbeitung im Publikationsprozess stattfindet. Diese Einschränkung wird gewählt, weil das Editieren des Content, wie bereits erwähnt, nur mit Autorenwerkzeugen möglich ist. Eine Integration dieser Werkzeuge für nachträgliche Korrekturaufgaben ist nicht sinnvoll. Ein weiterer Punkt, der zu dieser Einschränkung führt, ist dass Autoren im Normalfall den Content selbst erzeugen und eigenständig in das Publikationssystem einstellen. Als Beispiel ist eine pdf Datei zu nennen, die nicht grundlegend verändert werden kann.

Das Segmentieren soll komplex strukturierten Content in nicht komplexen Content zerlegen, der im Repository abgelegt werden kann. Um eine sinnvolle Aufteilung zu gewährleisten, sollte für jeden Content Typ festgelegt werden, welche Arten von Content extrahiert werden kann. Entsprechend dem vorhandenem und dem zu erstellendem Content Typ, kann die Extraktion

- automatisch,
- halbautomatisch,
- oder manuell

erfolgen. Wie Content extrahiert wird, hängt zum einen von den technischen Möglichkeiten ab, zum anderen vom technischen Aufwand der betrieben werden soll.

Eine automatische Segmentierung macht z.B. Sinn, wenn der Content strukturiert vorliegt. Ist der Content in einer XML Struktur angegeben, kann relativ einfach mit entsprechendem Wissen über die Struktur eine Segmentierung vorgenommen werden.

Das Segmentieren wird häufig als Teilprozess des Konvertierens ausgeführt. Für die konzeptionelle Beschreibung wurde, wegen der besseren Übersichtlichkeit, die Unterteilung nach [Boi01] gewählt.

Zusätzlich kann eine Zerlegung anhand der Semantik von Content vorgenommen werden. Dies ist durch das verwendete Datenmodell des Repository eingeschränkt. Dokumentenmodelle wie *MPEG-7* (siehe Abschnitt 2.3) unterstützen aber die Verwendung dieser Daten. Dabei kann z.B. versucht werden, Objekte innerhalb von Bildern oder Videos zu erkennen. Diese Objekte z.B. durch Umrisse auf einem Bild oder Eigenschaften eines Bildausschnitts beschrieben.
Durch die Segmentierung kann der Content in Content Elemente zerlegt werden. Die in Abschnitt 2.2 geforderte Wiederverwendbarkeit von Multimediadokumenten wird durch die Segmentierung unterstützt.

Wie in Abschnitt 2.1 beschrieben, lassen sich Metadaten in verschiedene Klassen, die auf unterschiedliche Weisen erzeugt werden, einteilen.

In Multimedia Datenbanken ist die Ähnlichkeitssuche ein wichtiges Verfahren, um Medienobjekte zu finden. Die Ähnlichkeit von zwei Objekten innerhalb eines Suchraums wird durch eine Ähnlichkeitsmetrik bestimmt. Die Mertrik ermittelt dabei den Abstand der Objekte im Suchraum. [SHS05]
Um das Information Retrieval für Multimediadokumente zu unterstützen, müssen entsprechende Features aus dem Content extrahiert und zu Vektoren zusammengefasst werden, die zusammen mit dem Content abgelegt werden. Verfahren zur Feature-Extraktion können

- manuell für inhaltsbezogene Features
- halbautomatisch oder automatisch für inhaltsbeschreibende Features

sein.

Ein Feature ist eine abgeleitete Eigenschaft eines Segments, dass die Suche unterstützen soll. Einzelne Features werden zu einem Feature-Vektor zusammengefasst. Das heißt, ein Feature-Vektor ist ein hochdimensionaler Vektor, der bei der Ähnlichkeitssuche eingesetzt werden kann.

Inhaltsbeschreibende Metadaten müssen *manuell* durch den Benutzer erstellt werden. Die Eingabe kann z.B. über Eingabemasken erfolgen. Wenn der Nutzer die Metadaten bereits in einem anderen System vergeben hat und diese weiter verwenden möchte, müssen die Daten importiert werden. Das Bildformat *JPEG* bietet beispielsweise die Möglichkeit, Exif[6] und IPTC[7] Informationen in der Datei mit anzugeben. Entsprechende Programme wie Adobe Photoshop können diese Angaben mit in der Datei ablegen. Eine automatische Extraktion bietet die Möglichkeit die Daten weiter zu verwenden.

Eine semiautomatische Extraktion bietet sich bei Informationen an, die automatisch erzeugt werden können, aber eine hohe Fehlerquote haben. Sie müssen später nachbearbeitet werden.

Die *automatische* Extraktion ist nur bei inhaltsbezogenen Metadaten sinvoll. Dabei können z.B. Farbhistogramme, Auflösung und Größe von Bildern ermittelt werden (siehe Abschnitt 2.1).

Ein wichtiger Punkt bei der Vergabe von Features in Bibliotheken ist die Vergabe von Schlagwörtern und Stichworten oder die Einordnung in Klassifikationen. Schlagwörter sind in der Schlagwortnormdatei (SWD) zusammengefasst und normiert.[Hac97] Aufgrund des enormen Umfangs der Normdatei, ist eine manuelle Vergabe von Schlagworten durch einen Autor nicht sinnvoll. Sie entfällt ebenfalls bei einer digitalen Bibliothek. Nach [Hac97] sind Stichwörter charakterisierende Worte, die den Inhalt eines Dokumentes beschreiben. Sie werden dem Sachtitel oder dem Zusatz zum Sachtitel entnommen. Da in digitalen Bibliotheken der Titel eines Dokumentes Volltext durchsuchbar ist, kann die explizite Vergabe entfallen. Klassifikationen bieten nach [EF00] die Möglichkeit, Aussagen über die Semantik des Suchraums zu treffen. Durch die Verwendung von Klassifikationen kann die Suche erheblich verbessert werden. Weitere Techniken, die in diesem Zusammenhang genannt werden, sind das *Wörterbuch* und der *Thesaurus*.

Wichtig in diesem Schritt ist, dass der Content so aufbereitet wird, dass die Auswahl und Identifikation durch entsprechende Suchmechanismen der Anwendung möglich ist. Weiterhin müssen die Daten so aufbereitet sein, dass eine Ähnlichkeitsberechnung effizient möglich ist. Die Daten müssen nach der Segmentierung und der Vergabe von Metadaten im Repository abgespeichert werden. Diese Variante hätte den Vorteil, dass die Daten für das eigentliche Multimediadokument aus dem Repository geholt werden können. Es ist keine Unterscheidung von Content aus dem Repository und dem neu erstellten Content mehr zu treffen. Dabei müssen die Daten so markiert werden, dass sie nur für entsprechend berechtigte Benutzer sichtbar sind, da sie noch nicht publiziert wurden.

Das Bearbeiten

In diesem Schritt wird innerhalb des Publikationsprozesses das Multimediadokument erzeugt. Der Autor hat die Möglichkeit, den im vorherigen Abschnitt erstellten Con-

[6] `http://www.exif.org/`
[7] `http://www.iptc.org/`

tent sowie schon in der digitalen Bibliothek vorhandene Publikationen und Content zu einer neuen Publikation zusammen zu fügen. Die Verwendung von Publikationen und Content, die bereits im System vorhanden sind, ist hier von besonderem Interesse, weil damit das Prinzip der Wiederverwendbarkeit von Inhalt unterstützt wird. Die Modellierung des Dokumentes wird auf der Basis des zugrundeliegenden Dokumentenmodells erstellt, welches für den Publikationsprozess verwendet wird. Entsprechend dieser Anforderungen werden die einzelnen Komponenten in Beziehung gesetzt. Dabei müssen z.B. die genannten Anforderungen aus Abschnitt 2.2 wie zeitliche, räumliche oder interaktive Beziehungen zwischen einzelnen Content Elementen modelliert werden. An dieser Stelle müssen Editoren für die Modellierung von Struktur zur Verfügung gestellt werden.

Zusätzlich zu den strukturellen Metadaten müssen für das neue Multimediadokument entsprechende bibliografische Metadaten angeben werden. Eine Deskribierung entsprechend der im vorherigen Abschnitt beschriebenen Vorgehensweise ist nicht notwendig, da die einzelnen Content Elemente bereits mit Feature-Vektoren versehen sind.

Das Prüfen des Content

Die Stufe *prüfen* soll es ermöglichen, je nach Content Typ zu testen, ob die Darstellung korrekt ist und ob der Inhalt richtig ist. Der Nutzer soll die letzte Möglichkeit haben, den Content der in das System eingestellt werden soll, zu kontrollieren. Die Kontrolle muss in Abhängigkeit des Content Typs durchgeführt werden, da z.B. ein Video nicht komplett angeschaut werden sollte.

Genehmigen/Ablehnen und Testen von Content

Nachdem der Content vom Autor geprüft und zum publizieren freigegeben wurde, kann gegebenenfalls eine weitere Instanz das Dokument überprüfen. Diese Instanz liegt nicht auf Seiten der Bibliothek, die als Betreiber der digitalen Bibliotheken fungiert. Vielmehr kann es sich hier um ein Organ innerhalb der Organisation des Autors, z.B. dessen Vorgesetzter handeln.

Dieser Schritt ist im Interesse des Lesers wichtig, um die Qualitätssicherung zu gewährleisten. Er stellt aber keinen Zwang dar. Im konventionellen Publikationsprozess übernehmen die Verlage die Kontrolle der Veröffentlichungen. Inwieweit eine Kontrolle entsprechend dem Konzept Gutachter und Lektor wie es in [EF00] beschrieben ist, umgesetzt wird, ist abhängig von der Anwendung. Eine Qualitätssicherung sollte dazu eine Reihe von Kriterien definieren, die bei der Kontrolle unterstützt werden. [EF00] schlägt dafür konkret Kriterien vor.

Nachdem das Dokument freigegeben wurde, wird es ein letztes mal, geprüft und an die digitale Bibliothek übergeben. Mit Abgeben ist das endgültige Freigeben des Dokumentes zur Publikation gemeint. Um die genannten Aufgaben zu verwalten und zu koordinieren ist der Einsatz eines Workflow-Management-Systems möglich. Bereits in Abschnitt 3.2.2 wurde die Verwendung eines solchen Systems vorgestellt. Die Entscheidung auf welche

Art und in welchem Umfang ein solches System genutzt wird, liegt aber auf Seiten der Anwendung. Hier können keine konkreten Vorgaben gemacht werden.

Bibliothekarische Evaluierung des Dokumentes

Nach der Freigabe des Dokumentes muss anhand von bestimmten Kriterien entschieden werden, ob ein Dokument bibliothekarisch evaluiert wird. Die Kriterien, ob und wie ein Dokument bearbeitet wird, wird von den Bibliotheken individuell festgelegt. In der digitalen Bibliothek der UB Rostock werden beispielsweise Dokumentenklassifikationen als Kriterium verwendet. Die Klassifikation ordnet das Dokument anhand seines Typs in eine Archivierungsklasse ein. Weiterhin wird durch die Klassifikation festgelegt, ob die Bibliothek eigene Metadaten für das Dokument vergeben muss, oder ob überhaupt einer Bearbeitung durch einen Bibliotheksmitarbeiter notwendig ist.

Wie ein Dokument verarbeitet wird, kann auch hier unter die Kontrolle eines Workflow-Management-Systems gestellt werden.

Publikation

Die Publikation von Multimediadokumenten beschränkt sich auf die Freigabe des Dokumentes. Um eine Kontrolle und bibliothekarische Evaluierung zu ermöglichen, wird das Dokument im Normalfall schon im Repository gehalten. Dabei ist es bis zum Punkt der Publikation nicht möglich, dass Dokument durch Suchmechanismen zu finden.

An dieser Stelle können weiterführende Überlegungen angebracht werden, die die Bearbeitung der Dokumente betrachten, die bereits in der digitale Bibliothek publiziert sind. Beispielsweise könnte, ähnlich dem Konzept eines Wikis, die Möglichkeit für einen ausgewählten Nutzerkreis bestehen, vorhandene Dokumente zu bearbeiten. Dabei könnten Inhaltliche sowie strukturelle Veränderungen der Dokumente möglich sein. Dokumente in bestimmten Archivierungsklassen könnten durch den Autor verändert werden. Dies würde z.B. eine Versionkontrolle von Dokumenten erfordern. Dies soll aber nicht Teil dieser Arbeit sein.

4 Entwurfsmuster und Frameworks

Die Umsetzung des in Abschnitt 3.3.2 vorgestellten Publikationsprozesses erfolgt mit Hilfe von Modulen. Der Publikationsprozess wird für Multimediadokumente in digitalen Bibliotheken entwickelt. Daraus ergibt sich die Forderung, dass einzelne Komponenten oder auch komplette Module entsprechend den Anforderungen des zu publizierenden Dokumentes austauschbar sind. Diese Aufgabe soll mit Hilfe eines Frameworks gelöst werden, welches eine Reihe von Interfaces dafür zur Verfügung stellt. Um eine hohe Qualität zu gewährleisten, sollen Verfahren aus dem Bereich der Softwaretechnik angewendet werden. Entwurfsmuster und Frameworks bieten eine gute Möglichkeit die Qualität des Frameworks sicher zu stellen und eine Wiederverwendbarkeit zu ermöglichen. Im Folgenden wird deshalb ein kurzer Überblick über wichtige Entwurfsmuster und Frameworks gegeben.

4.1 Entwurfsmuster

Der objektorientierte Entwurf soll durch die Verwendung von Entwurfsmustern (auch Pattern) unterstützt werden. Sie geben Lösungen für immer wiederkehrende Probleme im Entwurf. Durch die Muster wird es möglich, die Entwürfe wieder zu verwenden. Ein Entwurfsmuster benennt, abstrahiert und identifiziert nach [GHJV01] die relevanten Aspekte einer allgemeinen Entwurfsstruktur.

Im Folgenden werden verschiedene Techniken vorgestellt, die bei der Erstellung des Frameworks für den Publikationsprozess von Bedeutung sind. Dadurch lässt sich eine hohe Wiederverwendbarkeit des Frameworks erreichen.

Wichtige Mechanismen für die Wiederverwendbarkeit sind die Objektkomposition und die Vererbung. Das Prinzip der Vererbung basiert auf dem Prinzip der Unterklassenbildung und wird als *White-Box-Wiederverwendung* bezeichnet. Eine Unterklasse wird auf Basis einer Oberklasse definiert. Dabei werden Definitionen und Daten der Oberklasse an die Unterklasse übergeben. Abstrakte Klassen stellen einen Spezialfall dar, bei dem eine Schnittstelle für gemeinsame Unterklassen definiert wird. Eine abstrakte Klasse kann nicht erzeugt werden und ihre Operationen sind nicht implementiert.

Die Definition von abstrakten Klassen, also Schnittstellen, ermöglicht es Objekte gleichen Typs anzusprechen. Der Typ bezeichnet hier die Schnittstelle eines Objektes. Ein Objekt kann somit verschiedene Typen haben, aber immer nur eine Implementierung. Das Prinzip der Polymorphie ermöglicht es, Objekte zur Laufzeit auszutauschen. Wenn man Objekte nur über abstrakte Schnittstellen manipuliert, ergeben sich folgende Vorteile:

1. Die Implementierung der Objekte wird gekapselt. Der Klient kann das Objekt nur über die definierte Schnittstelle verwenden.

2. Klienten wissen nichts über die Implemtierung der jeweiligen Klassen. Es sind ihnen nur die jeweiligen Klassen bekannt, die die Schnittstelle implementieren.

In [GHJV01] wird folgende Schlussfolgerung gezogen:

> "Programmiere auf eine Schnittstelle hin, nicht auf eine Implementierung."

Die Objektkomposition basiert darauf, dass komplexe Funktionalität durch das Zusammenfügen von Objekten erreicht wird. Die Objekte müssen mit wohldefinierten Schnittstellen versehen sein. Dies ermöglicht die so genannte *Black-Box- Wiederverwendung*. Hierbei ist, wie oben beschrieben, nur die Schnittstelle zu einem Objekt Typ bekannt. Die konkrete Implementierung bleibt nach außen verborgen. Die Objekte sind durch die Verwendung von Schnittstellen gekapselt. Dies führt dazu, dass jedes Objekt zur Laufzeit von einem anderen ersetzt werden kann.

Die Vererbung hat nach [GHJV01] folgende Nachteile:

- Die geerbte Implementierung von Oberklasse kann nicht zur Laufzeit geändert werden. Die Vererbungsstruktur wird zur Laufzeit festgelegt.

- Die Unterklasse ist von der Oberklasse abhängig. Jede Änderung in der Oberklasse hat auch Auswirkungen in der Unterklasse.

- Es treten Probleme auf, wenn bestimmte Aspekte bei der Unterklasse nicht den der neuen Anwendung angemessen ist. Dies führt dazu, dass die Oberklasse geändert werden muss, was dann wiederum Auswirkungen auf anderen Unterklassen hat.

- Die Lösung der Probleme besteht darin, nur noch von abstrakten Klassen zu erben. Dadurch bestehen keine Abhängigkeiten an bestimmte Implementierungen mehr.

In [GHJV01] wird folgende Schlussfolgerung gezogen:

> "Ziehe Objektkomposition der Klassenvererbung vor."

Eine weitere interessante Technik ist die Delegation. Ein Objekt gibt eine Anfrage an ein anderes Objekt weiter. Im Klassenbaum stellt sich diese Beziehung dadurch dar, dass Unterklassen Anfragen an Oberklassen durchreichen. Das Prinzip bei der Delegation besteht darin, dass Objekte nicht von Oberklassen abgeleitet werden, um bestimmtes Verhalten zu erben, sondern entsprechendes Verhalten der Klassen wiederverwenden. Ein Objekt verwaltet eine Referenz auf ein Objekt, an dass Verhalten delegiert wird. Ein Objekt besitzt also ein anderes Objekt anstatt es zu sein. Das Prinzip erleichtert damit das Zusammensetzen von Verhalten zur Laufzeit.

Der Entwurf von Frameworks basiert häufig auf den im Folgenden vorgestellten Entwurfsmustern.

Interface Ein Interface trennt die Beschreibung einer Klasse von ihrer konkreten Implementierung. Klassen können mehrere Interfaces implementieren. Ein Interface kann von mehreren Klassen implementiert werden.

Singleton Das Singleton Pattern stellt sicher, dass von einer Klasse zur Laufzeit genau ein Objekt erstellt wird. Dabei wird eine globale Zugriffsmöglichkeit auf das Objekt bereitgestellt und bei der ersten Verwendung wird das Objekt erstellt.

Proxies Ein Proxy kontrolliert die Verwendung eine Objektes durch ein vorgelagertes Stellvertreter Objekt. Proxies fungieren als Platzhalter und verhalten sich genau so, wie die eigentlichen Objekte. Er ermöglicht es, die Verwendung des Objekts zu kontrollieren.

Pooling Das Pooling Pattern bietet die Möglichkeit zu kontrollieren, wie viele Objekte von einer Klasse erzeugt werden. Ein Beispiel hierfür ist das Singleton Pattern.

Factory Das Factory Pattern ist ein Hilfsmittel zum Erzeugen von Objekten, bei dem keine konkrete Klasse benannt werden muss. Häufig wird das Pattern eingesetzt, wenn die Menge der Klassen, die ein Objekt erzeugen, dynamisch und zur Laufzeit erweiterbar ist. Zwei wichtige Vertreter sind die *Fabrikmethode* und die *Abstrakte Fabrik*.

Command Das Command Pattern kapselt Befehle in Objekte. Dieses Vorgehen ermöglicht es, Warteschlangen, Logbücher und Redo Informationen für Befehle zu Implementieren. Außerdem können Befehle parametrisiert werden.

Inversion of Control (IoC) Dieses Pattern erlaubt es, dass ein Container für die Erstellung, das Initialisieren und das Konfigurieren einer Komponente verantwortlich ist. Der Container ruft entsprechende Methoden der Komponente auf um die genannten Aufgaben zu erfüllen.

Dependency Injection Ist eine Speziallform des IoC. Dabei werden Konfigurationen und ähnliches über Methoden dem Objekt *injiziert*. Das Konzept besteht darin, dass das Objekt nicht seinen Container nach Konfigurationen oder anderen benötigten Objekte fragt, sondern der Container diese Informationen selber ermittelt. Dies kann der Container über Methoden Signaturen und Konfigurationsdateien ermitteln und dann dem Objekt bereitstellen.

4.2 Frameworks

4.2.1 Überblick

Für die Integration eines Publikationsprozesses in *MyCoRe* soll ein Framework verwendet werden. In [GHJV01] wird ein Framework als eine "[...] Menge von zusammenarbeitenden Klassen [definiert], die einen wiederverwendbaren Entwurf für eine bestimmte Klasse von Software darstellen". Durch die Verwendung eines Frameworks soll dem

Entwickler die Architektur der Anwendung vorgegeben werden. Dabei beschreibt das Framework die Struktur der Anwendung und deren Aufteilung in Klassen und Objekte. Ein weiterer wichtiger Faktor ist die Festlegung der Beziehungen der Klassen untereinander. Damit werden Kontrollfluss, Zuständigkeiten und Zusammenarbeit definiert. Bei einem Framework liegt das Interesse mehr auf der Entwurfswiederverwendung als auf der Codewiederverwendung. Die Abstraktion eines Frameworks in Module ermöglicht deren separates Implementieren. Durch die Verwendung von Modulen kann die Wiederverwendbarkeit der bereitgestellten Komponenten und des implementierten Codes gesteigert werden.

Für die Entwicklung mit Java stehen eine Reihe von Frameworks zur Verfügung. Meta-Frameworks sind im Zusammenhang mit dieser Arbeit von besonderem Interesse, da andere Frameworks integriert werden können. Ein Meta-Framework beschreibt dabei ein Framework, dass andere Frameworks integriert. Wichtig ist hierbei, dass vorhandene Frameworks wie z.B. Hybernate oder Cocoon dem Entwickler vorkonfiguriert zur Verfügung gestellt werden können.

In diese Arbeit wurden die Open-Source Java Frameworks ***Spring*** [1], ***x18p*** , ***Avalon*** [2] und ***Keel*** [3] untersucht.

4.2.2 *Spring* Framework

Spring ist ein Open-Source Meta-Framework, dass zum Teil auf Java Bean Konzepten aufbaut. Das Komponentenmodell und die Property-Editoren sind zwei Beispiele dafür. Um das Programmieren zu vereinfachen, wird versucht, möglichst viele Komponenten durch vorhandene Open-Source Frameworks zur Verfügung zu stellen. Das Framework stellt z.B. keine eigenen Pakete für das Logging zur Verfügung. Es wird auch keine eigene Persistenzsicherung verwendet. Es werden externe O/R mappings verwendet, wobei Lösungen wie Hibernate und JDO zur Auswahl stehen. Um die Arbeit mit Datenbanken weiter zu vereinfachen, wird ein eigenes JDBC-Abstraktions-Framework bereitgestellt. Das Framework soll eine bessere Fehlerbehandlung und informativiere SQLExceptions gegenüber JDBC bieten. Für die Darstellung wird weiterhin ein eigenes MVC Web Framework zur Verfügung gestellt.

Die Architektur von *Spring* basiert auf verschiedenen Schichten. Die Teile der jeweiligen Schichten können völlig isoliert verwendet werden. So kann der Nutzer entscheiden, welche Teile des Frameworks er verwenden möchte. Die Service Abstraktion/ Komponentenmodell ist Java basiert. Properties werden in *Spring* mit Java Bean Properties oder Konstruktor Attributen ausgelesen. Dabei werden proprietäre XML Dateien Verwendet. Dies bietet dem Nutzer die Möglichkeit, seine Konfigurationsdateien einheitlich zu verwalten und zur Verfügung zu stellen. Ein wichtiges Konzept von *Spring* ist die *bean factory*. Sie ermöglicht es, Objekte anhand von Namen zu erstellen. Der Name

[1] `http://www.springframework.org/`
[2] `http://avalon.apache.org`
[3] `http://www.keelframework.org/`

wird als ID in einer entsprechenden XML Datei verwendet, die *bean definitions* enthält. In dieser XML Datei können Klassen und deren Eigenschaften definiert werden. Mit Hilfe eines *Service-Locators* können die Objekte instanziiert werden. Zusammen mit den Prinzipien *Inversion of Control* (IoC) und *Dependency Injection*, die in Abschnitt 4.1 erläutert wurden, wird die Verwendung von Properties weiter vereinfacht. Zum einen kann in der Konfigurationsdatei unterschieden werden, welche der beiden Arten von Objekten in *Spring* verwendet werden sollen:

- **Singleton** stellt eine Objekt-Instanz zur Verfügung.
- **Prototype** wird immer dann verwendet, wenn kein Singleton benutzt wird. Es werden unabhängige Objekte erstellt.

Jede so genannte *bean definition* kann z.B. ein POJO sein oder ein *FactoryBean*. Weiter Konzepte wie z.B. AOP (aspect orientated programming) sind ebenfalls möglich.

Durch das Prinzip des *Dependency Injection* können Konfigurationen direkt in der bean definition angegeben werden, aber auch durch aufrufende Objekte. *Spring* unterscheidet zwei Möglichkeiten, um die dependency injection umzusetzen:

- Properties können per setter Methoden gesetzt werden.
- Properties können als Konstruktor-Attribute übergeben werden.

Das Logging wird durch Commons Logging bereitgestellt. Der Zugriff auf einzelne Service Implementierungen wird über Bean Properties realisiert. Dabei bietet das System leider keine Modulabstraktion im Build-System.

4.2.3 *x18p* Sample Framework

Das *x18p* Framework wird in [Wu04] vorgestellt. Ein Ziel des Frameworks besteht darin, die feste Verdrahtung von Code zu verhindern. Es wird besonders auf die häufig in der Web-Schicht von J2EE Web Applikationen vorhandenen Referenzen auf Objekte anderer Schichten eingegangen. Weiterhin sollen Probleme mit anderen Frameworks wie z.B. *Spring* gelöst werden. Diese Frameworks bieten die Möglichkeit, mit Hilfe des Inversion-of-Controll (IoC) Pattern, entsprechende Objekte zu erzeugen. Im Falle des *Spring* Frameworks werden Objekte z.B. über Service-Locator initialisiert. Dabei müssen immer noch die resultierenden Objekte in den erwarteten Objekttyp konvertiert werden. Dieses Problem soll durch das *x18p* Framework behoben werden.

Das *x18p* Framework verfolgt den Ansatz, dass alle Funktionalität als Service angeboten wird. Um dieses Konzept umzusetzen, werden folgende Annahmen gemacht:

- Es wird eine Service-Vermittler-Schicht eingeführt, die zwischen der Anwendung und der Geschäftslogik vermittelt.

- Grundsätzlich wird eine Anfrage an die Geschäftslogik an einen Service-Router weitergeleitet. Dem Service-Router ist bekannt, wie er die Anfrage an den Services-Provider-Controller der Geschäftslogik weiterleiten kann. Diese Informationen werden in einer entsprechenden XML Datei abgelegt.
- Der Service-Provider-Controller kann konkrete Services einzubinden.
- Intern werden Objekte mit dem *Spring* Framework instanziiert.

Aufbauend auf diesen Annahmen wird das Framework in Abbildung 4.1 vorgestellt .

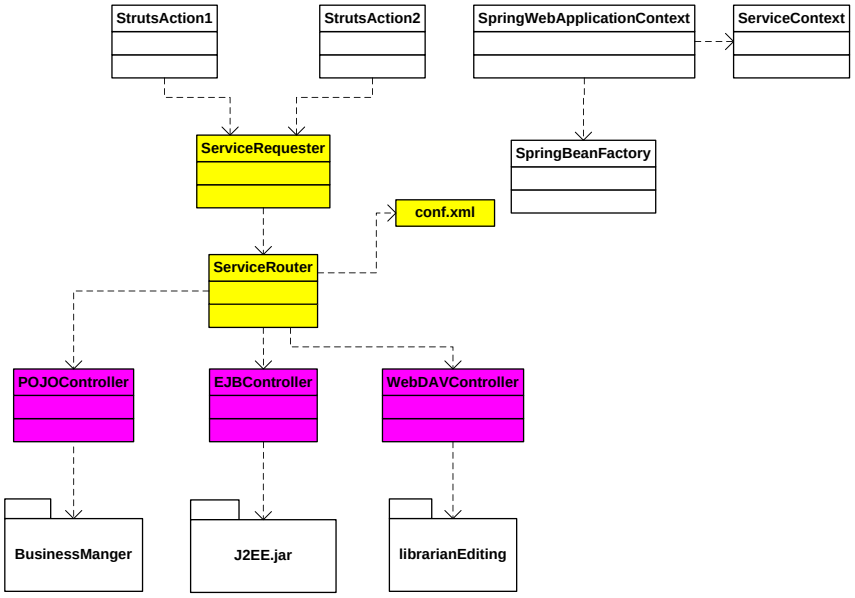

Abbildung 4.1: *x18p* Framework

Wenn eine Anwendung einen bestimmten Service in Anspruch nehmen will, fordert sie vom Service-Requester ein entsprechendes ServiceRequest-Objekt an. Innerhalb dieses Objektes werden alle Informationen gehalten, die Angaben über den Service geben und die zur Ausführung wichtig sind.

Das Objekt sendet sich anschließend selbständig weiter an den Service-Router. Der Service-Router kann nun mit Hilfe des in Abschnitt 4.2.2 vorgestellten *Spring* Frameworks einen entsprechenden Service-Controller aufrufen. Die entsprechenden Informationen stehen in der Datei *bean.xml* zur Verfügung. Zusätzlich wird eine zweite Konfigurationsdatei verwendet, in der die Service Definition zusammen mit einer Bean Definition gehalten werden. Damit soll auch der Service unabhängig von der eigentlichen Bean Definition gehalten werden. Mit der Methode getBean(...) wird der jeweilige Controller geladen. Der Service-Controller kann das ServiceRequest-Objekt auswerten und ein entsprechendes Service-Modul aufrufen, welches die Abarbeitung des Service-Request übernimmt. Durch das Prinzip des Dependency Injection kann das Service-Modul mit den entsprechenden Informationen aus dem ServiceRequest-Objekt versorgt werden.

4.2.4 *Keel* Framework

Avalon

Das Apache *Avalon* Projekt ist ein Open-Source J2SE Framework, dass die einheitliche Entwicklung von Java Komponenten ermöglicht. Seit 2004 ist das *Avalon* Framework im Projekt *Excalibur*[4] untergebracht.

Das Framework bietet die Möglichkeit Komponenten zu verwalten. Komponenten können dabei beliebige Java Klassen sein. Die Komponenten, die dem System bekannt sind, werden durch einen (Komponenten-) Container verwaltet. Der Container bietet dem Client Funktionen, um Komponenten zu finden. In *Avalon* wird die Schnittstelle einer Komponente als Rolle bezeichnet. Die Implementierung dieser Rollen werden durch den Container verwaltet und sind nur ihm bekannt. Die Anwendung ist also völlig unabhäng von den eigentlichen Implementierungen. Durch das verwendete *late-binding* kann die entsprechende Implementierung über Konfigurationsparameter bestimmt werden. Durch diesen Mechanismus wird es ebenfalls möglich, die Implementierung zur Laufzeit auszutauschen.

Auch im *Avalon* Framework spielt das Pattern *Inversion of Control* (IoC) eine zentrale Rolle. Die Idee des IoC Pattern wurde bereits in Abschnitt 4.1 beschrieben. In *Avalon* übernimmt der Container die Aufgabe der Erstellung, Konfiguration und Initialisierung der einzelnen Komponenten. Die Verwendung von einzelnen Komponenten wird durch *Marker-Interfaces* realisiert. Ein solches Interface beschreibt Eigenschaften und Anforderungen, die eine Komponente an das Runtime Environment hat. Komponenten können über diese Interfaces eingebunden werden. Soll eine Komponente z.B. konfigurierbar sein, dann muss sie das *Configurable Interface* implementieren. Diese Methode ermöglicht es der Anwendung nur Komponenten zu implementieren, die auch wirklich benötigt werden.

In der Benutzungsphase wird im *Avalon* Framework zwischen verschiedenen Komponenten unterschieden. Hier wird festgelegt, wie einzelne Komponenten dem Nutzer zugänglich gemacht werden können. Es stehen z.B. die Möglichkeiten *ThreadSafe* und *SingleThread* zur Verfügung. Weiterhin wird das jeweilige Verhalten durch ein Interface implementiert. [Zie03]

Keel

Das *Keel* Meta-Framework basiert auf dem oben vorgestellten *Avalon* Framework. Ähnlich wie das *Spring* Framework bietet *Keel* ebenfalls die Möglichkeit andere Frameworks und Dienste zu verwalten. Die Frameworks werden vorkonfiguriert zur Verfügung gestellt. Durch die Verwendung von Frameworks wie Struts, Cocoon oder Velocity werden z.B. GUI's und Webinterfaces unterstützt.

Die Architektur von *Keel* baut auf der von *Avalon* auf. Es gibt einen Keel-Container, der für die Verwaltung von Komponenten verantwortlich ist. In *Keel* ist es möglich, dass jede einfache Java Klasse eine Komponente vom Container anfordern kann. Der Zugriff auf die Service-Implementierungen wird mittels Service Locater realisiert. Das

[4] `http://excalibur.apache.org/`

Keel System wird in verschiedene Komponenten unterteilt. Jede Implementierung eines Interfaces bzw. einer Rolle ist eine Komponente. Die wichtigsten Komponenten sind in [HW04] beschrieben:

- Model: zur Implementierung von Applikations- und Geschäftslogik, enthält einen Mechanismus zur Request/Response-Verarbeitung
- Persistence: Implementierungen für JDBC,JDO, LDAP und Entity EJBs vorhanden
- Security: Umsetzung von Autorisierung und Authentifizierung, JAAS- Implementierung verfügbar, Autorisierung auf Komponenten- und Methodenebene möglich
- Scheduling: Dienst für Terminierung von Komponentenaufrufen läuft im Hintergrund ab, Default-Implementierung mit Quartz
- Crypto: Verschlüsselungsdienst mit existierenden Implementierungen basierend auf BASE64 und DES
- QUERY: Textindizierung und Suche, Default-Implementierung basiert auf Apache Lucene

Zur Konfiguration von Komponenten stellt *Keel* verschiedene Möglichkeiten zur Verfügung. Die Dateien *roles.xconf* und *system.conf* bieten die Möglichkeit per XML zu konfigurieren. Mithilfe der Javadoc Meta Tags (XDoclet) können die Konfigurationen für jede einzelne Komponente festgelegt werden. Außerdem können Module Voreinstellungen mitliefern. Diese Einstellungen können aber von der Anwendung im Bedarfsfall überschrieben werden. Der *life cycle* von Komponenten wird durch das Lifecycle-Tag mit seinen verschiedenen Anwendungen bestimmt:

- Singleton: Der Container erzeugt nur eine Instanz einer Komponente.
- Transient: Für jedes Request erzeugt der Container eine neue Instanz der Komponente und zerstört sie nach der Benutzung wieder.
- Pooled: Jedes Request entnimmt seine Instanz aus einem Pool.
- Thread: Jeder Thread bekommt eine neue Komponenten Instanz vom Container zugewiesen.

Der *Keel* Server kann unter einer JVM (Java Virtual Machine) laufen. Es besteht aber auch die Möglichkeit ihn verteilt auf mehreren JVM einzusetzten.

4.3 Entwurfsentscheidung

Das Framework für den Publikationsprozess wird entsprechend den Anforderungen an den Publikationsprozess in digitalen Bibliotheken erstellt. Der Schwerpunkt wird auf die hohe Wiederverwendbarkeit gelegt. Dem Anwender soll eine Möglichkeit geboten werden, mit Hilfe des Frameworks einen Publikationsprozess für Multimediadokumente in eine digitale Bibliothek zu integrieren. Im Folgenden werden wichtige Entwurfsentscheidungen vorgestellt.

Durch einen modularen Aufbau soll eine hohe Wiederverwendbarkeit des Frameworks erreicht werden. Entsprechend den in diesem Kapitel gegebenen Überblick über Entwurfsmuster und Meta-Frameworks, wird durch die Definition von abstrakten Klassen und Interfaces die Wiederverwendbarkeit des Frameworks ermöglicht. Die Anwendung des Prinzips der Objektkomposition soll dies weiter unterstützen. Mit Hilfe des Prinzips der Delegation soll versucht werden, die dynamische Zusammensetzung von Verhalten zu unterstützen. Insgesamt steht im Vordergrund, Module zu erstellen, die möglichst einfach austauschbar sind.

Im weiteren Verlauf der Arbeit, werden die in Abschnitt 4.2.3 und 4.2.2 vorgestellten Frameworks *x18p* und *Spring* verwendet. Dadurch soll die die Entwicklung allgemeiner Komponenten unterstützt werden. Wie oben bereits erwähnt, ist eine Komponente durch ihr Interface und durch ihre Implementierung gegeben. Jede Komponente soll durch das Prinzip Inversion of Control (IoC) einfach zu verwenden sein. Ein weiterer Vorteil liegt darin, dass Module austauschbar sind, ohne Seiteneffekte hervorzurufen. Mit Hilfe von Service-Locatorn können Objekt Implementierungen gesucht werden. Zusätzlich übernimmt *Spring* die Kontrolle darüber, ob ein Objekt als Singleton erzeugt wird, oder nicht. Dadurch können Anwendungen erstellt werden, die Services verwenden ohne deren Implementierung zu kennen.
Durch eine klare Trennung von Geschäftslogik und Präsentationslogik ermöglicht das *x18p* Framework einen hohen Grad an Modularität. Damit werden Abhängigkeiten innerhalb einer Applikation vermieden. Durch die Integration des Meta-Frameworks *Spring* wird zusätzlich eine umfangreiche Lösung für die Erstellung von modularen Anwendungen geboten. Im Gegensatz zum *Keel* Framework, welches ähnliche Ansätze wie das *Spring* Framework hat, konnte *Spring* durch eine hohe Stabilität, eine sehr umfangreiche Dokumentation und eine einfach zu konfigurierende und installierende Anwendung überzeugen.

Mit Hilfe der Frameworks *Spring* und *x18p* und den oben genannten Konzepten der Objektkomposition und Delegation soll ein Framework für den Publikationsprozess erstellt werden, das aus einer Menge von Service-Modulen besteht, deren Verwendung durch einen Service-Requester und entsprechende Service-Controller gesteuert werden. Die Module und deren Komponenten sollen gegen Interfaces erstellt werden, damit es leicht möglich ist, sie auszutauschen.

5 Konzeption des Frameworks für den Publikationsprozess

Die zentrale Aufgabe dieser Diplomarbeit ist die Erstellung eines Publikationsprozesses, speziell für multimediale Dokumente in digitalen Bibliotheken. Dieser Prozess soll anschließend in die digitale Bibliothek *MyCoRe* der Universitätsbibliothek Rostock integriert werden. In Kapitel 2 werden allgemeine Anforderungen an Multimedia Dokumentmodelle und konkrete Beispiele von Modellen diskutiert. Der Publikationsprozess für Multimediadokumente innerhalb digitaler Bibliotheken wird in Kapitel 3 untersucht. Fragestellungen aus der Softwaretechnik, wie die Verwendung von Entwurfsmustern und Frameworks werden in Kapitel 4 vorgestellt. Aufbauend auf dem Publikationsprozess und den vorgestellten Meta-Frameworks, wird in einem weiteren Schritt ein Framework für den Publikationsprozess entworfen.

Die Erstellung eines Frameworks ist von Interesse, da die Wiederverwendbarkeit des erstellten Codes ein wichtige Rolle spielt. Der Publikationsprozess soll schnell und einfach an neue Anforderungen angepasst werden können. Der Aufwand beim Programmieren kann dadurch verringert werden. Das Framework soll, wie in Abschnitt 4.3 beschrieben, Vorteile bei der Modellierung und Wiederverwendbarkeit bringen.

Das Framework für den Publikationsprozess soll interne Datenobjekte nicht in einem eigenen Daten-Store verwalten. Hier wird eine Schnittstelle benötigt, die eine allgemeine Möglichkeit bietet, Datenobjekte aus einem in der Anwendung vorhandenen Repository zu laden oder persistent abzuspeichern. Das Framework bietet auch keine eigene Ausgabefunktionalität an, da dies ebenfalls Teil der Anwendung ist. Es werden keine Schnittstellen zur Verfügung gestellt. Durch die Verwendung des Meta-Frameworks Spring, bietet das Framework aber grundsätzlich Möglichkeiten für die Anbindung von vorhandenen Frameworks wie z.B. Struts, Hibernate oder Cocoon.
Alle beschriebenen Systeme und Komponenten sind mit der Programmiersprache Java umgesetzt.

5.1 Architektur

In Anlehnung an den in Abschnitt 4.2.3 vorgestellten Ansatz, wird das Framework grundlegend in einen **Service-Requester**, **Service-Router**, **Service-Controller** und **Service-Module** unterteilt. Das Konzept ist in Abbildung 5.1 dargestellt.
Die grundlegende Funktionsweise der Architektur wurde bereits in Abschnitt 4.3 vorgestellt. Im Folgenden werden die spezielle Umsetzung und eigene Erweiterungen, bezogen auf den Publikationsprozess, beschrieben.

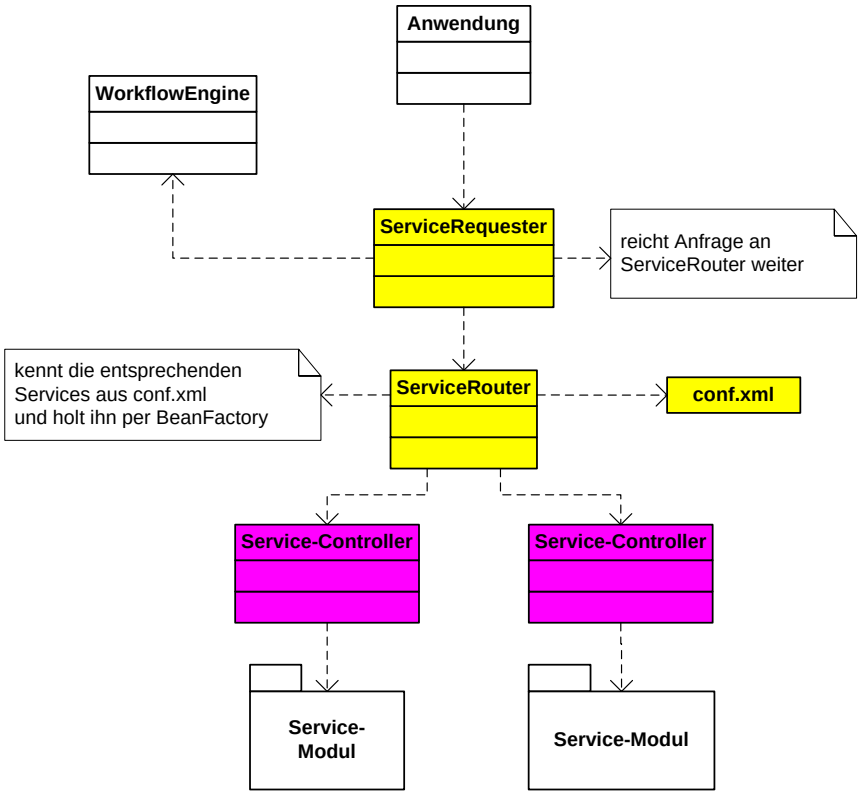

Abbildung 5.1: Architektur des Frameworks

Der Service-Requester

Ein zentraler Teil des Frameworks ist der Service-Requester. Anfragen von Java Servlets oder anderen Anwendungen werden an diesen Service-Requester gestellt. Entsprechend des *x18p* Frameworks wird ein ServiceRequest-Object zurück gegeben. Als Erweiterung des *x18p* Frameworks, wird der Service-Requester zusätzlich mit einer Workflow Anbindung ausgestattet. Damit wird es möglich, dass die Anwendung in Abhängigkeit eines Workflows ein bestimmtes ServiceRequest-Objekt geliefert bekommt. Im diesem Fall wird ein Teil der Information im voraus vom Workflow bereitgestellt. Mit Hilfe des in Abschnitt 4.2.2 vorgestellten Service-Locators und einer XML Datei, in der die Bean Definitionen eingetragen sind, wird ein entsprechendes Objekt erzeugt.

Das ServiceRequest-Objekt

Das ServiceRequest-Objekt dient zum Transport der Daten und der Aufgabenbeschreibung. Es hält Daten über

- den angeforderten Service
- die Operation, die der Service ausführen soll
- die Daten, die durch den Service verarbeitet werden sollen

Zusätzlich können auch Konfigurationsdateien an das ServiceRequest-Objekt übergeben werden. Nachdem das Objekt von der Anwendung mit Daten gefüllt wurde, wird dessen Methode `service()` aufgerufen. Damit sendet sich das Objekt selbständig an den Service-Router. Das Listing 5.1 zeigt einen Ausschnitt aus einem Service-Request-Objekt.

Listing 5.1: Auschnitt Service-Request

```
    public class ServiceRequest {
    public Object service() throws Exception {
         return ((ServiceRouter) this.serviceContext.
         getBean("serviceRouter")).route(this);
     }
     public String getServiceName() {
     ...
     }
     public void setServiceInputs(List list) {
     ...
     }
     public void addRequestInput(String key, Object value) {
     ...
     }
     ...
}
```

Der Service-Router

Der Service-Router erhält entsprechend den Vorgaben, ein solches ServiceRequest-Objekt und wertet die darin enthaltenden Informationen aus. Anhand des angegebenen Service wählt der Router einen entsprechenden Service-Controller aus. Anders als im *x18p* Framework vorgeschlagen, ruft der Service-Router den Spring Service-Controller direkt auf. Das Einführen einer weiteren Abstraktion, wie gefordert, ist hier nicht notwendig. Mögliche Änderungen können direkt in der Bean Definitions Datei vorgenommen werden.

Die Service-Controller

Die Verarbeitung der ServiceRequest-Objekte wird vom Service-Controller gesteuert. Sie kontrollieren die Abarbeitung der angegebenen Operationen. Dafür stehen Service-Module zur Verfügung.

Die Service-Module

Komponenten werden zu Service-Modulen zusammengefasst. Sie bieten den eigentlichen Service an. Sie sind nach ihrer Funktionalität in Service-Module eingeordnet, welche von entsprechenden Service-Controllern verwendet werden. Die Auswahl der Komponenten erfolgt ebenfalls durch den bereits vorgestellten Service-Locator.

5.2 Module und Komponenten

Das Framework für den Publikationsprozess baut auf den in Abschnitt 3.3.2 vorgestellten Stufen des Publikationsprozesses auf.

Wie bereits im vorherigen Abschnitt erwähnt, werden Service-Module in Komponenten unterteilt. Jede Komponente erfüllt eine ganz spezielle Aufgabe innerhalb des Moduls. Ein Modul stellt dabei einen Container für Komponenten dar. Alle Komponenten müssen dem Service Prinzip des *x18p* Frameworks aus Abschnitt 4.2.3 genügen. Das bedeutet, jede Komponente bietet genau einen speziellen Service an.

Um die Modularität und damit auch eine leichte Austauschbarkeit zu unterstützen, müssen für die einzelnen Service-Module und Komponenten Java Interfaces und abstrakte Klassen definiert werden. Ein Java Interface ist eine spezielle Klasse, die nur abstrakte Methoden und Konstanten definiert. Durch die Bereitstellung solcher Interfaces kann die Schnittstelle zur Kommunikation zwischen den Modulen und den Komponenten definiert werden, ohne auf spezielle Implementierungen einzugehen.

Die Service-Module werden entsprechend der Einteilung in Abschnitt 3.3.2 erstellt. Das Framework aus Abbildung 5.1 wird dabei um die jeweiligen Module erweitert. In Abbildung 5.2 wird die Erweiterung gezeigt.

5.2.1 Service-Controller

Komponenten in Modulen müssen durch einen Service-Controller gesteuert und verwaltet werden. Der Service-Controller entscheidet anhand der Operation, die im ServiceRequest-Objekt definiert ist, welche Komponente wie aufgerufen wird.

Die abstrakte Java Klasse `BaseController` definiert alle Methoden, die ein Service-Controller implementieren muss. Dazu gehören:

- afterPropertiesSet: Das *Spring* Framework ruft diese Methode automatisch auf, nachdem ein Objekt instanziiert wurde. Die Methode setzt eine bean factory und ein Workflow Objekt.
- prozess: Die abstrakte Methode prozess wird verwendet, um einen Service-Controller zu benutzten. Als Parameter wird ein ServiceRequest-Objekt übergeben.

Zusätzlich kann ein Workflow-Objekt gehalten werden, dass dem Controller die Kommunikation mit einem WFMS ermöglicht.

5.2.2 Asset Controller und Modul

Das Erfassen von Content ist eine wichtige Aufgabe im Publikationsprozess. Die Anforderungen daran wurden in Abschnitt 3.3.2 vorgestellt. Für die Umsetzung müssen der Service-Controller, der *AssetController* und das Service-Modul *AssetCreationModul* im Framework umgesetzt werden.

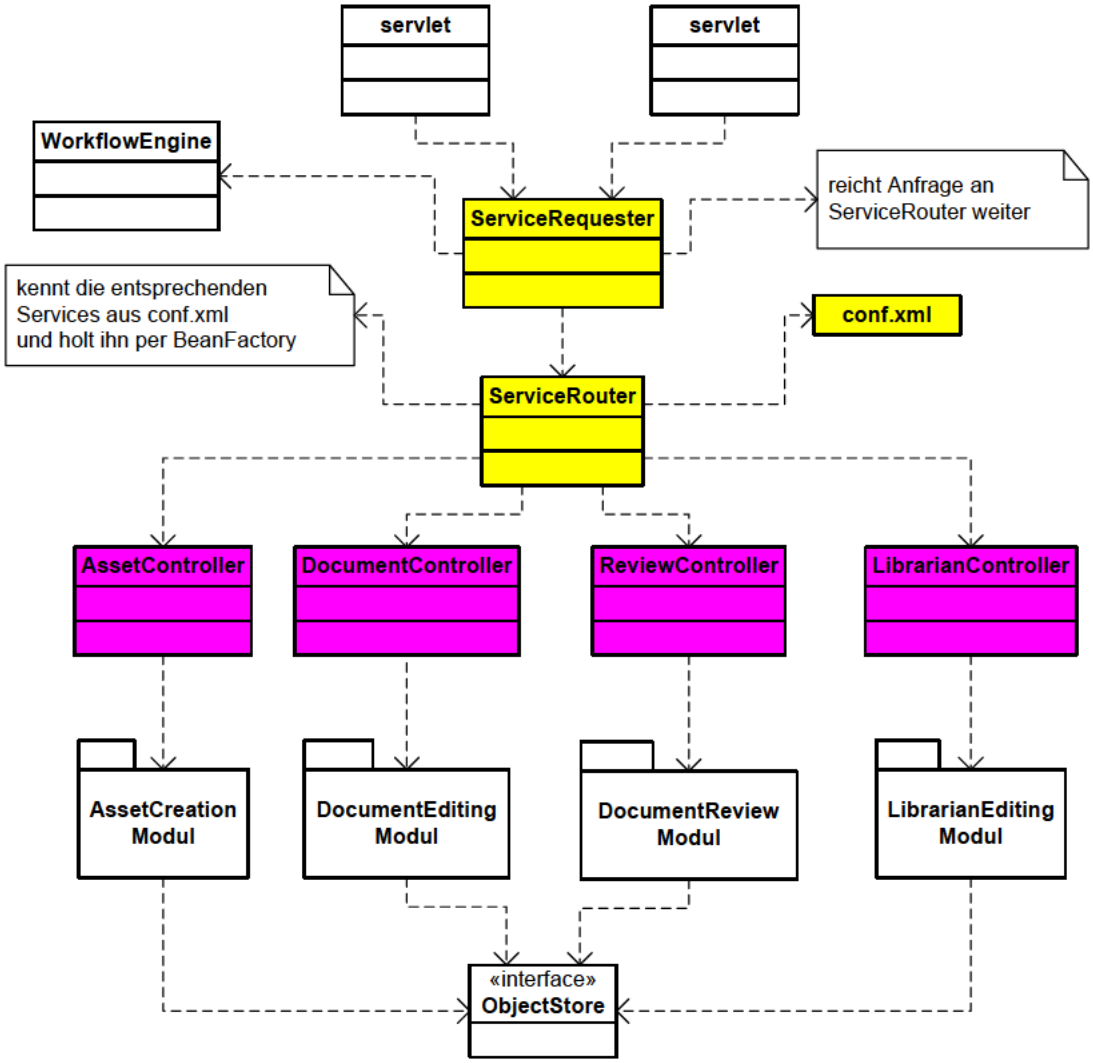

Abbildung 5.2: Framework für den Publikationsprozess in digitalen Bibliotheken

Der AssetController muss die Funktionalität bieten, um das AssetCreationModul zu steuern. Das Modul fasst entsprechend Abschnitt 3.3.2 die folgenden Komponenten zusammen:

- Datei-Upload
- Datei Konvertierung
- Aggregierung

In Abbildung 5.3 wird das UML Diagramm für das Modul *AssetCreationModul* dargestellt.

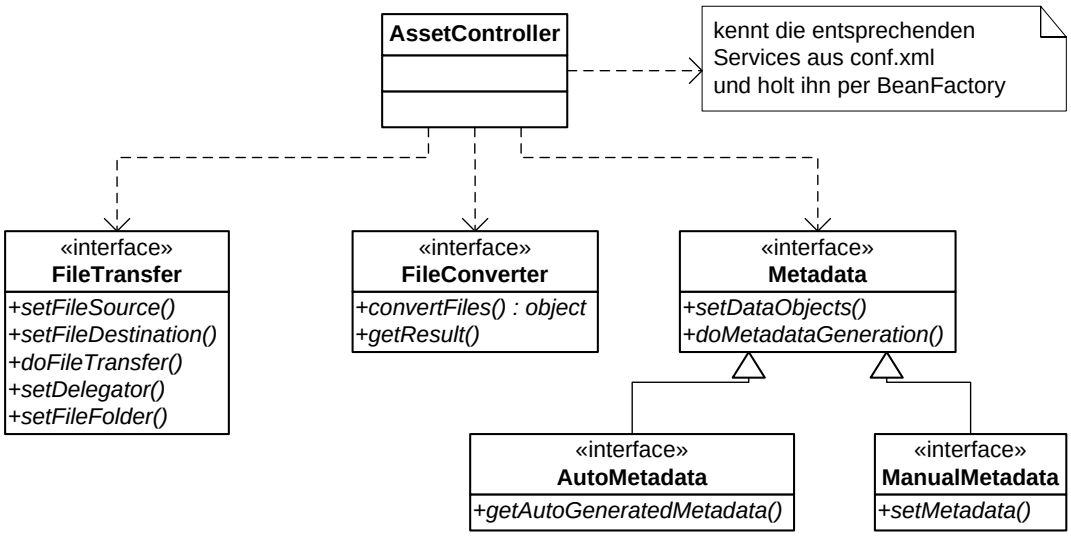

Abbildung 5.3: Service-Modul AssetCreationModul

Datei-Upload

Alle Komponenten, die einen Datei-Upload Service anbieten, müssen das Java Interface FileTransfer implementieren.
Es soll ermöglichen, Dateien aus verschiedenen Quellen in das System zu laden. Dazu zählen z.B. der Upload via HTML Formular oder FTP. Anwendungen, die durch eine Erweiterung Daten direkt in das System laden sollen, können diese Schnittstelle ebenfalls nutzen.

Datei Konvertierung

Alle Komponenten, die einen Konvertierer anbieten müssen das Interface FileConverter implementieren. Ein Konvertierer muss Objekte vom Typ `java.io.File` verarbeiten können. Diese Einschränkung folgt aus der Forderung, dass alle mit dem System zu verarbeitenden Dateien auf einem Dateisystem zwischengespeichert werden müssen (siehe Abschnitt 3.3.2). Die zu konvertierenden Dateien können vom Nutzer oder AssetCreationModul verwaltet werden.

Wie im Content-Life-Cycle für den Publikationsprozess bereits erwähnt, wird in diesem Schritt ebenfalls die Segmentierung der Dateien erfolgen.

Aggregierung

Die Aggregierung wird im Framework durch die Vergabe von Metadaten repräsentiert. In diesem Fall wird keine Segmentierung der Daten mehr vorgenommen. Dieser Schritt ist Bestandteil der Konvertierung.

Metadaten können automatisch oder manuell erzeugt werden (siehe Abschnitt 3.3).
Komponenten für die manuelle Erstellung von Metadatensätzen müssen das Interface ManualMetadata implementieren. Eine solche Komponente soll zu einer Menge von Da-

tenobjekten und einer Menge von vorher festgelegten Metadaten einen Systemabhängigen Metadatensatz erzeugen. Dieser Weg soll auch verwendet werden, wenn Metadaten bereits durch eine andere Anwendung erzeugt wurden und wie z.B. in Abschnitt 3.3.2 beschrieben, im Content enthalten sind, oder die Daten in einer extra Form vorliegen.

Zusätzlich soll die Möglichkeit bestehen, Metadaten zu den übergebenen Dateien automatisch zu erstellen. Eine Komponente die Metadaten automatisch erzeugt, muss das Interface AutoMetadata implementieren. Eine Feature-Extraktion kann zusätzlich die Überprüfung der erzeugten Daten erfordern. Die Übergabe der Dateien kann hier anwendungsspezifisch umgesetzt werden. Von der Anwendung muss ebenfalls festgelegt werden, wie die erzeugten Metadatensätze verwaltet werden.

5.2.3 DocumentController und Modul

Der Service-Controller *DocumentController* und das Service-Modul *DocumentEditingModul* fassen die Funktionalität für das Erstellen von Multimediadokumenten zusammen. Dokumente werden aus einer Menge anderer Dokumente zusammengesetzt.
Verwendet werden können dafür nur Dokumente, die bereits in einem Repository enthalten sind. Es sei hier nochmals darauf hingewiesen, dass alle Dokumente, die durch die vorherigen Schritte erzeugt wurden, im Repository gespeichert wurden.

Die einzelnen Dokumente werden in Beziehung zueinander gesetzt. Der DocumentController muss das Erstellen von verschiedenen Beziehungstypen unterstützen. Eine detaillierte Übersicht über wichtige Typen von Multimediadokumenten wird Abschnitt 2.2 beschrieben.

In Abbildung 5.4 wird das UML Diagramm für das Service-Modul *DocumentEditingModul* gezeigt.

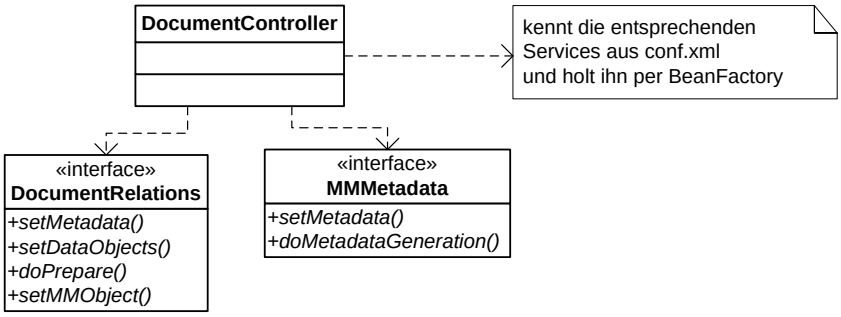

Abbildung 5.4: Service-Modul DocumentEditingModul

Jede Komponente, die eine bestimmte Beziehung modelliert, muss das Interface DocumentRelations implementieren. Für die Modellierung müssen die entsprechenden Datenobjekte und Metadaten angegeben werden. Die Metadaten sind beschreibende Daten über die Beziehung.

Zusätzlich zu den Beziehungen benötigt das Multimediadokument Metadaten. Um verschieden Metadaten anzugeben, wird das Interface MMMetadata eingeführt, welches durch entsprechende Komponenten zu implementieren ist.

5.2.4 Review Controller und Modul

Der Service-Controller *ReviewController* und das Service-Modul *DocumentReviewModul* fassen die Funktionalität für das Kontrollieren von erzeugten Dokumenten zusammen. Hilfreich kann die Anbindung an ein WFMS sein. Das WFMS kann Aufgaben verwalten und verteilen. Dadurch kann z.B. gesteuert werden, von wem und wie ein Dokument geprüft werden muss. Eine Komponente, die einen Service zum Testen von Dokumenten anbietet, muss das Interface DocumentChecker implementieren. Wie in Abschnitt 3.3.2 beschrieben, können die Dokumente getestet kontrolliert und auf bestimmte Anforderungen hin überprüft werden. Der Nutzer muss die Metadaten kontrollieren und gegebenenfalls ändern können. Dazu wird das Interface ManualMetadata aus Abschnitt 5.2.2 übernommen. In Abbildung 5.5 wird das UML Diagramm für das Service-Modul *DocumentReviewModul* gezeigt.

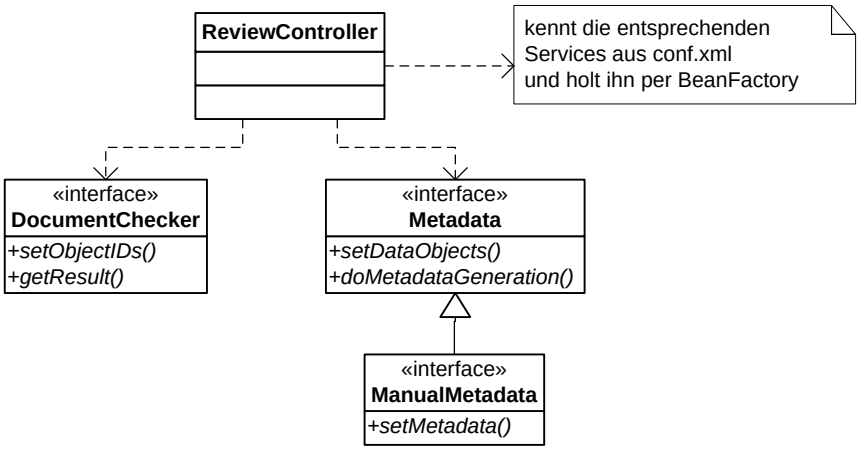

Abbildung 5.5: Service-Modul DocumentReviewModul

5.2.5 Librarian Controller und Modul

Der Service-Controller *LibrarianController* und das Service-Modul *LibrarianEditingModul* fassen die Funktionalität für das bibliothekarische Evaluieren von erzeugten Dokumenten zusammen. Hier können durch die Bibliotheksmitarbeiter erweiterte Metadaten wie z.B. Klassifikationen angegeben werden. Dazu müssen die Interfaces ManualMetadata und AutoMetadata implementiert werden, welche bereits in Abschnitt 5.2.2 über Servie-Controller AssetController vorgestellt wurden. In Abbildung 5.6 wird das UML Diagramm für das Service-Modul *LibrarianEditingModul* gezeigt.

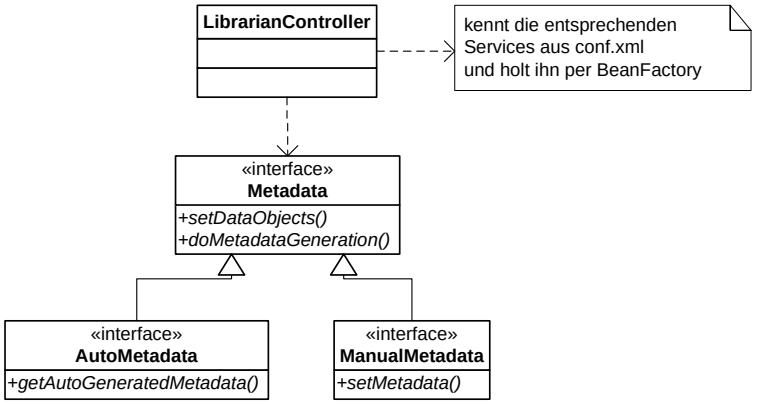

Abbildung 5.6: Service-Modul LibrarianEditing

5.2.6 Object Store Modul

Um Datenobjekte in ein Repository zu speichern oder daraus zu laden, wird eine allgemeine Schnittstelle angegeben. Das Interface *ObjectStore* soll von allen Komponenten implementiert werden, die Zugriff auf ein Repository bieten. An dieser Stelle wird kein eigener Service-Controller eingeführt, da das Repository nur von internen Komponenten und Modulen direkt verwendet werden sollte. Die Anwendung muss Objekte immer über eine entsprechende Komponente anfordern (siehe Abbildung 5.7).

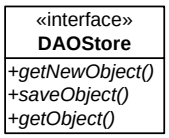

Abbildung 5.7: Service-Modul ObjectStore

5.3 Workflowanbindung für das Publikationsframework

Der Content-Life-Cycle für Multimediadokumente soll im Publikationsprozess einer digitalen Bibliotheken zusätzlich durch einen Workflow gesteuert werden. Für die Verwendung von Workflows wird eine Schnittstelle für ein Workflow-Management-System benötigt (siehe Abbildung 5.2).

Um Workflow Funktionalität in das Framework zu integrieren, wird das WorkflowInterface eingeführt. In [Sch04] ist die Einführung einer Workflow-Komponente für digitale Bibliotheken detailliert beschrieben. Das Interface bietet die Möglichkeit, Workflows

anzulegen und zu bearbeiten. Es können nutzerabhängige Verbindungen zum Workflow-Management-System erstellt werden. In welchem Umfang eine Workflow Unterstützung genutzt wird, ist sehr stark von den spezifischen Anforderungen einer Anwendung abhängig (siehe 3.2.2).

Die Anbindung eines WFMS an das Framework wird in Abbildung 5.8 dargestellt.

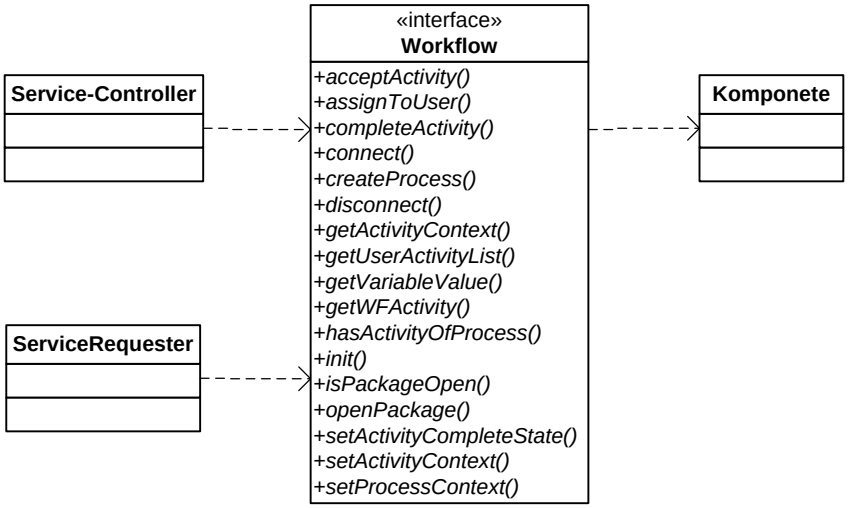

Abbildung 5.8: Workflow Anbindung an das Framework

Da die Anforderungen einer Anwendung schwer vorher zu sagen sind, gestaltet sich die allgemeine Anbindung eines Workflow-Management-Systems schwierig. Komponenten dürfen keinen direkten Zugriff auf das Workflow-Management-System (WFMS) erhalten. Der hier gewählte Ansatz geht davon aus, dass alle Anfragen an das WFMS vom

- Service-Requester
- oder einem Service-Controller

ausgehen.

Der Service-Requester kann den aktuellen Status des Workflow abfragen. Anhand des Status, kann entschieden werden, welches ServiceRequest-Objekt vom Service-Requester ausgeliefert wird. Informationen, wie Service-Name und Operationen, können ebenfalls durch den Workflow vorgegeben werden. Durch Verwendung eines WFMS, stellt eine Anwendung sicher, dass der Content-Life-Cycle im Publikationsprozess immer entsprechend eines vorgegebenen Workflows abläuft.

Neben dem Service-Requester können alle Service-Controller die Schnittstelle für das WFMS verwenden. Komponenten müssen die Ergebnisse ihrer Arbeit an den entsprechenden Service-Controller weiter leiten. Dies können einfache Statusmeldungen sein,

aber auch spezielle Daten. Ein Controller muss die Informationen seiner Komponenten, entsprechend des verwendeten Workflows, an das WFMS weiterreichen, damit der Workflow aktualisiert werden kann. Weiterhin können alle Daten, die ein Controller vom Workflow benötigt, abgefragt werden.

Wenn eine Anwendung es erfordert, kann das WFMS bestimmte Komponenten selbständig ausführen. Dies steht nicht unter Kontrolle des Publikationsframeworks.

6 Implementierung für die digitale Bibliothek Rostock

In diesem Kapitel wird eine prototypische Implementierung des in Kapitel 5 vorgestellten Frameworks für den Publikationsprozess in digitalen Bibliotheken gezeigt. Der Publikationsprozess wurde in die digitalen Bibliothek der Universitätsbibliothek Rostock `@libri` integriert. Die Umsetzung erfolgt beispielhaft anhand eines Multimedia Dokumententyps für Bildserien. Das digitale Bibliothekssystem basiert auf dem Content-Repository-System MyCoRe. Im Folgenden wird *@libri* als Synonym für das digitale Bibliothekssystem verwendet. Als Workflow-Management-System (WFMS) kommt die Open-Source Software *Enhydra Shark* zum Einsatz. Die Lösungen sind in der Programmiersprache Java implementiert. Die SQL Datenbank MySQL wird in *@libri* als Datenbank Backend verwendet. Der Java Servlet Container Tomcat in der Version 5.0 kommt als Web Server zum Einsatz.

Die Integration des WFMS Shark in *MyCoRe* wird in [Sch04] beschrieben.

6.1 Integration einer Bildserie

Die Grundvoraussetzung für die Implementierung einer Bildserie ist ein installiertes *MyCoRe* Framework. In dieser Arbeit wurde die *MyCoRe* [1] Version 1.2 verwendet. Für die Workflow Unterstützung wurde das oben genannte Workflow-Management-System (WFMS) *Enhydra Shark* [2] in der Version 1.1-2 genutzt. Die Workflow Beschreibungen wurden mit dem Tool *Enhydra JaWE* [3] in der Version 1.4.2 erstellt. Die Installation der Systeme wurde entsprechend der zur Verfügung stehenden Anleitungen durchgeführt.

Für die prototypische Implementation des Publikationsprozesses wurde ein Multimedia Dokumentenmodell für Bildserien erstellt. Das Modell beschreibt einen Container für Bilder. Eine Bildserie kann Metadaten und n Bilder enthalten. Diese Bilder müssen vom Datentyp Bild sein, der ebenfalls definiert wird. Der Nutzer hat die Möglichkeit, Bilder die Bestandteil einer Bildserie sind, auszuwählen um zu den beschreibenden Daten dieses Einzelbildes zu gelangen.

Als Ausgangspunkt für eine neue Bildserie dient die Bildverwaltungssoftware Picasa[4] von Google. Sie bietet die Möglichkeit eine Auswahl von Bildern in eine Ordnerstruktur

[1] `www.mycore.de`
[2] `http://shark.objectweb.org/`
[3] `http://jawe.objectweb.org/`
[4] `http://picasa.google.com`

mit beschreibender XML Datei zu exportieren. Der Publikationsprozess soll den Autor beim Import dieser Dateien in *@libri* unterstützen. Entsprechend den Anforderungen aus den vorherigen Kapiteln, kann der Autor z.B. Metadaten vergeben und ein neues Multimediadokument für Bildserien mit den importierten Bildern erstellen.

Um diese Funktionalität zu gewährleisten, musste *@libri* mit zwei neuen Datenmodellen erweitert werden. Ein Datenmodell für Bilder
(`datamodel-mmimages.xml`) und ein Datenmodell für die Bildserien
(`datamodel-mmimageseries.xml`), welches eine Art Container für Bilder darstellt. Die XML Dateien werden von *MyCoRe* automatisch in eine XML Schema Datei umgewandelt. Für die neuen Datenmodelle mussten neue Editor Masken erstellt werden, um die Dokumente der neuen Datentypen editieren zu können. Als Nutzerschnittstelle wurden neue Java Servlets eingeführt.

Das Framework für den Publikationsprozess wurde in dieser Arbeit in Form von Java Dateien in *@libri* integriert. Zukünftig besteht auch die Möglichkeit, die Daten als Java jar Archiv zu verteilen. Die vom Framework zur Verfügung gestellten Java Interfaces wurden entsprechend implementiert. Dazu gehören z.B. eine Komponente für den Datei-Upload oder ein File-Converter. Die Funktionalität des Frameworks wird über Java Servlets in *@libri* eingebunden.

6.2 Das Spring Framework

Das Spring Framework wird mit der Datei `spring.jar` eingebunden. Spring wird in dieser Arbeit hauptsächlich für die Instanziierung, Konfiguration und Verwaltung von Objekten verwendet. Um ein Objekt mit Spring instanziieren zu können müssen mehrere Bedingungen erfüllt werden.

- Das Objekt muss eine Bean Definition in der Datei `bean.xml` besitzen.
- Das aufrufende Objekt muss eine bean factory erzeugen.
- Das Objekt muss alle erforderlichen Interfaces implementieren.

Eine Anleitung, wie Objekte eingebunden werden, ist unter
`http://www.springframework.org/documentation` zu finden.

Eine bean factory wird z.B. über den in Listing 6.1 gezeigten Java Code erzeugt.

Listing 6.1: bean factory

```
Resource res = new FileSystemResource("beans.xml");
XmlBeanFactory factory = new XmlBeanFactory(res);
```

Ein Beispiel für eine Bean Definition wird in Listing 6.2 beschrieben.

Listing 6.2: Bean Definition

```
<bean id="exampleBean" class="examples.ExampleBean"/>
```

Das Code Beispiel in Listing 6.3 zeigt, wie man mit Hilfe einer XMLBeanFactory und einer Bean ID ein neues Objekt erzeugen kann.

Listing 6.3: Objekt mit bean factory erzeugen

```
ExampleBean bean = (ExampleBean) factory.getBean("exampleBean");
```

In der oben beschriebenen Art und Weise, werden die meisten Objekte im Framework erzeugt.

6.3 Das *MyCoRe* Content Repository

MyCoRe ist ein Content Repository System für digitale Bibliotheken und Archivlösungen. Das Projekt ist ein Zusammenschluss verschiedener Universitäten, die *MyCoRe* als Open Source Projekt entwickeln. Das Ziel der Community besteht darin, einen Software-Kern bereit zu stellen, der es ermöglicht, eigene Anwendungen zu erstellen, die auf die jeweiligen Bedürfnisse der Nutzer angepasst sind. Grundlage für das System stellen Java-Klassenbibliotheken, XML-Techniken sowie verschiedene Datenbank-Backends dar. Die digitale Bibliothek der Universitätsbibliothek Rostock basiert auf dem *MyCoRe* Framework.

6.3.1 Überblick

Das *MyCoRe* System bietet die folgende Grundfunktionalität:

- ein frei wählbares Metadatenmodell
- ein eigenes internes Dateisystem (IFS)
- ein hierarchisches Klassifikationssystem
- eine eigenes Rechtesystem mit Benutzerverwaltung
- eine rudimentäre Workflow Unterstützung
- eine Volltextsuche auf Textdokumenten

Die Architektur von *MyCoRe* baut auf dem 3-Schichten Modell auf.

- Präsentationsschicht
- Geschäftslogik

- Persistenz- und Datenschicht

Die Architektur von *MyCoRe* ist in Abbildung 6.1 dargestellt. Es wird versucht, die Daten streng von ihrer Repräsentation zu trennen. Java Klassen, Datenmodelle und XML Dateien der Geschäftslogik werden von den Java Servlets verwendet. Die serverseitige Nutzerschnittstelle in *MyCoRe* wird durch Java Servlets implementiert. Um die Darstellung von der Verarbeitungslogik und den Daten zu trennen, erzeugen die Servlets ihre Ausgaben im XML Format. Die Ausgaben werden von XSL Stylesheets in entsprechende Darstellungen wie z.B. HTML transformiert. Die Zuweisung von XML Ausgaben zu einem Stylesheet wird durch das Layout-Servlet durchgeführt. Alle Layout Informationen werden in diesen Stylesheets verwaltet. Für weitere Informationen wird auf [LKDa][LKDb] verwiesen.

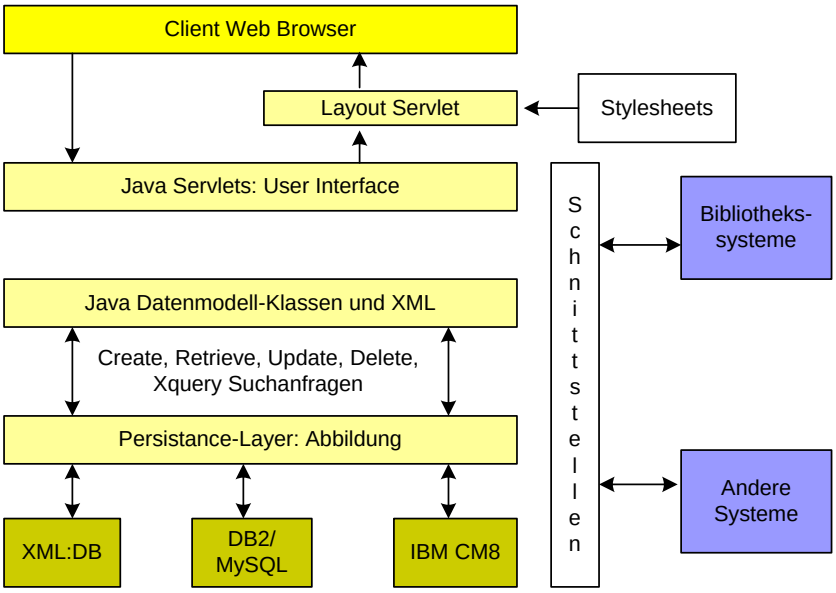

Abbildung 6.1: *MyCoRe* Architektur

6.3.2 *MyCoRe* Dokumentenmodell

Für die Verwaltung digitaler Objekte stellt *MyCoRe* ein eigenes Datenmodell zur Verfügung. Das Datenmodell besteht aus drei Modulen:

- Metadatenmodell
- Klassifikationssystem
- logisches Dateisystem

Das Metadatenmodell bietet ein frei konfigurierbares Modell für die Beschreibung von digitalen Objekten mit Metadaten. Dabei lassen sich neue Objekttypen definieren, für die jeweils ein eigenes Metadatenmodell erstellt werden kann.

Das Klassifikationssystem bietet die Möglichkeit, Objekte nach einer Menge von Kategorien zu klassifizieren. Dabei können eigene Klassifikationen eingeführt werden. Ein Dokument kann in *MyCoRe* mehreren Kategorien angehören. Mit einem Web-Interface kann der Nutzer über die Klassifikationen browsen.

Das logische Dateisystem verwaltet die digitalen Objekte (Derivate), die in *MyCoRe* gespeichert werden. Der Content wird in das System importiert und dort selbständig verwaltet. Dafür wird eine eigene API zur Verfügung gestellt, die die entsprechend benötigten Funktionen zur Verfügung stellt.

Das Datenmodell wird in Abbildung 6.2 dargestellt.

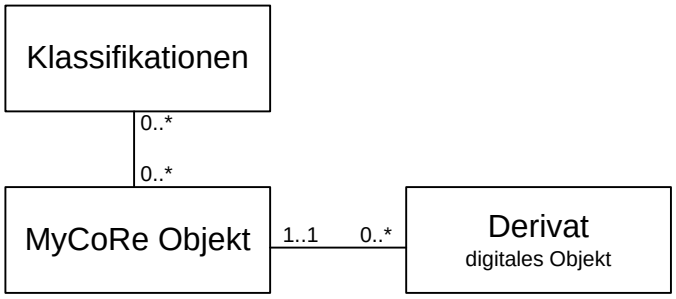

Abbildung 6.2: *MyCoRe* Datenmodell

Das Datenmodell kann erweitert werden, indem zum Kernsystem projektspezifische Java Klassen hinzugefügt werden. Damit ist es möglich, dass Modell mit Programmieraufwand zu erweitern.

Konzepte und Besonderheiten

In *MyCoRe* wird XML als Beschreibungsprache der einzelnen Modelle verwendet.

Ein Zentrales Element des *MyCoRe* Modells ist die *MCRObjectID*. Sie stellt eine eindeutige ID dar, die aus einem Projektkürzel, einem Dokumenten-Typ und einer Nummer besteht (**ID="projektkürzel_type_nummer"**).

Das Metadatenmodell beschreibt die Metadaten für digitale Objekte. Die digitalen Objekte werden im logischem Dateisystem abgelegt, auf das weiter unten noch eingegangen wird. Das Modell erlaubt es, neben beschreibenden Daten wie Titel und Autor, auch Struktur und Service Daten zu verwalten. Wie bereits erwähnt, werden dafür XML-Strukturen verwendet. Abbildung 6.3 zeigt die Struktur eines *MyCoRe* Metadaten Objektes, das aus einem Struktur-Teil, einem Metadaten-Teil und einem Service-Teil besteht. Mit dieser Aufteilung wird eine Trennung von Struktur-, Beschreibungs- und Servicedaten erreicht. Außerdem soll dadurch ermöglicht werden, dass die Metadaten auf unterschiedlichen Datenspeichern abgelegt werden können. Standarddatentypen und Grundstrukturen werden im *MyCoRe* Framework als XML Schema Dateien mitgeliefert.

```
<?xml version="1.0" encoding="UTF-8" ?>
<mycoreobject xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="....xsd"
      xmlns:xlink="http://www.w3.org/1999/xlink" ID="..." label="..." >
      <structure>
       ...
      </structure>
      <metadata xml:lang="de">
       ...
      </metadata>
      <service>
       ...
      </service>
</mycoreobject>
```

Abbildung 6.3: *MyCoRe* Metadatenmodell

Es wird zwischen Metadaten der digitalen Objekte und den digitalen Objekten (Derivate) unterschieden. Die Metadaten werden für jedes Objekt in drei Bereiche eingeteilt:

- structure
- metadata
- service

Im Folgenden werden die drei Abschnitte eines *MyCoRe* Objektes beschrieben.
Der Abschnitt **structure** in einem *MyCoRe* Objekt bietet die Möglichkeit, Objekthierarchien und Referenzen zu digitalen Objekten zu formulieren. Objekthierarchien werden mit den Tags *parent/parents* und *child/children* bezeichnet. Dabei können Elternbeziehungen vom Anwender bzw. der Anwendung explizit angegeben werden. Die Kinder eines Objektes werden hingegen durch das *MyCoRe* System automatisch erzeugt. Dies gilt ebenfalls für den *derivate* Tag. Dem Anwender steht es offen, neue Beziehungen einzuführen, was aber mit einem erhöhten Programmieraufwand verbunden sein wird.

Im Abschnitt **metadata** werden die beschreibenden Daten, wie z.B. bibliografische Angaben, unterstützt. *MyCoRe* stellt dafür eine Reihe von Standard Datentypen zur Verfügung, die in beliebiger Reihenfolge und Anzahl angeordnet werden können. Zu jedem Datentyp MCRMeta-Typ gehört eine entsprechende Java Klasse MCRMeta-Java im *MyCoRe* Framework. Für jedes Attribut kann einzeln angegeben werden, ob es suchbar sein soll und ob es vererbbar ist. Es wird aber auch die Möglichkeit geboten, eigene Datentypen zu erstellen. Standardmäßig wir der Standard Dublin Core (DC) unterstützt. In Tabelle 6.1 werden die Standard Datentypen aufgelistet. Eine genaue Beschreibung dieser Typen enthält der *MyCoRe UserGuide* [LKDb].

Im Knoten **service** können Daten für die Unterstützung von Workflows und Wartung abgespeichert werden. Dabei ist eine bestimmte Menge von Tags, wie z.B. *acceptdate* und *createdate*, fest vorgegeben. Die service Daten können beliebig erweitert werden.

Einfache Typen	Komplexe Typen
MCRMetaBoolean	MCRMetaAddress
MCRMetaClassification	MCRMetaInstitutionName
MCRMetaDate	MCRMetaPersonName
MCRMetaISBN	MCRMetaIFS
MCRMetaLangText	
MCRMetaLinkID	
MCRMetaNBN	
MCRMetaNumber	
MCRMetaXML	

Tabelle 6.1: *MyCoRe* Standard Datentypen

Mit Hilfe von Metadatenmodellen können verschiedene Objekttypen, wie z.B. document, person oder dissertation definiert werden. Eigene Datentypen können per XML Dateien und zugehörigen Java Klassen, die für die interne Verarbeitung benötigt werden, erstellt werden.

Die digitalen Objekte werden in *MyCoRe* Framework im **IFS** (Internal File System) abgelegt. In einer Metadatendatei sind alle Beziehungen der Objekte mit entsprechenden Metadaten gespeichert. Die digitalen Objekte werden in *MyCoRe* als so genannte *Derivate* gespeichert. Ein Objekt kann mehrere Derivate besitzen. In einem Derivat lassen sich zusätzlich mehrere Dateien ablegen. Dieses Prinzip ermöglicht es, z.B. eine Grafik in mehreren Formaten zu speichern.

Das Klassifikationen Datenmodell ermöglicht es, die Daten anhand von Kategorien zu klassifizieren. In *MyCoRe* können dafür verschiedene XML Dateien verwendet werden, die entsprechende Klassifikationen beinhalten. Eine Klassifikation kann mehrstufig sein, wobei jede Stufe eine Hierachieebene bildet. Die einzelnen Stufen werden durch Punkte (.) getrennt. Dadurch sind Anfragen wie "Uni.* "möglich. Ein Dokument kann beliebig vielen Kategorien zugeordnet werden. Dafür wird ein entsprechendes XML Tag in den *metadata* Teil eines *MyCoRe* Objektes eingefügt.

Bewertung

Das *MyCoRe* Modell ist speziell für den Einsatz in Content Repositories und digitalen Bibliotheken entwickelt worden. Das Metadatenmodell biete eine klare Trennung von strukturellen und beschreibenden Metadaten. Ein Nachteil besteht darin, dass im Strukturteil nur Hierarchien dargestellt werden. Es ist aber möglich, die Struktur Elemente zu erweitern. Dafür ist ein Mehraufwand bei der Erweiterung des Frameworks notwendig. Die beschreibenden Metadaten können ebenfalls durch eigene Datentypen erweitert werden. Im *service* Teil können Informationen für eine Workflow Unterstützung und administrative Zwecke abgelegt werden. Die Anforderungen an Multimedia Dokumentenmodelle sind insgesamt aber höher. Hier muss das *MyCoRe* Modell erheblich erweitert werden.

6.3.3 Datenmodelle, Editoren und Stylesheets

In Abschnitt 6.3.2 wurde ein Überblick über die Standard-Datentypen des *MyCoRe* Systems gegeben. Um den Anforderungen an Multimedia Dokumentenmodelle besser zu genügen, muss das System erweitert werden (siehe Abschnitt 2.2). Damit das Framework für den Publikationsprozess prototypisch umgesetzt werden kann, wurden Datenmodelle sowohl für Bilder als auch für Bildserien erstellt und in *MyCoRe* integriert. Das Datenmodell für Bildserien zeigt dabei exemplarisch die Anforderungen für eine interaktive Beziehung zwischen Bildern. Die vollständige Umsetzung der in Abschnitt 2.2 geforderten Anforderungen an Multimediadokumente wird in dieser Arbeit nicht vorgenommen.

Folgende Anforderungen wurden an das Datenmodell für Bildserien gestellt:

- Angabe von allen Dublin Core (DC) Elementen
- Eine Bildserie muss eine Menge von Referenzen auf Bild Objekte enthalten können.
- Für alle Bild Objekte soll ein Thumbnail angezeigt werden
- die Bilder sollen in einer extra Ansicht hintereinander anzusehen sein.

Das Datenmodell für Bilder enthält nur die DC Elemente.
Für die Umsetzung wurden die Datenmodelle `datamodel-mmimageseries.xml` und `datamodel-mmimage.xml` eingeführt.

Das Datenmodell *datamodel-mmimageseries.xml* besteht entsprechend den *MyCoRe* Vorgaben, aus den Bereichen *structure*, *metadata* und *service*.
Im Teil *structure* wird gegenüber den Standardvorgaben nichts verändert. Nach Angaben der *MyCoRe* Entwickler ist es noch nicht möglich, an dieser Stelle eigene Erweiterungen in Form von neuen Elementtypen einzubringen. Die naheliegende Lösung, Objektbeziehungen im Strukturteil des Datenmodells anzugeben, kann aufgrund dieser Einschränkungen nicht vorgenommen werden. Außerdem kann mit der parents/parent Beziehung, die im Standard Modell definiert ist, entgegen den Aussagen der *MyCoRe* Dokumentation und den mitgelieferten Datenmodellen, keine Mehrfachbeziehung zwischen einem Kindelement und Eltern dargestellt werden. Wenn dieser Schritt umgesetzt wird, können Bildobjekte eine Referenz auf alle Bildserien haben, denen sie angehören. Zusätzlich könnte jede Bildserie mit Hilfe der children/child Beziehung Referenzen auf Bilder haben. Wichtig ist, dass in *MyCoRe* children/child Beziehung nur vom System erstellt werden dürfen.

Der *metadata* Bereich beschreibt:

- die Dublin Core (DC) Elemente
- Referenzen auf Bild Objekte

Für die Implementierung der Referenzen auf Bild Objekte, wird der Datentypen MCRMetaLinkID verwendet. Er basiert auf dem W3C XLink Standard und wird als Referenztyp verwendet. Dafür gelten die folgenden Attribute:

- **xlink:href**: beschreibt eigentliche Referenz auf eine MCRObjectID
- **xlink:from**: MCRObjectID
- **xlink:to**: MCRObjectID

Das Element hat folgende Struktur:

```
<element name="immageLinks" minOccurs="0" maxOccurs="1" parasearch="true"
    textsearch="false">
    <mcrmetalinkid name="imageLink" class="MCRMetaLinkID" minOccurs="1"
        maxOccurs="unbounded"/>
</element>
```

Abbildung 6.4: *MyCoRe* MCRMetaLinkID

Der Tag *service* wird für interne Verwaltungsaufgaben verwendet.

Zusätzlich zum Datenmodell für Bildserien wurde das Datenmodell *datamodel-mmimage.xml* eingeführt. Es unterscheidet sich vom Bildserien Datenmodell nur darin, dass das Element `immageLinks` nicht enthalten ist.

Für die Datenmodelle wurden die Editoren eingeführt:

- `editor-author-mmimage.xml`
- `editor-author-mmimageseries.xml`

Die Editoren bieten die Möglichkeit, Metadaten über ein HTML Formular anzugeben. Jeder Editor erzeugt Metadaten für genau ein Datenmodell.

6.3.4 Implementierte Servlets

Für die Implementierung des Publikationsprozesses in *MyCoRe* , wurden die folgenden Servlets erstellt:

- `FileUploadServlet.java`
- `PubImageSeriesServlet.java`
- `PubMetadataServlet`

Das Servlet `FileUploadServlet.java` wird nach einem HTML File-Upload aufgerufen, der in der XML Datei `fileupload.xml` definiert ist. Es bietet dem Nutzer die Möglichkeit Dateien in *@libri* zu laden.

Das Servlet `PubMetadataServlet.java` wird von dem Editor `editor-author-mmimage.xml` aufgerufen. Ein Editor sendet seine Ausgabe als XML

JDOM Objekt an eine angegeben Adresse. Das Servlet verarbeitet als erstes die Editor Ausgabe. Dem Konzept von *MyCoRe* folgend, muss an dieser Stelle eine eindeutige Objekt ID vom Typ MCRObjectID erzeugt und der Editor Ausgabe hinzugefügt werden. Die Ausgabe muss ebenfalls durch eine *MyCoRe* Funktion in eine Form transformiert werden, die vom internen System weiter verwendet werden kann.

Das Servlet `PubImageSeriesServlet` wird von dem Editor `editor-author-mmimageseries.xml` aufgerufen. Es verarbeitet die Editor Ausgabe wie oben beschrieben.

Nachdem die internen Verarbeitungsschritte beendet sind, werden folgende Operationen von den Servlets ausgeführt:

- Ein Service-Requester wird instanziiert.
- Das ServiceRequest-Objekt wird mit dem angeforderten Service Namen, der auszuführenden Operation und Daten gefüllt und ausgeführt.

6.4 Allgemeine Service Klassen

Die Schnittstelle zwischen Geschäftslogik und Präsentationsschicht stellen allgemeine Service Klassen dar (siehe Abschnitt 5.1). Für das Framework wurden folgende Klassen implementiert.

ServiceRequest

Das ServiceRequest-Objekt ist aus dem *x18p* Framework übernommen und wird, wie in Abschnitt 5.1 beschrieben, verwendet. Es wird durch die Java Klasse `ServiceRequest` implementiert. Die Integration in *@libri* und die Erweiterung des Frameworks mit einem WFMS erfordern eine Anpassung des ServiceRequest-Objektes. Das Objekt wird mit den folgenden Funktionen erweitert:

- getWfOperation und setWfOperation: Es kann die aktuelle Workflow Aktivität für einen bestimmten Nutzer gesetzt bzw. abgefragt werden.
- getUserID und setUserID: Setzt den aktuellen Nutzer einer *@libri* Session. Dieser Nutzer wird verwendet, um Anfragen an das WFMS zu stellen. Die Service-Controller verwenden den Nutzer für die korrekte Zuordnung von Daten zu Nutzern.

Service-Requester

Der Service-Requester (siehe Abschnitt 5.1) ist durch die Java Klasse `ServieRequester` implementiert. Er liefert ein Objekt vom Typ ServiceRequest. Die Schnittstelle wird durch eine Funktion definiert:

- getService: Die Methode liefert ein Objekt vom Typ ServiceRequest.

Die Anbindung an das WFMS bietet das Interface WFService. Abhängig von der Anfrage, kann die aktuelle Workflow Aktivität für einen Nutzer mit dem ServiceRequest-Objekt übergeben werden. Der Nutzer muss dabei als Parameter übergeben werden. In diesem Fall ist es immer der Nutzer aus der aktiven *MyCoRe* Session.

Workflow Service

Ein WFMS wird über die weiter unten vorgestellte Klasse `ATLSharkWorkflow.java` eingebunden. Die Verbindung zwischen dem Framework und dem WFMS stellt die Klasse `WFService.java` dar. Die Klasse kapselt zusätzlich den Zugriff auf das WFMS. Die folgenden Methoden werden von ihr Implementiert:

- *getCurrentService* liefert die aktuelle Workflow Aktivität für einen Nutzer. Es wird ein Objekt vom Typ ATLSharkWorkflow erzeugt. Anhand von Parametern wird die Workflow Engine (WFE) konfiguriert:
- *initUserWF* initialisiert einen neuen Workflow für ein bestimmtes Workflow Package, wenn es noch nicht geladen wurde. Danach wird der entsprechende Workflow Prozess gestartet.
- *getWF* liefert ein Objekt vom Typ ATLSharkWorkflow. Das Objekt kann z.B. von einem Service-Controller verwendet werden, um Daten mit der WFE auszutauschen.

6.5 Controller, Module und Komponenten

Um die Funktionsweise des Frameworks für den Publikationsprozess in digitalen Bibliotheken zu zeigen, wurden die in Kapitel 5 vorgestellten Module und die Komponenten beispielhaft für das Multimedia Dokumentenmodell Bildserie umgesetzt. Das Framework ist in die Architektur von *MyCoRe* integriert worden. In Abbildung 6.5 wird diese Erweiterung des Frameworks in *MyCoRe* gezeigt.
Die Funktionalität der Module und Komponenten wird an einem Beispiel über das Erzeugen einer Bildserie erläutert. Für das Beispiel wird eine ZIP Datei mit Bildern erzeugt. Dazu wird die Bildverwaltungssoftware Picasa von Goolgle verwendet. Damit können Bilder in eine Verzeichnisstruktur der Form

- /images
- /thumbnails
- index.xml

erzeugt werden. Diese wird dann in eine ZIP Datei zusammengefasst.

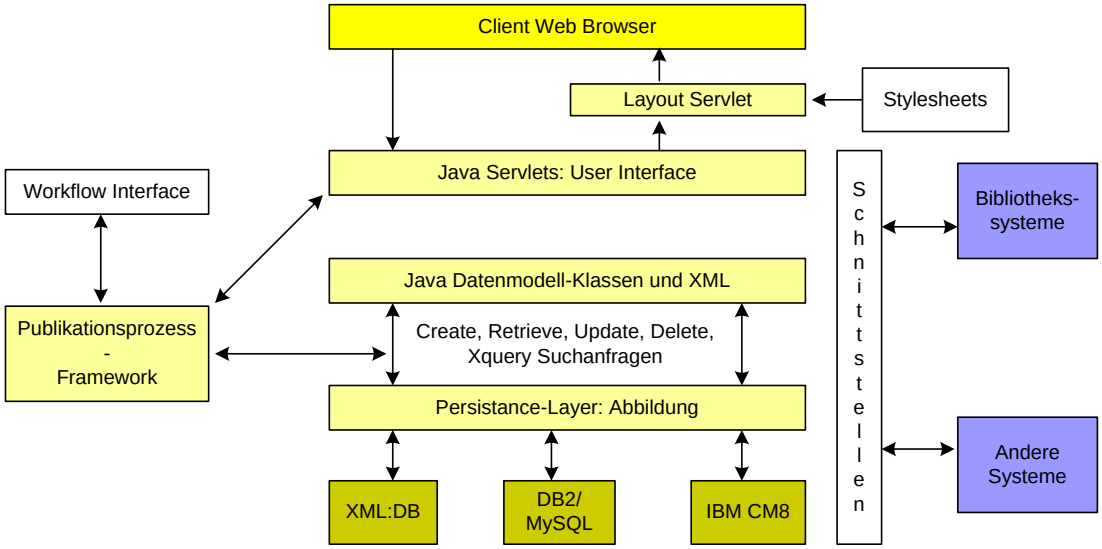

Abbildung 6.5: Erweiterung von *MyCoRe*

6.5.1 AssetController und AssetCreationModul

In diesem Abschnitt werden die implementierten Komponenten des *AssetCreationModul* vorgestellt.

Datei-Upload

Das Java Interface FileTransfer für den Datei-Upload, wird von der Klasse `FSFileTransfer.java` implementiert. Damit können Dateien im Dateisystem des Servers gespeichert werden.
In Abbildung 6.6 wird der allgemeine Ablauf eines Datei-Uploads gezeigt:

1. Das FileUploadServlet wird in diesem Fall von einem HTML Formular aufgerufen. Es erhält eine Liste von Dateien, die vom Nutzer hochgeladen wurden.
2. Das Servlet fordert vom Service-Requester eine ServiceRquest-Objekt an und setzt bei diesem alle wichtigen Informationen.
3. Das ServiceRequest-Objekt ruft den Service-Router auf, und sendet sich selbst an ihn.
4. Der Service-Router kann anhand des Service Namen im ServiceRequest-Objekt den AssetController instanziieren und das ServiceRequest-Objekt an den Controller weiterleiten.
5. Der AssetController wertet das ServiceRequest-Objekt aus, und instanziiert ein FileTransfer-Objekt vom Typ FSFileUpload. Er setzt alle benötigten Daten.
6. Das FileTransfer-Objekt speichert die übergebenen Daten auf dem angegebenen Bereich im Server.

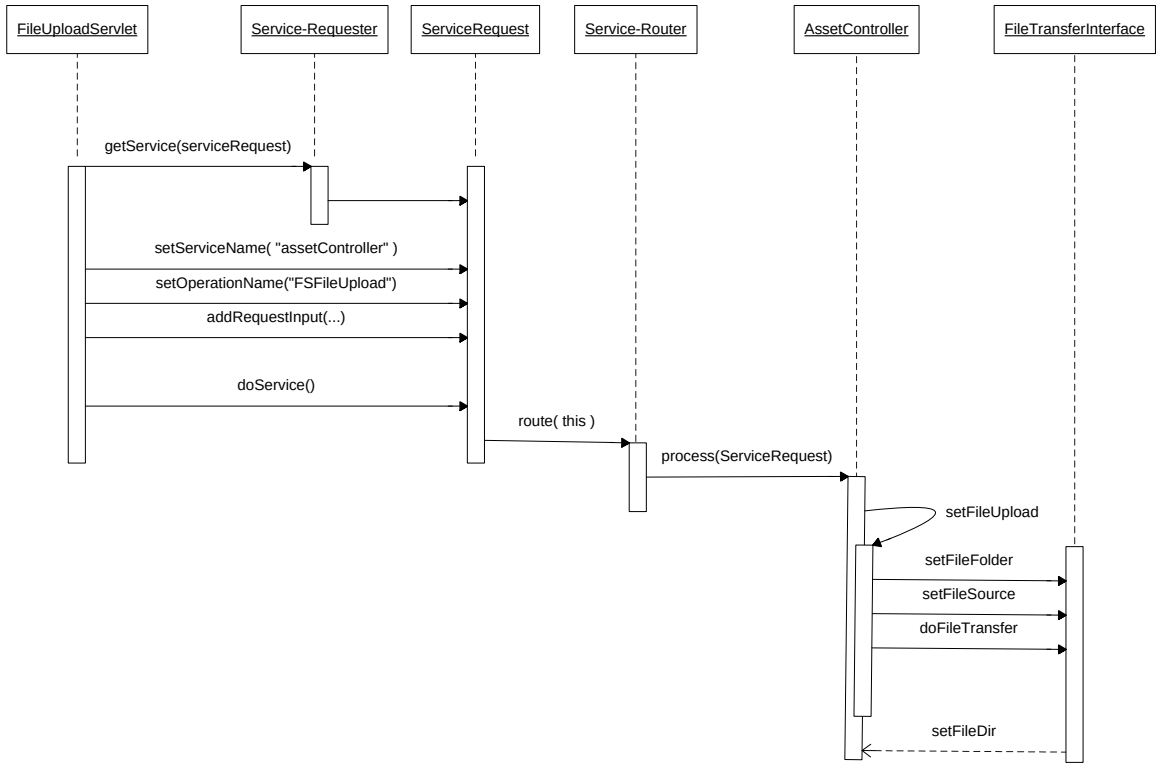

Abbildung 6.6: Sequenzdiagramm: Datei-Upload

Die folgende Liste erläutert, wie die Methoden des Interfaces FileTransfer umgesetzt wurden.

- *setFileSource* setzt ein globales Listen Objekt, welches Objekte vom Type `org.apache.commons.fileupload.FileItem` enthält.
- *setFileDestination* setzt einen String, der einen absoluten Pfad auf das Dateisystem des Servers hält.
- *setDelegator* setzt den AssetController, damit eine bidirektionale Verbindung zwischen AssetController und FSFileUpload möglich ist.
- *setFileFolder* setzt einen String, der das Unterverzeichnis im Dateisystem für den aktuellen Datei-Upload festlegt.
- *doFileTransfer* ruft als erstes eine Methode auf, die die ZIP Datei entpackt. Danach werden die Dateien an dem Ziel abgespeichert, welches durch die Variablen `destination` und `folder` angegeben wurden.

Datei Konvertierung

Das Interface FileConverter wird von der Datei `ImageFileConverter.java` implementiert. Der AssetController lädt mit Hilfe des Service-Locators alle Bean Elemente aus

der Datei `bean.xml`, deren ID auf den String *FileConverter* endet. Dieser Mechanismus ermöglicht, es alle File-Converter zur Laufzeit dynamisch in das System zu laden. Der AssetController durchläuft rekursiv den Verzeichnisbaum, der aus der ZIP Datei extrahiert wurde. Die Verzeichnisse werden für jeden gefundenen Konvertierer einmal durchlaufen. Der rekursive Durchlauf durch die Verzeichnisse wird mit Hilfe des *Visitor Pattern* im AssetController umgesetzt.

Die Klasse `ImageFileConverter.java` ermöglicht es, Bilder vom Typ *gif* oder *png* in *jpeg* Bilder zu konvertieren. Nach der Konvertierung werden die originalen Bilder in einen neuen Unterordner `/old` verschoben. Die Methoden des Interfaces sind folgendermaßen umgesetzt:

- *convertedFiles* konvertiert die übergebene Datei, wenn sie einen der beiden oben genannten Typen beinhaltet.
- *getResult* liefert `true`, wenn die Konvertierung erfolgreich war.

Aggregierung

Nachdem die Daten in einem einheitlichen Format vorliegen, müssen Metadaten vergeben werden. Das Interface ManualMetadata wird von der Klasse `MyCoReImageMetadata.java` implementiert. Die Klasse erzeugt aus einem übergebenen JDOM Objekt ein *MyCoRe* Metadaten Objekt vom Typ `MCRObject`. Das JDOM Objekt ist die Ausgabe des Editors `editor-author-mmimage.xml`. Aus den übergebenen Dateien werden Derivat Objekte erzeugt. Diese werden dem MCRObject hinzugefügt.

Die folgende Liste erläutert, wie die Methoden des Interfaces Metadata umgesetzt wurden.

- *setMetadata* setzt das globale Objekt `metaData`.
- *setDataObjects* setzt den globalen Vector `ObjectUris`, der die absolute Adresse der Dateien enthält.
- *doMetadataGeneration* erzeugt die eigentlichen Metadaten Objekte und Derivat Objekte. Zusätzlich werden die Objekte im Repository abgespeichert. Ab diesem Zeitpunkt sind sie suchbar.

An dieser Stelle werden keine weiteren Maßnahmen dafür ergriffen, dass die erzeugten Objekte nicht durch die Suchfunktion von *MyCoRe* erreichbar sind. Objekte werden ohne eventuelle Kontrolle freigegeben. Das Problem kann z.B. über entsprechende Service Flags im *service* Teil der Metadaten behoben werden. Die Umsetzung erfordert aber viele Änderungen in *MyCoRe* Stylesheets, was nicht Gegenstand dieser Arbeit ist.

Zusätzlich zu den inhaltsbeschreibenden Metadaten können inhaltsbezogene Metadaten automatisch erzeugt werden. Die Klasse ExifAutoMetadata implementiert das Interface AutoMetadata. Es werden automatisch alle Exif Informationen aus JPEG Bildern extrahiert und zurückgegeben.

- *getAutoGeneratedMetadata* gibt die generierten Metadaten zurück. Das Format ist anwendungsspezifisch.
- *setDataObjects* setzt den globalen Vector `Objects`, der eine Menge von MCRObjects enthält, für die eine automatische Metadaten Extraktion erfolgen soll.
- *doMetadataGeneration* extrahiert die Metadaten.

6.5.2 DocumentController und DocumentEditingModul

Nachdem alle Dateien aus der ZIP Datei in das System geladen wurden, und entsprechende Metadaten vergeben sind, kann das eigentliche Multimediadokument erstellt werden. Für das Dokument müssen ebenfalls Metadaten angeben werden. Dafür wird die Ausgabe des Editors `editor-author-mmimageseries.xml` verwendet. Die Ausgabe wird von dem Servlet `PubImageSeriesServlet` verarbeitet. Das Servlet fordert ein ServiceRequest-Objekt vom Service-Requester an und füllt dies mit der Editor Ausgabe und den zu verwendenden Bild Objekten, die in den vorherigen Schritten erzeugt wurden. In der prototypischen Implementierung werden genau die Bilder verwendet, die der Anwender mit Hilfe der ZIP Datei in das System geladen hat. In einer späteren Anwendung steht es aber offen, welche Bilder aus dem Repository verwendet werden.

Der DocumentController erzeugt aus den Metadaten, die mit dem ServiceRequest-Objekt übergeben wurden, ein neues MCRObject. Auch hier treten die in Abschnitt 6.5.1 genannten Probleme in Bezug auf das Einstellen in das Repository auf. Eine weitere Einschränkung ist, dass man mit dieser Klasse einer Bildserie keine Derivate zuordnen kann.

Um Bilder der Bildserie hinzuzufügen, muss mindestens ein Beziehungstyp erstellt werden. Dieser Typ setzt Bilder in Beziehung zueinander. Um Bilder in einer Reihenfolge zu speichern und sie mit einer Interaktionsmöglichkeit zu versehen, wird die Klasse `SlideShowInteraction.java` verwendet. Sie implementiert das Interface DocumentRelations.

Die folgende Liste beschreibt die Methoden der Interfacesimplementierung:

- *setMetadata* setzt ein globales Objekt, das Metadaten über die Beziehungen enthält.
- *setMMObject* setzt das Multimedia Objekt, welchem die Beziehungen hinzugefügt werden sollen.
- *setDataObject* setzt den globalen Vector `ObjectUris`, der die Objekt IDs aller Bilder hat, die der Bildserie hinzugefügt werden sollen.
- *doPrepare* startet die Abarbeitung.

Wie bereits in Abschnitt 6.3.2 vorgestellt, wird für jedes Bild eine Referenz mit Hilfe einer MCRMetaLinkID erzeugt.

6.5.3 ReviewController und DocumentReviewModul

Die Nutzerschnittstelle für den Kontrollprozess einer Bildserie ist das Servlet `DocumentReviewServlet.java`. Es gibt alle Entscheidungen der Nutzer an den ReviewController weiter.

Der ReviewController ruft die Klasse `DummyCheck.java` auf, die das Interface DocumentChecker implementiert. Die Klasse `DummyCheck.java` besitzt keine sinnvolle Funktionalität. Die Daten werden nur zur Demonstration des Interfaces an die Klasse übergeben.

6.5.4 LibrarianController und LibrarianEditing

Das Servlet `LibrarianEditingServlet.java` verarbeitet alle Nutzereingaben, die das bibliothekarische Evaluieren betreffen. Die Eingaben werden an den Service-Controller LibrarianController weitergereicht. Der Nutzer hat nur die Möglichkeit Metadaten zu verändern, um so z.B. die korrekte Angabe einer Klassifikation zu kontrollieren oder eine Klassifikation hinzuzufügen.

MyCoRe bietet mit dem Konzept der Editoren eine einfache Möglichkeit, vorhandene Dokumente zu bearbeiten.Die Metadaten werden mit dem Editor `editor-commit-mmimageseries.xml` bearbeitet. Das Ergebnis des Editors wird durch die *MyCoRe* Klasse `MCRCheckCommitDataServlet.java` verarbeitet. Sie führt ein Update der Metadaten im Repository durch. Der LibrarianController wird für die Kommunikation mit dem WFMS verwendet. Es ist keine eigene Service Komponente notwendig.

6.5.5 Object Store

Um Daten Objekte vom *MyCoRe* System zu erhalten wird die Klasse `MyCoReDaoStore.java` verwendet. Sie implementiert das Interface DaoStore. Die Methoden sind folgendermaßen implementiert:

- *getNewObject* liefert ein leeres MCRObject zurück.
- *saveObject* wird nicht benötigt, da in *MyCoRe* sich die Objekte mit eigenen Methoden im Repository speichern.
- *getObject* liefert ein MCRObject zu der angegbenen ID.

Die Klasse wurde zusätzlich mit einer Methode erweitert, die gültige Objekt IDs erzeugt. In *MyCoRe* muss für jedes neue Objekt eine gültige ID erzeugt werden, bevor es abgespeichert werden kann.

6.6 Workflow Engine *Enhydra Shark*

In Abschnitt 5.3 wurde die Anbindung eines Workflow-Management-Systems (WFMS) beschrieben. Aufbauend darauf, wird das WorkflowInterface durch die Klasse

`ATLSharkWorkflow.java` implementiert. Diese Implementierung wird aus der laufenden Anwendung *@libri* weiter genutzt. Sie ist im Rahmen der Arbeit [Sch04] entstanden und weiterentwickelt worden.

Die Klasse `ATLSharkWorkflow.java` stellt eine Anbindung der Workflow Engine (WFE) *Enhydra Shark* zur Verfügung. Die WFE verwendet die Beschreibungssprache *XPDL* für Workflow Beschreibungen. *XPDL* wird von der Workflow Management Coalition (WfMC) entwickelt und stellt einen Standard für Workflow Beschreibungen dar. Ein Workflow kann mit Hilfe des Tools *Enhydra JaWE* erstellt werden. Das Tool erzeugt eine *XPDL* konforme Ausgabe, die mit Shark weiter verwendet werden kann. Eine Beschreibung erfolgt in [Sch04].

Um die Workflow Anbindung zu demonstrieren, wurde der in Abbildung 6.7 dargestellte Workflow erstellt. Er unterstützt den Publikationsprozess für eine Bildserie.

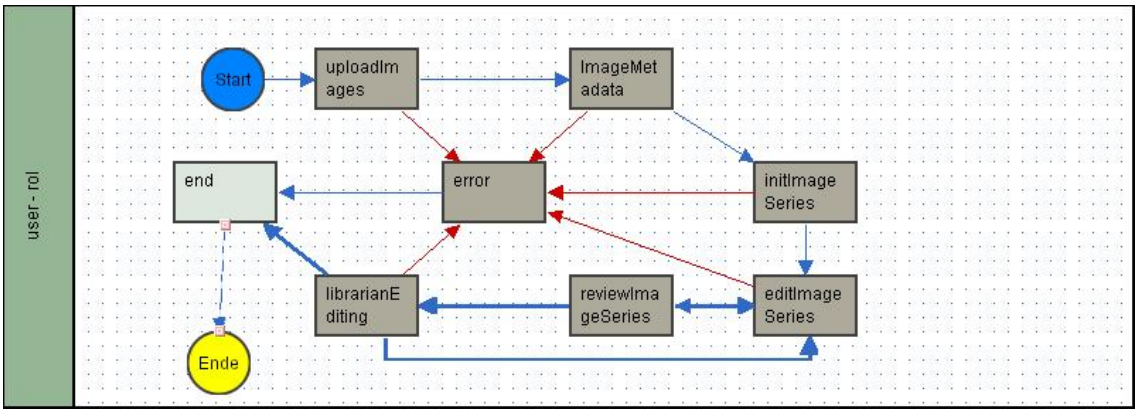

Abbildung 6.7: Beispiel Workflow für eine Bildserie

Die Aktivitäten des Workflows werden im Folgenden kurz erläutert:

1. **Start und Ende**: Die beiden Aktivitäten beschreiben den Beginn und das Ende eines Workflows.

2. **uploadImages**: Diese Aktivität beschreibt den Zustand, in dem Dateien in das System geladen werden. Der Service-Controller AssetController kann in diesem Zustand verwendet werden.

3. **ImageMetadata**: Die Vergabe von Metadaten ist in in diesem Zustand möglich. Es kann der Service-Controller AssetController verwendet werden.

4. **initImageSeries**: In diesem Zustand können Bildserien erzeugt werden. Hier hat der Service-Controller DocumentController Zugriff.

5. **editImageSeries**: Dieser Zustand ermöglicht das Bearbeiten einer Bildserie, nachdem sie angelegt wurde.

6. **reviewImageSeries**: In diesem Zustand können Bildserien überprüft werden. Der Service-Controller ReviewController kann in diesem Fall verwendet werden. Wenn eine Bildserie zurückgewiesen wird, ruft das WFMS die Aktivität *editImageSeries* auf.

7. **librarianEditing**: In diesem Zustand wird die bibliothekarische Evaluierung durchgeführt. Die Bildserie kann abgelehnt werden. In diesem Fall wird der Workflow in den Zustand *editImageSeries* zurückgesetzt. Außerdem kann die Bildserie gelöscht werden.

8. **error**: Diese Aktivität wird aufgerufen, wenn ein Fehler auftritt.

9. **end**: Die Aktivität wird eingeführt, da in XPDL die letzte Aktivität keine ausgehende Transition besitzen darf, mit Ausnahme der Transition *Ende*.

7 Schlussbetrachtungen

7.1 Zusammenfassung

Mit den wachsenden technischen Möglichkeiten wird das Einsatzgebiet von Multimediadokumenten immer größer. Autoren stehen z.B. vor der Aufgabe, multimediale Lehrmaterialien oder komplex strukturierte Dokumente zu erstellen. Diese sollen einem großem Nutzerkreis in digitalen Bibliotheken zur Verfügung gestellt werden. Eine Unterstützung der Autoren bei der Modellierung solcher Dokumente findet aber häufig nur bei der Erstellung von Teildokumenten statt. Aus diesem Grund sollten digitale Bibliotheken Publikationsprozesse für Multimediadokumente anbieten.

Im Rahmen dieser Arbeit wurde ein Publikationsprozess für Multimediadokumente in digitalen Bibliotheken erstellt. Um den Publikationsprozess an den Anforderungen von aktuellen Multimedia Dokumentenmodellen auszurichten, wurden im Vorfeld die Dokumentenmodelle *MPEG-7* und *METS* untersucht. Vorher definierte Anforderungen an Multimedia Dokumentenmodelle in digitalen Bibliotheken waren die Grundlage für die Betrachtungen. Für die Integration des Publikationsprozesses in digitale Bibliotheken wurden der Content-Life-Cycle von Content-Management-Systemen und digitalen Bibliotheken analysiert.

Das zentrale Ergebnis der Arbeit ist ein Java Framework für den Publikationsprozess von Multimediadokumenten in digitalen Bibliotheken. Das Framework basiert auf dem erstellten Publikationsprozess und baut auf dem Meta-Framework *Spring* und dem Framework *x18p* auf. Das Framework verwaltet so genannte Service-Module, die entsprechende Service-Anfragen bearbeiten. Module für das Erfassen von Content bieten dem Autor die Möglichkeit beliebigen Content in das System zu laden. So können auch vorhandene Werkzeuge integriert werden. Nachdem der Content in ein einheitliches Format überführt wurde, kann eine medienspezifische Deskribierung vorgenommen werden. Das Framework unterstützt neben Modulen für die manuelle Deskribierung auch Module für die [semi-] automatische Deskribierung von Inhalten. Für die Modellierung von Multimediadokumenten können Module integriert werden, die komplexe Beziehungen erzeugen. Um Review-Prozesse bei der Publikation zu unterstützen, können Module für das Überprüfen und Testen von Multimediadokumenten verwaltet werden. Die bibliothekarische Evaluierung von Content wird durch den vorgeschlagenen Publikationsprozess ebenfalls unterstützt. Das Ausführen des Publikationsprozesses kann durch die Angabe eines Workflows unterstützt und kontrolliert werden.

Das vorgestellte Framework wurde prototypisch in das digitale Bibliothekssystem der Universitätsbibliothek Rostock integriert. Das System baut auf dem Content-Repository

MyCoRe auf. Anhand eines Dokumentenmodells für Bildserien wurde die Verwendung des Frameworks exemplarisch gezeigt.

7.2 Ausblick

7.2.1 Publikation von Multimediadokumenten

In der Arbeit wurde ein Publikationsprozess für Multimediadokumente in digitalen Bibliotheken eingeführt. Dieser Publikationsprozess beschreibt auf einer allgemeinen Ebene die Schritte, die ausgeführt werden müssen, um ein Dokument zu erzeugen. Die Unterstützung von Autoren bezieht sich aber nicht nur auf die Vorgabe eines Publikationsprozesses. Ebenso wichtig ist ein Autorensystem, mit dessen Hilfe Multimediadokumente modelliert werden können. Der Autor muss verschiedene Informationen auf unterschiedlichen Ebenen angeben. Er muss Elemente auswählen und ein räumliches Layout definieren. Weiterhin müssen zeitliche Zusammenhänge angegeben und Interaktionen bzw. Links definiert werden. Zusätzlich muss der Inhalt an die Anforderungen der Ausgabeumgebung angepasst werden. Anforderungen an ein zu erstellendes künftiges System können sein [JRT00]:

- Die Definition von Nutzer Interfaces, die effektive Autorensysteme zur Verfügung stellen.
- Autorenumgebungen, die den WYSIWYG Ansatz unterstützen.
- Graphische Schnittstellen, die mehrere Sichten bereitstellen.

Die genannten Anforderungen sollen unabhängig von dem verwendeten Dokumentenmodell sein.

Bei der Einführung von WYSIWG Editoren mit mehreren Sichten für Multimediadokumente, müssen ebenfalls neue Paradigmen für die Erstellung gefunden werden. In der Literatur werden verschiedene genannt. [BH05] diskutiert folgende Ansätze:

- Struktur basiert
- Zeitstrahl basiert
- Graph basiert
- Skript basiert

Bei der Auswahl von Elementen für ein Multimediadokument müssen die Möglichkeiten des Systems erweitert werden. Es wird z.B. versucht, Deskriptoren, die voranging auf dem Gebiet des Information Retrieval verwendet werden zu nutzen, um Fragmente eines Dokumentes zu integrieren. In aktuellen Systemen wie *MyCoRe* werden Dokumente oft als Black-Box Elemente betrachtet. Die Autorensysteme bieten keine Möglichkeit, Medienfragmente zu integrieren. Wenn der Autor z.B. nur eine Szene aus einem Video

verwenden möchte, muss er die Szene vorher mit einem entsprechenden Tool ausschneiden und als eigenständiges Dokument speichern. In [TR03] wird ein Ansatz auf der Basis von *MPEG-7* vorgestellt.

7.2.2 Softwaretechnische Erweiterungen

Die Integration von verschiedenen Content-Repositories ist häufig damit verbunden, dass für jeden Anbieter eine eigene API eingebunden wird. Dies hat den Nachteil, dass ein großer Teil der Informationen von Anbietern gar nicht verwendet werden kann. Abhilfe soll hier die JSR 170 (Java API for a Content Repository) schaffen. Sie bietet einen standardisierten Weg, um implementierungsunabhängig auf Content-Repositories zuzugreifen.

Ein weiterer Schritt kann die Erweiterung der allgemeinen Service-orientierten Architektur (SOA) des Frameworks sein. Ansätze dazu sind:

- *UDDI* (Universal Description, Definition, and Integration) ist ein Standard um Services zu registrieren.
- Mit *QoS* (Quality of Service) werden Sicherheitsanforderungen, Authentifizierung und Autorisierung für Services definiert. Wichtig ist auch die zuverlässige Nachrichtenübermittlung bzw. Richtlinien dafür, wer Services aufrufen darf.
- *EDA* (event-driven architecture) ermöglicht die Verwaltung von asynchronen Ereignissen. Neue Services können einfach eingebunden werden und die Wiederverwendbarkeit soll weiter verbessert werden. Systeme wie Mule[1] können für die Integration von EDA verwendet werden. Dabei unterstützt Mule z.B. explizit das Meta-Framework *Spring* , auf dem das hier vorgestellte Framework basiert [Han05].

[1] `http://mule.codehaus.org/`

Literaturverzeichnis

[All83] Allen, James F.: Maintaining knowledge about temporal intervals. In: *Commun. ACM* 26 (1983), Nr. 11, S. 832–843. – ISSN 0001–0782

[BG00] Baca, Murtha ; Gilliland, Anne J. *Introduction to Metadata.* Website: `http://www.getty.edu/research/conducting_research/standards/intrometadata/index.html`. 2000

[BH05] Bulterman, Dick C. A. ; Hardman, Lynda: Structured multimedia authoring. In: *TOMCCAP* 1 (2005), Nr. 1, S. 89–109

[BMRS02] Benitz, Ana B. ; Martínez, José M. ; Rising, Hawley ; Salembier, Philippe: Description of a Single Multimedia Document. In: Salembier, Phillipe (Hrsg.) ; Sikora, Thomas (Hrsg.): *Introduction to MPEG-7: Multimedia Content Description Interface.* John Wiley & Sons, Inc., 2002, Kapitel 8, S. 111–138

[Boi01] Boiko, Bob: *Content Management Bible.* New York, NY, USA : John Wiley & Sons, Inc., 2001. – ISBN 076454862X

[Bol01] Boll, Susanne: *ZYX – Towards flexible multimedia document models for reuse and adaptation*, Vienna University of Technology, Diss., 2001

[BS02] van Beek, Peter ; Smith, John R.: Navigation and Summariation. In: Salembier, Phillipe (Hrsg.) ; Sikora, Thomas (Hrsg.): *Introduction to MPEG-7: Multimedia Content Description Interface.* John Wiley & Sons, Inc., 2002, Kapitel 9, S. 140–151

[BYF02] van Beek, Peter ; Yoon, Kyoungro ; Ferman, A. M.: User Interaction. In: Salembier, Phillipe (Hrsg.) ; Sikora, Thomas (Hrsg.): *Introduction to MPEG-7: Multimedia Content Description Interface.* John Wiley & Sons, Inc., 2002, Kapitel 11, S. 163–175

[EF00] Endres, Albert ; Fellner, Dieter W.: *Digitale Bibliotheken.* 1. Aufl. dpunkt-Verl., 2000. – ISBN 3–932588–77–0

[GHJV01] Gamma, Erich ; Helm, Richard ; Johnson, Ralph ; Vlissides, John: *Entwurfsmuster.* 5. Addison-Wesley, 2001

[Göt] Göttingen, SUB. *Metadata Server - Einführung in Metadaten.* Website `http://www2.sub.uni-goettingen.de/metaguide/index.html`

[Hac97] HACKER, Rupert: *Bibliothekarisches Grundwissen*. 6. K G Sauer, 1997

[Han05] HANSON, Jeff. *Event-driven services in SOA*. Website: `http://www.javaworld.com/javaworld/jw-01-2005/jw-0131-soa.html`. 2005

[HD02] HEUER, Andreas ; DITTRICH, Klaus R.: Schwerpunktthema: Content Management und digitale Bibliotheken. In: *Datenbank-Spektrum* 2 (2002), Nr. 4, S. 7–8

[Heu] HEUER, Andreas. *Vorlesung Multimedia-Datenbanken*. Vorlesung

[HM05] HEUER, Andreas ; MEYER, Holger. *Informationssysteme und -dienste*. Vorlesung. 2004/2005

[HS] HEIMRICH, Thomas ; SPECHT, Günther. *Struktur-, Verhaltens- und Ausgabeschemata für Multimedia-Daten in objektrelationalen Datenbanksystemen*. `http://www.old.netobjectdays.org/pdf/03/papers/ws-mik/314.pdf`

[HW04] HOFFMANN, Thorsten ; WOLFRATH, Michael. *Framework ahoi!* `http://javamagazin.de/itr/online_artikel/show.php3?nodeid=11&id=597`. 08 2004

[JRT00] JOURDAN, Muriel ; ROISIN, Cécile ; TARDIF, Laurent: A Scalable Toolkit for Designing Multimedia Authoring Environments. In: *Multimedia Tools Appl.* 12 (2000), Nr. 2/3, S. 257–279

[Kos02] KOSCH, Harald: MPEG-7 and multimedia database systems. In: *SIGMOD Rec.* 31 (2002), Nr. 2, S. 34–39. – ISSN 0163–5808

[Lib] LIBRARY OF CONGRESS, The. *METS - Metadata Encoding & Transmission Standard*. `http://www.loc.gov/standards/mets/`

[LKDa] LÜTZENKIRCHEN, F. ; KUPFERSCHMIDT, J. ; DEGENHARDT, D. *MyCoRe Starting Guide*. `http://www.mycore.de/content/main/documentation.xml`

[LKDb] LÜTZENKIRCHEN, F. ; KUPFERSCHMIDT, J. ; DEGENHARDT, D. *MyCoRe User Guide*. `http://www.mycore.de/content/main/documentation.xml`

[MPE05] PEREIRA, F. (Hrsg.). *MPEG-7 Requirements Document V.18*. `http://www.chiariglione.org/mpeg/working_documents/mpeg-07/requirements/requirements.zip`. January 2005

[SB02] SMITH, John R. ; BENITEZ, Ana B.: Content Organization. In: SALEMBIER, Phillipe (Hrsg.) ; SIKORA, Thomas (Hrsg.): *Introduction to MPEG-7: Multimedia Content Description Interface*. John Wiley & Sons, Inc., 2002, Kapitel 10, S. 153–161

[Sch02] SCHMITT, Ingo: Retrieval in Multimedia-Datenbanksystemen. In: *Datenbank-Spektrum* 2 (2002), Nr. 4, S. 28–35

[Sch04] SCHICK, Sebastian: *Eine Workflow-Komponente für Digitale Bibliothekssysteme*. 2004. – Studienarbeit

[SHS05] SAAKE, Gunter ; HEUER, Andreas ; SATTLER, Kai-Uwe: *Datenbanken: Implementierungstechniken*. 2. mitp, 2005

[SS02a] SALEMBIER, Philippe ; SMITH, John R.: Overview of Multimedia Description Schemes and Schema Tools. In: SALEMBIER, Phillipe (Hrsg.) ; SIKORA, Thomas (Hrsg.): *Introduction to MPEG-7: Multimedia Content Description Interface*. John Wiley & Sons, Inc., 2002, Kapitel 6, S. 83–93

[SS02b] SALEMBIER, Phillipe ; SIKORA, Thomas: *Introduction to MPEG-7: Multimedia Content Description Interface*. New York, NY, USA : John Wiley & Sons, Inc., 2002. – ISBN 0471486787

[TMPP04] TUMMARELLO, Giovanni ; MORBIDONI, Christian ; PIAZZA, Francesco ; PULITI, Paolo. *Facing the hard problem: automatic RDF annotations from MPEG7 streams*. `http://semanticweb.deit.univpm.it/swap2004/cameraready/morbidoni.pdf`. 2004

[TR03] TIEN, Tran-Thuong ; ROISIN, Cécile: Multimedia modeling using MPEG-7 for authoring multimedia integration. In: *Multimedia Information Retrieval*, 2003, S. 171–178

[WHM02] WALKER, Toby ; HEUER, Jörg ; MARTÍNEZ, José M.: Basic Elements. In: SALEMBIER, Phillipe (Hrsg.) ; SIKORA, Thomas (Hrsg.): *Introduction to MPEG-7: Multimedia Content Description Interface*. John Wiley & Sons, Inc., 2002, Kapitel 7, S. 95–109

[Wu04] WU, Fangjian. *Design a simple service-oriented J2EE application framework*. `http://www.javaworld.com/javaworld/jw-10-2004/jw-1004-soa.html#resources`. 10 2004

[Zie03] ZIEGELER, Carsten. *Die Nebel von Avalon*. `http://javamagazin.de/itr/online_artikel/show.php3?nodeid=11&id=276`. 01 2003

Abbildungsverzeichnis

A Anhang

A.1 Web-Anwendung für Bildserien mit *@libri*

Die digitale Bibliothek der Universitätsbibliothek Rostock (*@libri*) wurde durch ein Framework erweitert (siehe Kapitel 6). Wie das Framework verwendet wird, sollen Grafiken aus der Web-Anwendung zeigen.

Für die Demonstration wird eine Bildserie über Schiffe, die an der *Hanse Sail 2004* teilgenommen haben, angelegt. Dafür müssen folgende Schritte ausgeführt werden:

- eine zip Datei auf den Server laden
- Metadaten für die Bilder angeben
- Metadaten für die Bildserie angeben

Abbildung A.1: Startseite für den Publikationsprozess

In Abbildung A.1 wird die Startseite für das Erzeugen einer Bildserie gezeigt. Diese Seite dient als Einstiegspunkt für alle weiteren Schritte. Der Nutzer muss sich über den in *MyCoRe* System üblichen Weg anmelden. Danach kann er die Seite *Bildserie erstellen* aufrufen. Dort wird mit Hilfe eines Workflow-Management-Systems die aktuelle Aktivität für den Nutzer ermittelt und angezeigt. In der Liste wird der Punkt als Link ausgegeben. Alle anderen Elemente können nicht ausgewählt werden.

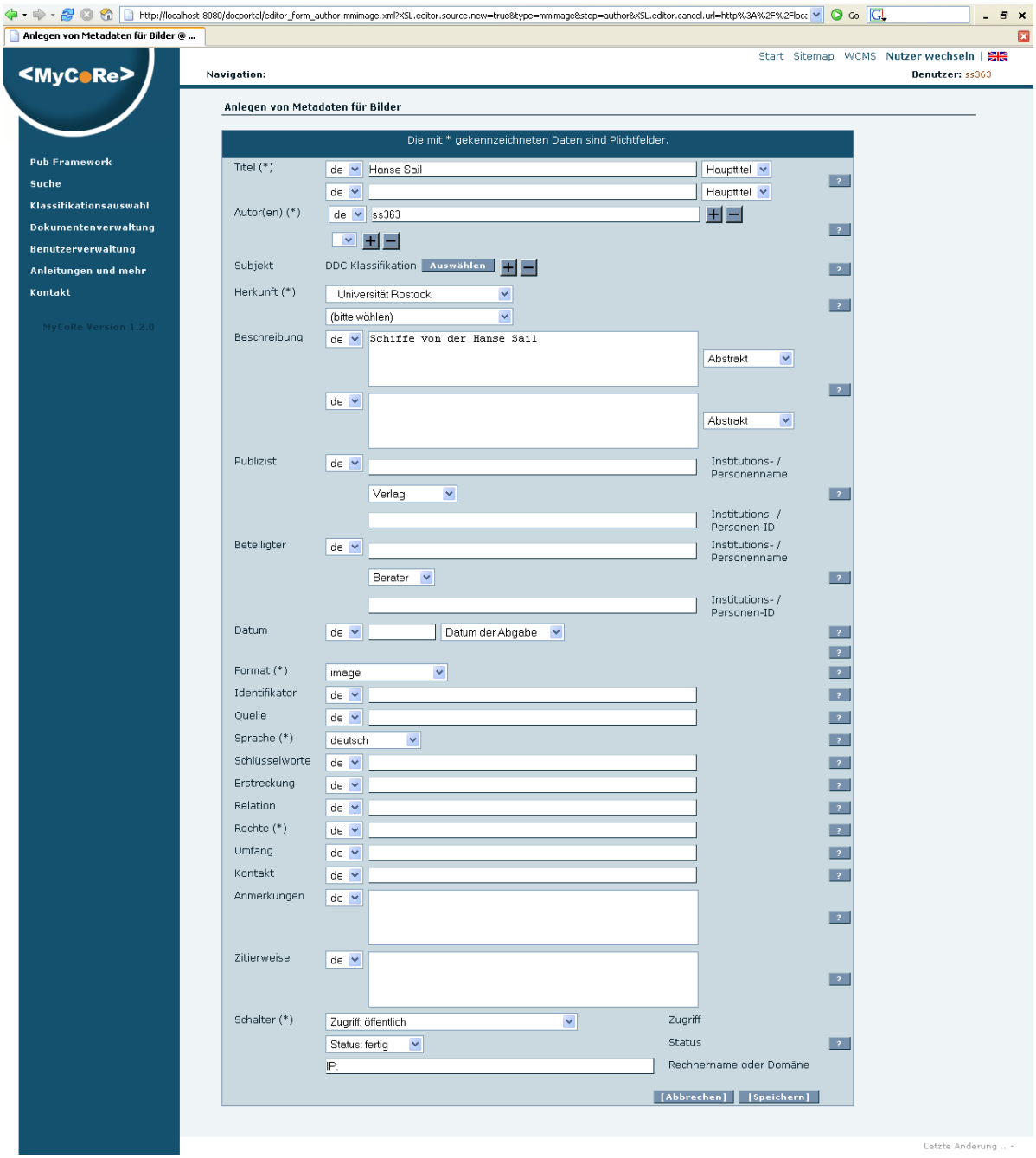

Abbildung A.2: Anlegen von Metadaten für Bilder

Nachdem der Nutzer über ein einfaches HTML Datei-Upload Formular die zip Datei auf den Server geladen hat, muss er Metadaten für die Bilder eingeben.

Abbildung A.2 zeigt das Formular (in *MyCoRe* Editor genannt), in das die Metadaten

eingegeben werden. Im aktuellen Prototyp gibt es keine Möglichkeit zur Auswahl von Dateien. Die eingegebenen Metadaten werden für alle Bilder verwendet, die in der zip Datei enthalten sind.

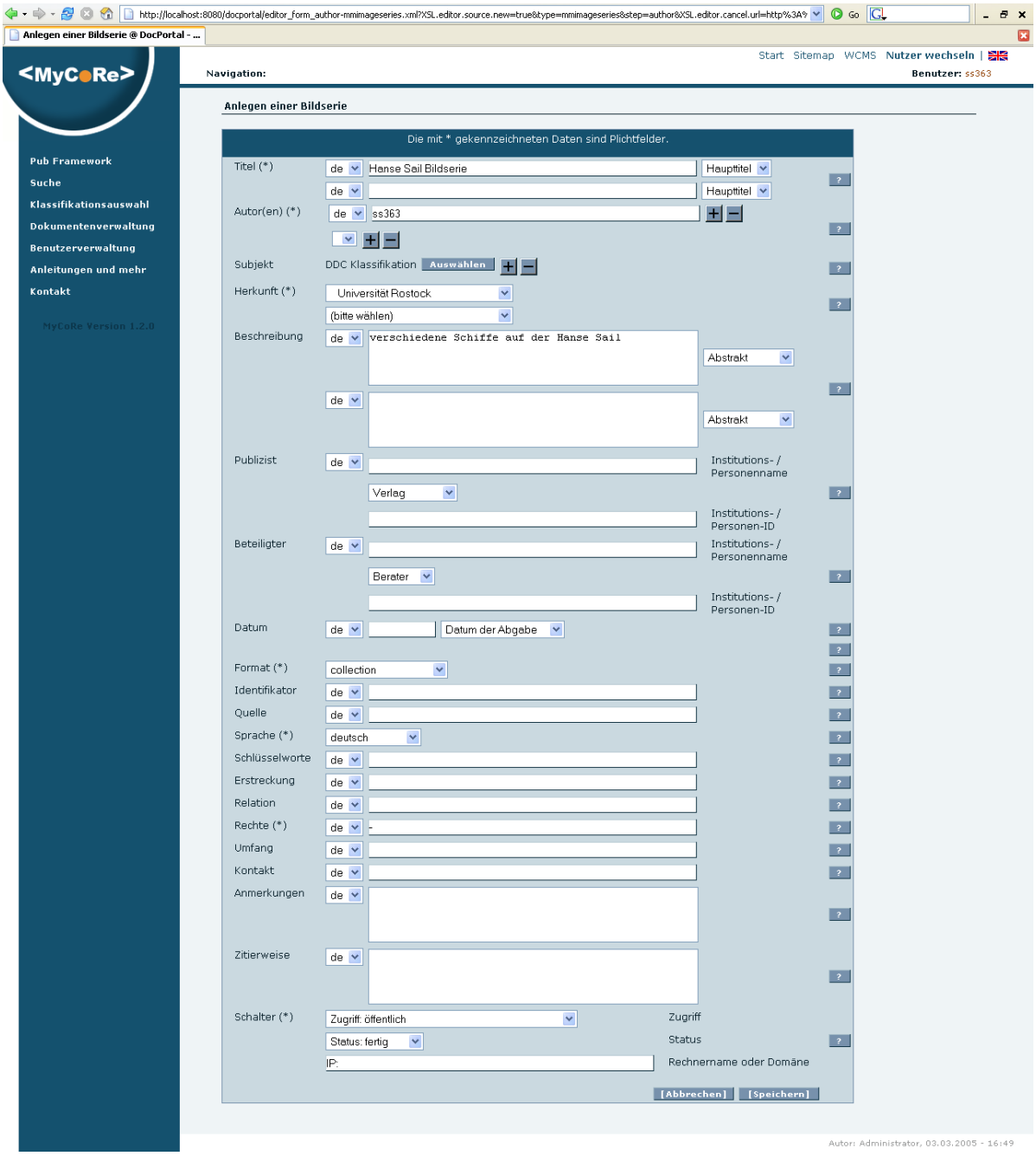

Abbildung A.3: Anlegen von Metadaten für eine Bildserie

Die Metadaten für eine Bildserie werden über ein ähnliches Formular wie für Bilder

eingegeben. Das Formular wird in Abbildung A.3 gezeigt. Die Formulare für Bildserien und Bilder unterstützen den Dublin Core Standard.

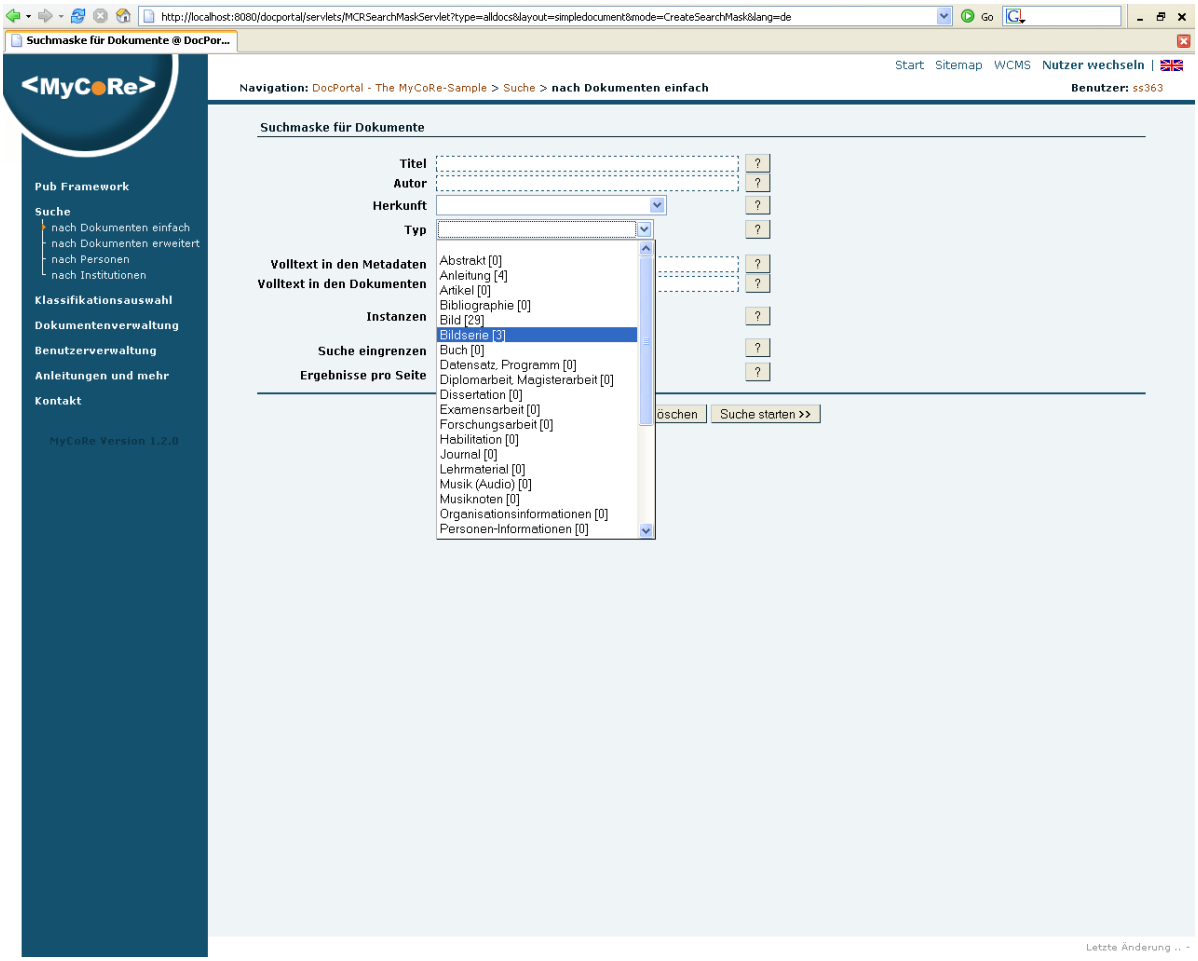

Abbildung A.4: Suchmaske mit Bildserie

Nachdem die Metadaten für die Bilder und Bildserien angeben sind, können die Bilder und die Bildserie in der digitalen Bibliothek gesucht werden. Ein Suchformular wird in Abbildung A.4 gezeigt. Der Nutzer kann die Suche zusätzlich auf die Dokumenttypen Bilder oder Bildserien einschränken.

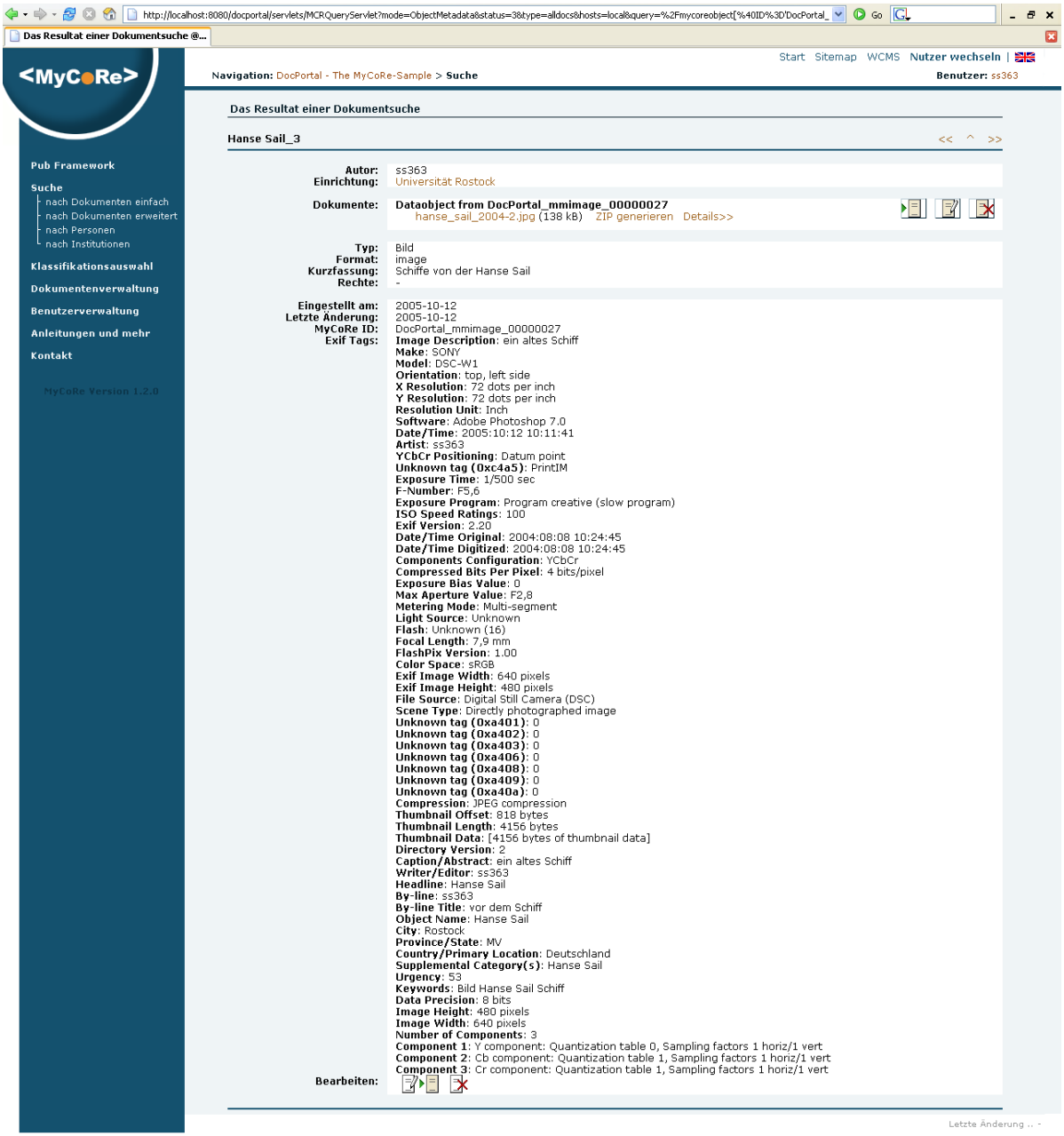

Abbildung A.5: Metadaten eines Bildes

Ein Metadatensatz für ein Bild wird in Abbildung A.5 dargestellt. Es werden auch Metadaten gezeigt, die nicht über das Formular eingeben wurden. Im unteren Teil der Grafik sind Metadaten aufgelistet, die in einem jpeg Bild enthalten sind. Sie wurden automatisch aus dem Bild extrahiert und dem Metadatensatz des Bildes hinzugefügt.

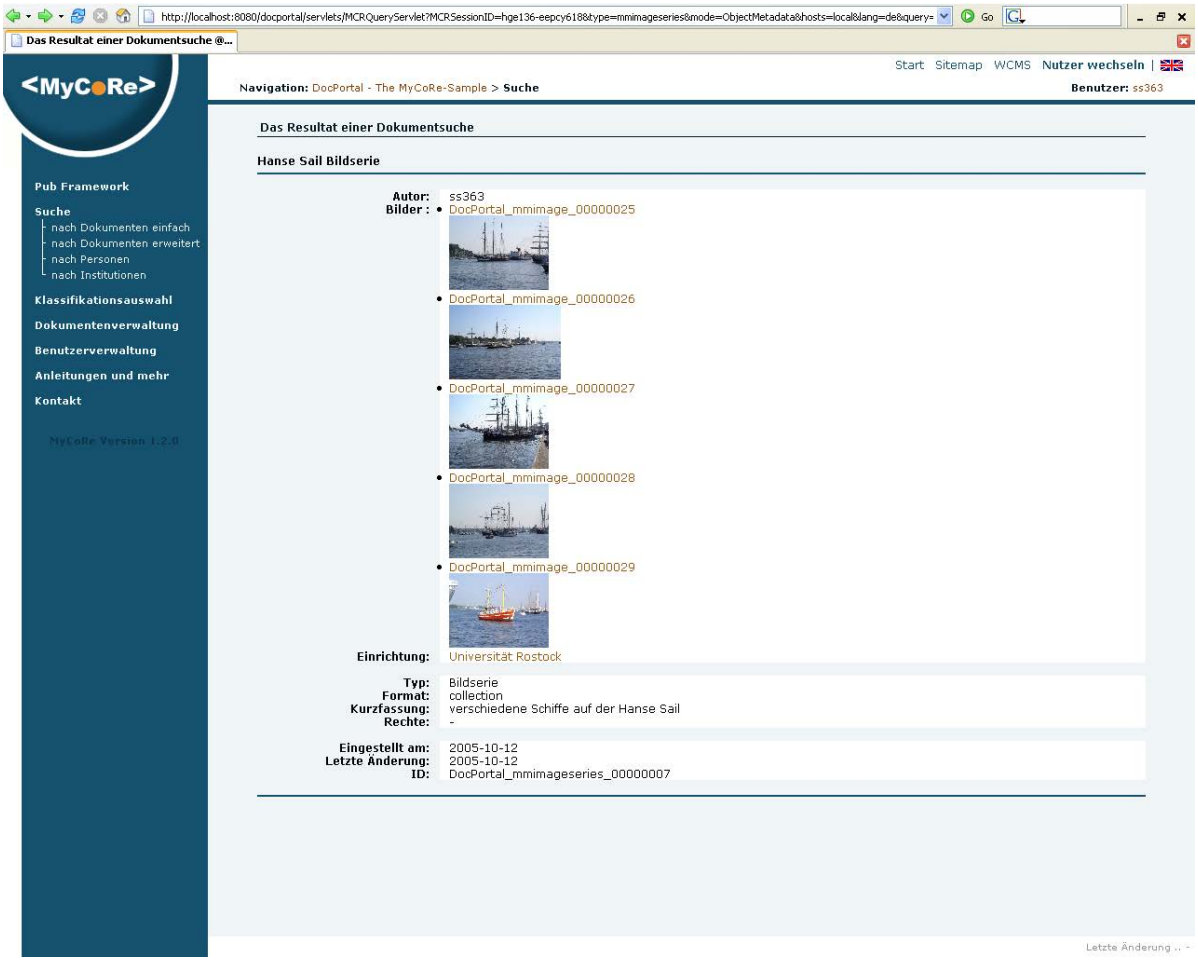

Abbildung A.6: Metadaten einer Bildserie

In Abbildung A.6 werden die Metadaten einer Bildserie dargestellt. Eine Bildserie enthält Bilder, die als Thumbnails untereinander abgebildet werden. Die Bildserie enthält alle Bilder für die im vorherigen Schritt Metadaten angegeben wurden. Zu jeden Bild gibt es einen Link, der das entsprechende Bild-Datenobjekt refenrenziert. Zusätzlich ist es möglich, dass der Nutzer eine Bildshow öffnet. Dazu kann er ein Bild anklicken.

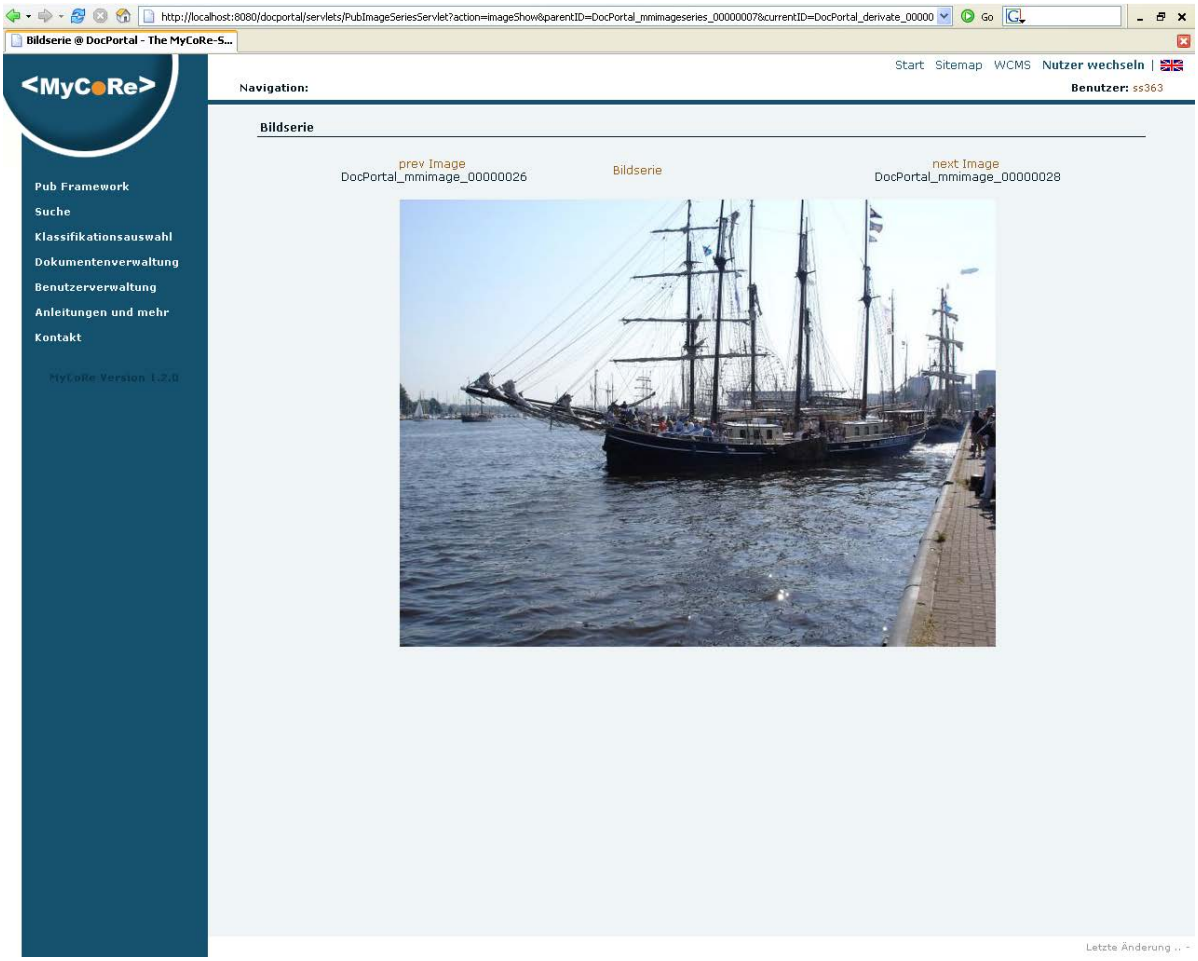

Abbildung A.7: Bildserie

Eine Bildshow zeigt die Bilder, die zu einer Bildserie gehören in ihrer originalen Größe. Dabei wird immer ein Bild angezeigt. Wenn der Nutzer eine Bildshow geöffnet hat, startet die Anzeige immer mit dem Bild, welches er angeklickt hat. Abbildung A.7 zeigt eine Bildshow. Die Bilder einer Bildserie haben eine Reihenfolge. Der Nutzer kann im oberen Bereich der Seite das vorherige Bild oder das nächste Bild auswählen. Die Bildserie zu dem die Bilder gehören, kann über den Link Bildserie erreicht werden.

Nachdem eine Bildserie erzeugt wurde, kann sie in einem zweiten Schritt überprüft werden. Mit Hilfe eines Editors können die Metadaten verändert werden. Zusätzlich zu den Metadaten, die auch bei der Erzeugung von Bildserien anzugeben sind, werden die Bilder einer Serie aufgelistet. Der Nutzer hat die Möglichkeit Bilder zu entfernen, Bilder hinzuzufügen oder die Reihenfolge der Bilder zu verändern (siehe Abbildung A.8).

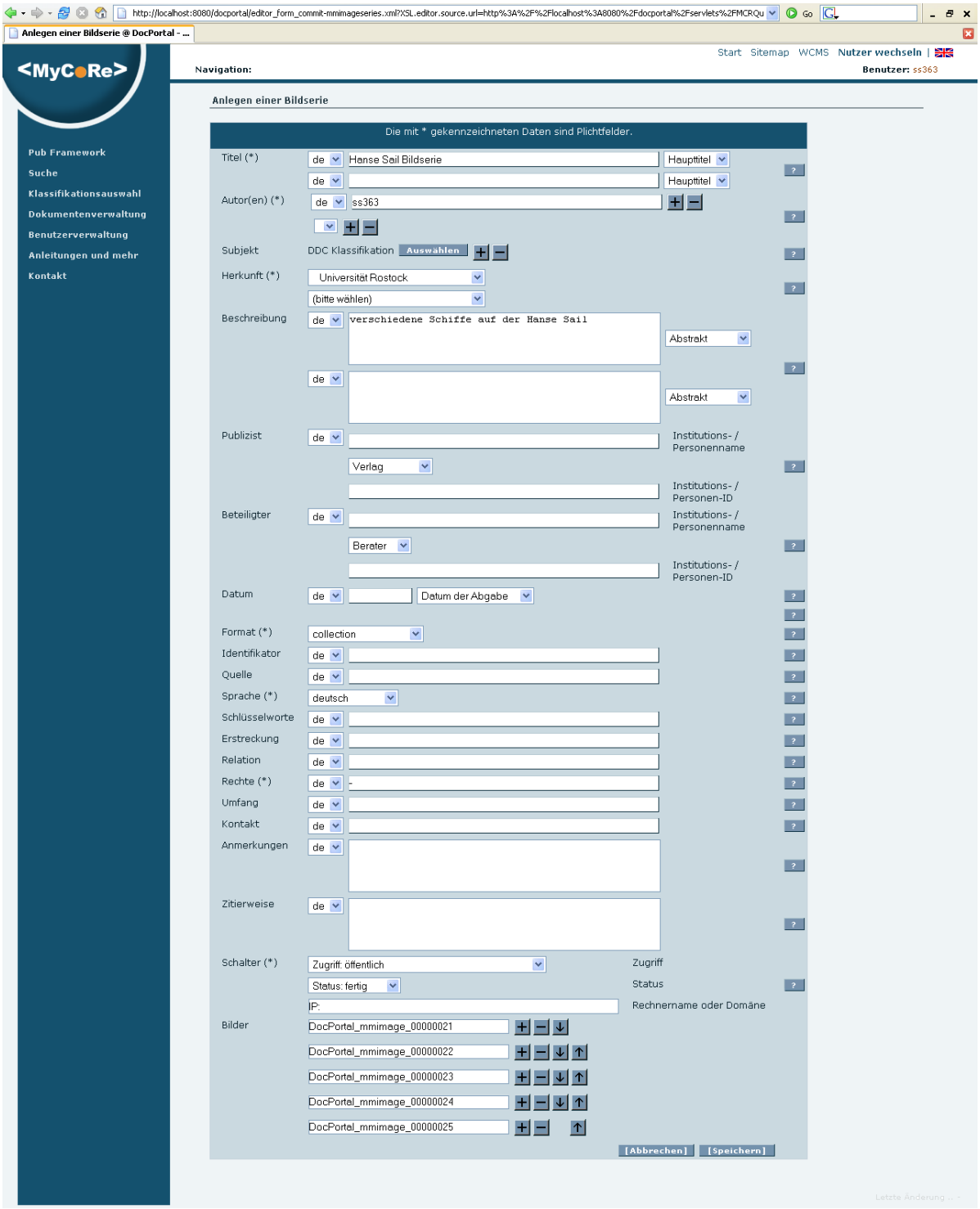

Abbildung A.8: Review Bildserie

Thesen

1. Autoren erstellen komplexe Multimediadokumente und möchten diese in digitalen Bibliotheken veröffentlichen. Das Modellieren der Dokumente wird häufig nicht unterstützt. Deshalb sollten digitale Bibliotheken Publikationsprozesse für Multimediadokumente anbieten.

2. Multimediadokumente lassen sich durch Multimedia Dokumentenmodelle darstellen. Die Modelle *MPEG-7* und *METS* sind aktuelle Beispiele für solche Modelle. Der Publikationsprozess wird an Anforderungen dieser aktuellen Modelle ausgerichtet.

3. Content-Management-Systeme werden verwendet, um die Publikation von Dokumenten zu unterstützen. Der Content-Life-Cycle in diesen Systemen soll als Ausgangspunkt für die Definition des Publikationsprozesses verwendet werden.

4. Entwurfsmuster und Frameworks erhöhen die Qualität von Software und ermöglichen eine einfachere Wiederverwendbarkeit. Um diese Vorteile zu nutzen, wird der Publikationsprozess als Java Framework entwickelt.

5. Das Trennen von Geschäftslogik und Präsentationslogik verhindert Abhängigkeiten in Applikationen. Die Service orientierte Architektur des Frameworks *x18p* unterstützt diese Trennung und wird hier verwendet.

6. Module und Komponenten können einfacher ausgetauscht werden, wenn sie nach dem Prinzip des Inversion-of-Control und Dependency-Injection entworfen sind. Das Framework Spring bietet dafür alle benötigten Funktionen an.

7. Das Einbinden von vorhandenen Autorenwerkzeugen minimiert die wiederholte Eingabe von Informationen.

8. Im Publikationsprozess müssen Arbeitsabläufe effizient verwaltet und kontrolliert werden. Ein Workflow-Management-System kann diese Aufgabe übernehmen.

9. Der Entwickelte Prototyp demonstriert die Integration des Frameworks in einer digitalen Bibliothek.

Selbständigkeitserklärung

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

Rostock, den 24. Oktober 2005