Technische Universität Ilmenau
Fakultät für Elektrotechnik und Informationstechnik

Diplomarbeit

Softwarearchitektur eines User Interfaces für ein digitales Filmarchiv

vorgelegt von:

Mathias Weis

Matrikel-Nr.: 32969

Studiengang: Medientechnologie

Fachrichtung: Medienproduktion

Verantwortlicher Professor: Prof. Dr.-Ing. Karlheinz Brandenburg

Betreuender wiss. Mitarbeiter: Dipl.-Ing. Christian Weigel

Betreuer am Fraunhofer IIS: Dipl.-Ing. Arne Nowak

Ilmenau, 31. Mai 2008

urn:nbn:de:gbv:ilm1-2008200020

Danksagung

An dieser Stelle möchte ich denjenigen „Vielen Dank!“ sagen, die mich bei der Diplomarbeit unterstützt haben. Ich danke Dipl.-Ing. Arne Nowak vom Fraunhofer IIS sowie Prof. Dr.-Ing. Karlheinz Brandenburg und Dipl.-Ing. Christian Weigel vom Institut für Medientechnik der TU Ilmenau für die Ermöglichung und die Betreuung dieser Arbeit. Außerdem danke ich Imke Hoppe für den seelischen Beistand, die Tipps zur Gestaltung und die Hilfe bei der Korrektur der Arbeit.

Diese Arbeit entstand im Rahmen des EDCine-Projektes. Das EDCine - Enhanced Digital Cinema - Projekt wird finanziert von der Europäischen Komission im Rahmen des EU-Programms FP6/2004/IST/4.1, Vertragsnr. 038454 EDCine.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1.

Einleitung

Die Kunst des bewegten Bildes ist ein großer Teil des Kulturschatzes der Gegenwart und des vergangenen Jahrhunderts. Mit dem Einzug des „Digitalen Zeitalters“ und der Möglichkeit der digitalen Aufbewahrung dieser Werke ergeben sich einige Vorteile für die Archivierung von Filmen. Im Gegensatz zum herkömmlichen Filmarchiv, in dem das Material nur durch aufwendige Lagerbedingungen vor Qualitätsverlust und Zerstörung geschützt werden kann [WB93] [NF07], bietet die digitale Aufbewahrung eine einfache Form der verlustlosen Lagerung. Aber auch im digitalen Filmarchiv muss, um die Qualität des Filmmaterials zu sichern, mit Bedacht vorgegangen werden, etwa bei der Art der Digitalisierung, bei der Wahl des Codierungsverfahrens oder beim Systementwurf des digitalen Filmarchivs. Eine parallele Aufbewahrung der Originale ist natürlich unerlässlich.

Jedoch ist nicht allein die Speicherung der Filme von Interesse. Im digitalen Filmarchiv werden hohe Ansprüche an die Zugänglichkeit und Reproduzierbarkeit des Materials gestellt. Stichwort „Digitales Kino“: Die Übermittlung der digitalen Kopie eines Kinofilms aus einem Filmarchiv an die Kinobetreiber ist nur eine der vielen denkbaren Distributionsvarianten des archivierten Filmmaterials. Dies ist ein großer Fortschritt im Gegensatz zu traditionellen Filmarchiven, in denen die Verfügbarkeit des Materials stark eingeschränkt ist. Weitere Aufgabenfelder, wie die digitale Videobearbeitung oder die Verknüpfung mit Metadaten, verdeutlichen den zusätzlichen Nutzen und die Komplexität eines digitalen Filmarchivs.

1.1. Motivation

Vor allem die Zugänglichkeit des digitalen Filmarchivs gibt den Anstoß für diese Arbeit. Sämtliche Tätigkeiten, bei denen Personen auf das digitale Filmarchiv zugreifen, werden über das User Interface des digitalen Filmarchivs abgewickelt. Ziel ist es, eine optimale Architektur für die Umsetzung eines solchen User Interfaces zu schaffen. Dabei muss sich die Architektur verschiedenen Herausforderungen stellen: Je mehr Aufgabenbereiche das digitale Filmarchiv umfasst und je vielfältiger diese sind, desto größer wird die Notwendigkeit eines einheitlichen User Interfaces, welches gleichzeitig den vollen Funktionsumfang der Anwendung unterstützt. Dazu kommt, dass den vielfältigen Möglichkeiten eines digitalen Filmarchivs eine breite, ebenfalls heterogene Gruppe an Nutzern gegenübersteht [Sto08]. Auf diese Probleme soll in dieser Arbeit eingegangen und eine Lösung gefunden werden.

1.2. Projektrahmen

Die Entwicklungen in dieser Arbeit finden im Rahmen des EDCine-Projektes der Europäischen Union statt. Mit Projektpartnern aus Wirtschaft und Wissenschaft werden im EDCine-Projekt für das europäische Umfeld speziell im Bereich „Digitales Kino“ die Entwicklung neuer Technologien sowie die Anpassung bestehender Standards und Richtlinien gefördert [Mic08].

Das Fraunhofer IIS entwickelt für das EDCine-Projekt ein System zur digitalen Archivierung von Filmmaterial [NFNS07], welches im folgenden als das Softwaresystem „Digitales Filmarchiv“ bezeichnet wird. Verschiedene Komponenten des Systems werden erarbeitet, die grundlegende Systemarchitektur ist bereits modelliert und elementare Workflows sind festgelegt [EDC07b] [EDC07a]. Ein Prototyp, bestehend aus einem digitalen Archivsystem sowie einem exemplarischen User Interface, ist bereits umgesetzt. Weiterhin wurde auf Basis dieses Prototyps eine Usability-Studie durchgeführt. [Sto08]

1.3. Ziele der Arbeit

Ziel dieser Arbeit ist die Entwicklung einer Softwarearchitektur für das User Interface des „Digitalen Filmarchivs“. Die User-Interface-Architektur muss sich modular und generisch in die Systemarchitektur des „Digitalen Filmarchivs“ einfügen und die Anwendungskomponenten des Archivsystems unterstützen. Weitere Anforderungen an das User Interface des „Digitalen Filmarchivs“ sollen spezifiziert und im Architekturentwurf beachtet werden.

Die bereits im Projektrahmen entwickelte konkrete Ausgestaltung des User Interfaces, beispielsweise in Form von Workflows, Dialogmasken und Usability-Guidelines, wird ebenfalls zum Gegenstand der Anforderungen, soll jedoch nicht weiterentwickelt werden. Eine Implementierung des User Interfaces ist im Rahmen der Architekturentwicklung notwendig, soll aber ebenfalls nur beispielhaft umgesetzt werden. Eine vollständige Implementierung ist nicht Aufgabe des Architekturdesigns.

1.4. Vorgehen

Zunächst soll ein Blick auf die Grundlagen der Softwarearchitektur und der User Interfaces geworfen werden. Da das zu entwickelnde User Interface Teil des Softwaresystems „Digitales Filmarchiv“ ist, beziehen sich die weiteren grundlegende Betrachtungen auf dessen Eigenschaften. Danach folgt die Analyse der Anforderungen an das User Interface. Die daraus abgeleitete Definition der Anforderungen bildet dann die Grundlage für den Architekturentwurf. Bevor jedoch zum konkreten Entwurf geschritten wird, sind Recherchen hinsichtlich verschiedener Architekturstile und Architekturmuster notwendig, weiterhin sollen relevante Referenzarchitekturen den Anstoß für einen passenden Architekturentwurf geben. Nach der Konzeption und Entwicklung eines Architekturentwurfs für das User Interface des „Digitalen Filmarchivs“ folgt die Dokumentation, die Bewertung und eine exemplarische Implementierung der Architektur.

Dieses Vorgehen orientiert sich am Rational Unified Process und wird in Punkt 2.1.1 genauer beschrieben. Für das Vorgehen folgt aus diesem Modell, dass die verschiedenen Abschnitte auch mindestens einer Iteration unterzogen werden.

Kapitel 2.

Grundlagen

2.1. Softwarearchitektur

Spricht man von Architektur, springt der Gedanke schnell zu markanten Bauwerken, wie kunstvoll gestalteten Häusern, außergewöhnlich langen Brücken oder Bauwerken im Stil einer vergangenen Zeitepoche. Es wird schnell deutlich, was von Architektur allgemein erwartet wird. Sie soll etwas angemessen Ausdrucksvolles, meist etwas Angenehmes im Auge des Betrachters bzw. des Nutzers darstellen, dabei den Raum für die gewünschten Funktionen und Eigenschaften des Bauwerks bieten sowie auch optimieren und sich weiter den zeitgemäßen Ansprüchen stellen. Schon im frühen Verständnis von Architektur [VM60] wird klar, welche vielseitigen Einflüsse auf eine Architektur wirken. Diese können praktische Erfahrungen sowie theoretische Kenntnisse verschiedenster Wissenschaften zusammen mit der Kunst und gesellschaftlichen, wirtschaftlichen oder auch rechtlichen Gegebenheiten sein.

In der Softwarearchitektur ist dies ähnlich. Unter Beachtung einer Vielzahl von Technologien, Bedingungen und Konzepten wird ein Grundgerüst für das Softwareprodukt geschaffen. Dieser Plan setzt für alle weiteren Prozesse in der Softwareentwicklung bestimmte Vorgaben und Richtlinien und hilft auf diese Weise bei der Erfüllung von Erwartungen gegenüber dem „Gesamtwerk“. Was unter diesem Grundgerüst genauer zu verstehen ist, verdeutlicht das folgende Zitat:

> „Die Software-Architektur ist die grundlegende Organisation eines Systems, dargestellt durch dessen Komponenten, deren Beziehungen zueinander und zur Umgebung, sowie die Prinzipien, die den Entwurf und die Evolution des Systems bestimmen“ [RH06, S. 1] nach [IEE00]

Eine Architekturbeschreibung umfasst auch die Funktion und das Verhalten sowie die extern sichtbaren Eigenschaften der Komponenten. Beziehungen zwischen den Komponenten werden dabei als Schnittstellen definiert. Die Beschreibung der Organisationsstruktur findet nur auf den obersten Abstraktionsebenen statt [PBG07, S. 9]. Somit wird der Architekturentwurf vom Feindesign des Softwareprodukts abgegrenzt. Die verschiedenen Ebenen der Abstraktion in einer Architekturbeschreibung werden durch verschiedene Architekturbausteine wie Klassenstrukturen, Frameworks, Pakete oder Subsysteme verkörpert [RH06, S. 1]. Im System „Digitales Filmarchiv“ bildet das User Interface ein Subsystem, welches wiederum mehrere Komponenten vereint. Zentrale Betrachtungsebene dieser Arbeit ist das Subsystem „User Interface“, die Organisation der Komponenten innerhalb des Subsystems sowie die Beziehungen des Subsystems zum übergeordneten System „Digitales Filmarchiv“.

2.1.1. Einordnung in den Softwareentwicklungsprozess

Im Softwareentwicklungsprozess steht der Entwurf einer Softwarearchitektur an zweiter Stelle. Zuvor müssen in der Planungsphase Anforderungen für das Softwaresystem definiert und analysiert werden, da diese Anforderungen die Rahmenbedingungen für den Architekturentwurf liefern. Nach dem Architekturentwurf beginnt die Implementierung, anschließend folgen Tests der Software und abschließend die Inbetriebnahme. Weitere Aufgaben des Architekturdesigns, wie die Bewertung, die Dokumentation und die Kommunikation der Architektur, kommen begleitend hinzu. [PBG07]

Der Entwicklungsprozess beschreibt einen stufenweisen zeitlichen Ablauf, welcher tatsächlich inkrementell iterativ durchlaufen wird [RH06, S. 66 ff.]. So können zum Beispiel bei Softwaretests Anforderungen überarbeitet werden und sich damit die Rahmenbedingungen für die Architektur verschieben. Damit muss der Architekturentwurf überdacht werden, eventuell werden somit Änderungen in der Implementation notwendig und es finden erneut Tests statt. Ein ausgereiftes Softwareprodukt sollte mit der finalen Inbetriebnahme erreicht sein. Dieses Vorgehensmodell wird im sogenannten Rational Unified Process (RUP, vgl. [Kru01], siehe Abb. 2.1) beschrieben, welcher die Grundlage für den Zeit- und Projektplan dieser Arbeit liefert (siehe Punkt 1.4).

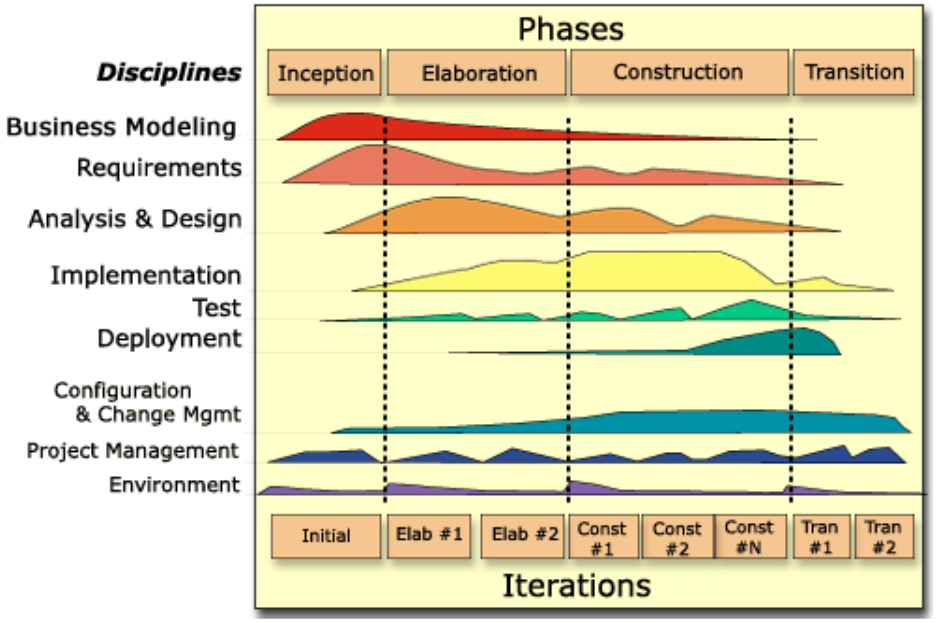

Abbildung 2.1.: RUP-Modell nach [Kru01]

Die Entwicklung des User Interfaces ist ein Teilprozess der Entwicklung des „Digitalen Filmarchivs“ und wird somit vom übergeordneten Systementwicklungsprozess beeinflusst. Daraus begründet sich die Notwendigkeit einer Iteration der User-Interface-Entwicklung bis zur Fertigstellung des Gesamtsystems. Ein ausgereiftes User Interface ist somit auch erst mit einem ausgereiften Softwaresystem „Digitales Filmarchiv“ zu erwarten.

2.1.2. Ziele und Aufgaben

Softwarearchitektur ist elementarer Bestandteil des Softwareentwicklungsprozesses. Dort übernimmt sie wichtige Aufgaben und verfolgt Ziele, die zu einer Verbesserung der Qualität des Softwareproduktes beitragen. [RH06] und [PBG07] liefern den Anstoß für die folgenden Gedanken in diesem Abschnitt:

Generell wird durch Softwarearchitektur eine zuverlässige Erfüllung von Verhaltens-, Qualitäts- und Lebenszyklusanforderungen erst möglich. Eine Softwarearchitektur liefert bereits vor der Implementierung die Grundlage für fundierte Aussagen über die Qualitätsmerkmale und Risiken eines Systems, so dass schon in der Entwurfsphase sonst unbekannte Faktoren und Einflüsse berücksichtigt werden können. Da im Zuge dieser Arbeit keine umfassende Implementierung der Software stattfindet,

ist dies ein wichtiger Punkt. So kann bereits auf Basis der Architekturbeschreibung zuverlässig auf Eigenschaften des User Interfaces geschlossen werden und, soweit notwendig, auch ohne eine Implementierung die Architektur überarbeitet werden.

Eine Softwarearchitektur bildet außerdem ein Kommunikationsmedium für die Ermittlung und Diskussion von Anforderungen und Erwartungen zwischen verschiedenen Beteiligten des Softwareprojektes. Eine gute Dokumentation von Anforderungen und ein Architekturentwurf mit Modellen des geplanten Softwaresystems hilft Verständnis für das geplante System zu schaffen und bislang unbeachtete Aspekte zu entdecken und zu diskutieren. Auch trägt es zur frühzeitigen Erkennung von Risiken bei, Problemlösungs- bzw. Problemminimierungsstrategien können so rechtzeitig eingebracht werden.

Gerade bei großen Softwaresystemen, zu denen auch das „Digitale Filmarchiv“ und dessen User Interface zählt, ist eine gute Softwarearchitektur unerlässlich. Durch das Abstrahieren von Details und die Organisation der Systembestandteile hilft sie entscheidend bei der Beherrschung der Komplexität eines solchen Systems. Das wird zum Beispiel im Entwicklungsprozess deutlich: Entkoppelte Architekturbausteine lassen sich leicht als einzelner Bestandteil entwickeln, es ist also eine einfache Untergliederung in Teilaufgaben möglich. Weiterhin wird die Effizienz des Entwicklungsprozesses gesteigert, da Funktionen und Schnittstellen der Bausteine bereits dokumentiert und aufeinander abgestimmt sind.

Weitere allgemeine Ziele und Aufgaben sind die Erhöhung der Wiederverwendbarkeit durch klar abgegrenzte Strukturen zwischen gut dokumentierten Architekturbausteinen. Gerade die Bestandteile eines generischen User Interfaces, wie es für das „Digitale Filmarchiv“ geplant ist, lassen sich auch in anderen Bereichen der User-Interface-Entwicklung einsetzen. Eine gute Dokumentation hilft auch bei der Konservierung und Speicherung des Wissens, welches später für die Wartung und Weiterentwicklung der Software relevant ist.

2.2. User Interface

Das User Interface, auch Benutzerschnittstelle genannt, ist für alle Interaktionen zwischen dem Benutzer und dem Softwaresystem zuständig. Es ermöglicht den Informationsaustausch und steuert die Kommunikation zwischen Nutzer und System, zwischen Ein- und Ausgabegeräten und den Anwendungsschnittstellen. Auf der Nutzerseite macht das User Interface sichtbar „was der Benutzer tun kann und wie das System darauf reagiert“ [Ras01, S.18]. Ein User Interface soll auch die Kommunikationsqualität zwischen Benutzer und System optimieren und damit den Nutzer beim Erreichen seiner Ziele unterstützen.

2.2.1. Accessibility

Accessibility (vgl. [ETS02] [ETS03]), auch Barrierefreiheit genannt, bezieht sich auf die Möglichkeit des Nutzers, trotz technischer oder körperlicher Einschränkungen Zugang zur Bedienung und Nutzung des User Interfaces zu haben. User Interfaces sollten also zum einen behindertengerecht, also auch zugänglich für blinde oder sehbehinderte, sowie schwerhörige oder gehörlose Nutzer sein, zum anderen aber auch ein möglichst breites Feld an technischen Einschränkungen berücksichtigen und eine Nutzbarkeit für verschiedene Hard- und Softwarekonfigurationen ermöglichen. Auch Sprachbarrieren können Nutzer von der Benutzung eines Systems ausschließen und sind damit ebenfalls zu bedenken. Allgemeingültige Richtlinien zur Barrierefreiheit werden vom W3C unter dem Namen „Web Content Accessibility Guidelines“ (WCAG) veröffentlicht [CCRV08], in Deutschland gibt es für Körperschaften des öffentlichen Rechts eine gesetzliche Verordnung zu Barrierefreiheit [Bun02], welche weitere Anhaltspunkte für eine barrierefreie Gestaltung von grafischen Benutzeroberflächen bietet.

2.2.2. Usability

Ein markantes Merkmal von User Interfaces ist die Usability (vgl. [ETS02] [BJ01] [Dix04, S. 261 ff.]). Sie beschreibt den Grad, in dem das User Interface die Psychologie und Physiologie des Anwenders berücksichtigt und in die Gestaltung einbezieht. Dabei wird u.a. der Aufwand bemessen, den der Nutzer benötigt, um seine Ziele über

die Benutzerschnittstelle zu erreichen bzw. den Umgang mit dem User Interface zu erlernen. Eine hohe Usability steht für ein intuitiv bedienbares und leicht erlernbares User Interface, dass dem Nutzer einen hohen Grad an Befriedigung seiner Bedürfnisse verschafft. Eine ausführliche Studie zur Usability des User Interfaces für das „Digitale Filmarchiv“ wird in [Sto08] durchgeführt.

2.2.3. User Interface-Arten

Ein System benötigt Ein- und Ausgabegeräte, um mit einem Menschen kommunizieren zu können. Je nach Art der Interaktion zwischen System und Benutzer variiert der Grad der Eignung bestimmter Geräte (vgl. [ETS02] [ETS03] [Dix04]). Auch die Präferenzen und Möglichkeiten eines Benutzers beeinflussen die Wahl der Ein- und Ausgabegeräte. So werden zum Beispiel blinde Menschen an Stelle eines Bildschirms einen Lautsprecher als Ausgabegerät verwenden. Benutzer, deren Ziel es ist, gewohnte Arbeitsschritte möglichst schnell umzusetzen, möchten möglicherweise auf eine Maus als Eingabegerät verzichten und Anfragen an das System nur über die Tastatur stellen. Ein mobiles Gerät hat üblicherweise einen kleineren Bildschirm und gegenüber einem Desktoprechner begrenzte Eingabemöglichkeiten, eventuell wird es auch über ein Touchscreen gesteuert. Gestaltungs-, Accessibility- und Usabilityfaktoren, sowie die Verschiedenartigkeit der Ein- und Ausgabegeräte und der Nutzertypen führen also zu verschiedenen Typen von User Interfaces. Einige für das „Digitale Filmarchiv“ relevante User-Interface-Arten, auch Dialogmodalitäten [Chl06, S. 40] genannt, werden im Folgenden genauer betrachtet.

Grafische Oberflächen

Die Mehrzahl kommerzieller Benutzeroberflächen sind heute den grafischen Oberflächen (GUI, Graphical User Interfaces, vgl. [RL04]) zugehörig [Chl06, S. 40]. Für ihre Bedienung benötigt man ein Zeigegerät (Maus, Touchpad, Touchscreen), üblicherweise wird dieses von einer Tastatur unterstützt. Die grafische Ausgabe erfolgt über einen Bildschirm oder ähnliche, grafische Ausgabegeräte. Informationen sind bei der grafischen Benutzeroberfläche zwei- oder dreidimensional dargestellt. Werden an Stelle von grafischen Elementen nur Textelemente zweidimensional angeordnet, nennt man diese Art der Benutzerschnittstelle Text User Interface. Verbreitete Kategorien grafischer User Interfaces sind beispielsweise WIMP (Windows, Icons, Menus,

Pointers)-Interfaces oder Point-and-click-Interfaces [Dix04, S. 141 ff.]. Das User Interface des „Digitalen Filmarchivs“ soll in erster Linie als GUI umgesetzt werden.

Kommandozeilen-basierte Interfaces

Kommandozeilen-basierte Interfaces (CLI, Command Line Interfaces, vgl. [RL04] [Dix04, S. 137]) sind eindimensionale, textbasierte Benutzerschnittstellen, bei denen der Nutzer Befehle mit Parametern über eine Tastatur eingeben kann und eine Antwort in Textform bekommt, die in der Regel grafisch ausgegeben wird. Alle Funktionen des Systems können so direkt genutzt werden. Sie werden meistens von versierten, der Befehlssprache des Systems kundigen Nutzern angewandt. Für das „Digitale Filmarchiv“ ist ein CLI als Alternative zur GUI sinnvoll. Fachleute, wie beispielsweise Administratoren oder im Umgang mit dem System geschulte Archivmitarbeiter, können so in bestimmten Bereichen schneller zum Ziel gelangen. Auch eine Automatisierung, beispielsweise von Ingest- oder Abfragevorgängen, würde damit möglich sein.

Voice User Interfaces

Bei Voice User Interfaces (VUI) werden die Informationen akustisch ein- und ausgegeben, die Interaktion erfolgt im Dialog zwischen Nutzer und System. VUIs lassen sich durch die Kombination einer Spracherkennung mit einem CLI umsetzen [Chl06, S.41]. Das Voice User Interface ist vor allem im Sinn der Accessibility für Menschen mit technischen oder körperlichen Barrieren eine große Hilfe [ETS03]. Diese sollten natürlich nicht vom User Interface des „Digitalen Filmarchivs“ ausgeschlossen werden, womit die Notwendigkeit eines solchen Interfaces begründet ist.

2.2.4. Benutzertypen

Die heterogene Gruppe der Nutzer eines Multi-User-Softwaresystems lässt sich, betrachtet man verschiedene Benutzermerkmale, in mehrere Nutzergruppen einteilen. Eine Nutzergruppe fasst Nutzer zusammen, welche in Bezug auf ein oder mehrere Merkmale übereinstimmende Eigenschaften besitzen. Nutzergruppen vereinfachen

die Organisation der Nutzer, beispielsweise müssen Benutzermerkmale beim Anlegen neuer Nutzer so nicht einzeln zugewiesen werden. Üblicherweise beziehen sich die Benutzermerkmale bei Software auf bestimmte Zugriffsrechte, eine Einteilung in Gruppen erfolgt meist nach dem Autorisierungsstatus in drei Typen:

Öffentliche Nutzer: Sie werden auch Gastnutzer genannt, haben keinen personalisierten Account und sind entweder unangemeldet oder mit einem Gastzugang im System tätig. Dabei haben sie stark eingeschränkte Rechte.

Registrierte Nutzer: Sie authentifizieren sich über einen Nutzernamen und ein Passwort im System und können danach personalisierte Dienste mit eingeschränkten Rechten nutzen.

Administrative Nutzer: Sie müssen sich ebenfalls am System anmelden und haben danach Zugriff auf Funktionen der Systemverwaltung.

Für das „Digitale Filmarchiv“ wird in [Sto08] für die ersten beiden Typen eine feingliedrigere Einteilung vorgenommen. Die administrativen Nutzer lassen sich ebenfalls weiter untergliedern, für das „Digitale Filmarchiv“ sind hier Designer, Entwickler und Archiv- sowie Anwendungsadministratoren denkbar (siehe auch Abb. A.4 im Anhang A.2 dieser Arbeit).

2.3. Softwaresystem „Digitales Filmarchiv“

Das User Interface, für welches im Rahmen dieser Diplomarbeit eine Architektur entwickelt werden soll, ebnet dem Nutzer den Zugang zum „Digitalen Filmarchiv“. Das „Digitale Filmarchiv“ lässt sich als ein verteiltes Softwaresystem für die Langzeitarchivierung von Video- und Metadatencontent beschreiben. Es bietet seinen Nutzern sowohl grundlegende Archivfunktionen, wie etwa das Hinzufügen, Bearbeiten oder Entfernen von Archivmaterial, das Datenmanagement und die Archivadministration, als auch erweiterte Funktionen, welche speziell für die Archivierung von Filmmaterial und Metadaten interessant sind. Die erweiterten Funktionen sind beispielsweise verknüpft mit der Videocodierung, beispielsweise das Hinzufügen oder Umwandeln

von Archivpaketen, mit den Möglichkeiten der Videobearbeitung, der Darstellung von Metadaten oder der Validierung des Archivmaterials. [EDC07b]

2.3.1. Modelle

Das „Digitale Filmarchiv“ orientiert sich am OAIS- Referenzmodell (Open Archival Information System Reference Model) [Con02] (siehe Abb. 2.2) für digitale Archivsysteme. Dieser ISO-Standard beschreibt ein umfassendes Modell für die Langzeitarchivierung digitaler Informationsobjekte. Die Informationsobjekte werden dabei in SIPs (Submission Information Packages), AIPs (Archival Information Packages) und DIPs (Dissemination Information Packages) gegliedert. SIPs werden über eine Ingest-Prozedur in das Archiv aufgenommen, als AIPs im Archiv gespeichert und dem Nutzer als DIPs zugänglich gemacht.

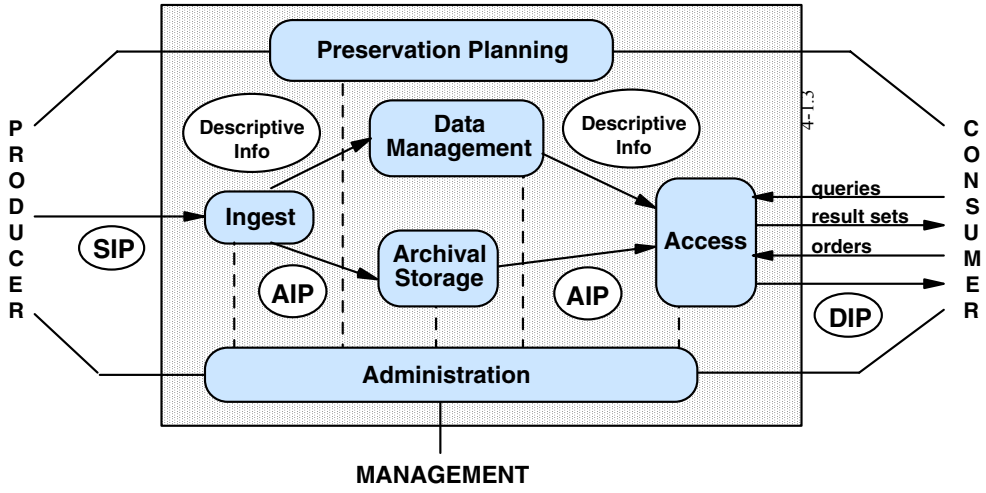

Abbildung 2.2.: OAIS Funktionsübersicht [Con02]

Das EDCine Videoarchiv teilt die AIPs nochmals in zwei Sorten auf. MAPs (Master Archive Packages) enthalten das verlustlos komprimierte Filmmaterial für die Langzeitaufbewahrung. IAPs (Intermediate Archive Packages) dagegen stellen das verlustbehaftet komprimierte Filmmaterial zur Onlineaufbewahrung bereit. So wird ein schneller und unkomplizierter Zugang zum Filmmaterial gewährleistet, während die Qualität des Archivmaterials gesichert bleibt. Bei der Wahl des Aufbewahrungsformates werden die Vorteile offener Standards und der Skalierbarkeit der Videocodierung sowie der Erweiterbarkeit der Metadatenstruktur hervorgehoben. Die Video-

codierung stützt sich daher auf die JPEG2000-Kompression, Metadaten werden im XML-Format aufbewahrt. Videomaterial und Metadaten werden im Paketformat MXF als Archivpaket hinterlegt. So wird sichergestellt, dass zusammengehöriges Material auch zusammen aufbewahrt wird. [EDC07b] [EDC07a]

2.3.2. Systemarchitektur

Die Systemarchitektur des „Digitalen Filmarchivs“ (siehe Abb. 2.3) lässt sich als dreischichtige (3-tier) Client-Server-Architektur beschreiben. Die Präsentationsschicht („Presentation tier“) enthält das User Interface, in der Anwendungsschicht („Application tier“) sind die Anwendungskomponenten des „Digitalen Filmarchivs“ versammelt, die Datenschicht („Data tier“) dient der Speicherung von Daten. Weitere Informationen zu dem zugehörigen Architekturstil „Layers“ sind im Abschnitt 4.2.2 zu finden.

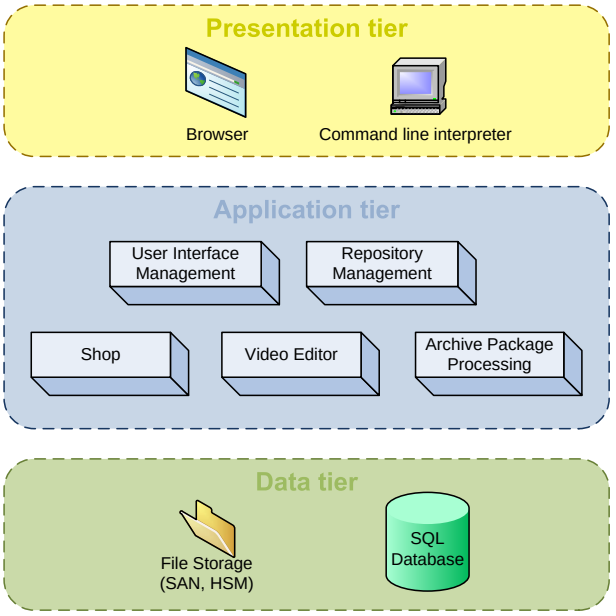

Abbildung 2.3.: Dreischichtige Systemarchitektur des „Digitalen Filmarchivs“

Den zentralen Teil des „Digitalen Filmarchivs“ stellt das eigentliche Archivsystem dar. Es erfüllt die beiden Hauptfunktionen der Aufbewahrung des Archivmaterials in der Datenschicht und der Verwaltung („Repository Management“) des Archivmaterials. Als Beispiel für ein digitales Archivsystem lässt sich die frei verfügbare Open-Source-Software „DSpace“ [DSp] nennen. Für das „Digitale Filmarchiv“ kommen nun

noch weitere Anwendungskomponenten in Betracht. Diese zeigen sich hauptsächlich für Vorgänge der Verwaltung und Bearbeitung der Archivpakete und des enthaltenen Video- und Metadateninhalts verantwortlich. Hier ist die Umwandlung der Videoformate zwischen den einzelnen Pakettypen zu nennen, weitere Beispiele sind die automatische Metadatenextraktion und die Validierung der Paketformate.

Zusätzliche Features, umgesetzt als Anwendungskomponenten im Softwaresystem „Digitales Filmarchiv“, sind denkbar. Für den Zugang der Nutzer zu all diesen Anwendungskomponenten mit ihren Funktionen tritt nun das User Interface hervor, welches im weiteren Verlauf der Arbeit ausführlich behandelt wird. Zusätzlich stellt sich noch die Frage, wie die Anwendungskomponenten des Systems miteinander kommunizieren. Hier sind als Interface vor allem SOAP-basierte Webservices angedacht, aber auch der direkte Zugriff auf APIs der Anwendungskomponenten ist möglich.

Kapitel 3.

Anforderungen an das User Interface

Die Anforderungen des Subsystems „User Interface“ liefern die Grundlage für die Architekturentwicklung und müssen daher eingehend analysiert und detailliert spezifiziert werden. Anforderungen beschreiben die Bedingungen und Fähigkeiten des zu entwickelnden Systems, welche zur Erfüllung der Wünsche und Ziele der Nutzer sowie zur Umsetzung weiterer benötigter Eigenschaften beitragen [Poh07, S. 14] [IEE00, S. 172].

Anforderungen lassen sich für verschiedene Kontextaspekte gewinnen [PBG07, S. 63], im folgenden ein kurzer Überblick über die relevanten Blickwinkel auf das geplante User Interface und deren Aspekte.

Die Anforderungen an das User Interface können unter anderem aus Sicht am Projekt beteiligter Personen beschrieben, in Dokumenten nachgelesen oder in existierenden Systemen widergespiegelt werden. Im Digitalen Filmarchiv kommen folgende Quellen in Frage:

- Stakeholder
  - Projektleiter
  - Archivnutzer und -betreiber
  - Experten
  - Entwickler
- Dokumente
  - Beschreibungen des geplanten Gesamtsystems

  - bereits spezifizierte Anforderungen
  - User-Interface- und Usability-Konzept
  - Evaluation des Prototypen
  - OAIS-Referenzmodell
- existierende Systeme
  - Prototyp
  - Referenzsoftware

Auch aus den Betrachtungsgegenständen des Digitalen Filmarchivs lassen sich Anforderungen ableiten. Einige Beispiele für Betrachtungsgegenstände sind:

- Nutzerprofile
- Filmbeschreibungen
- Archivpakete
- UI-Elemente
- Bestellungen
- Empfehlungen

Die Eigenschaften der Betrachtungsgegenstände und die Beziehungen zwischen den Betrachtungsgenständen geben weitere Hinweise auf mögliche Anforderungen.

- im Nutzerprofil gespeicherte Empfehlungen
- das UI-Element für Filmbeschreibungen
- die Bestellung eines Archivpaketes

Nicht alle Faktoren der geplanten Software sind vorhersehbar. Speziell in diesem Projekt sind vor allem organisatorische und technologische Faktoren variabel, auch ist später eine Erweiterung der funktionalen Anforderungen wahrscheinlich. Somit müssen bei der Spezifikation der Anforderungen auch unbekannte Größen bedacht werden [RH06, S. 1].

Da es sich bei dem User Interface um ein Subsystem des Systems „Digitales Videoarchiv“ handelt, lassen sich eine Reihe von Anforderungen aus den Dokumenten der übergeordneten Systeminstanz ableiten. Im Gegensatz können Anforderungen des Subsystems User Interface auch Auswirkungen auf die Anforderungen des Gesamtsystems haben. Ein weiterer Teil der Anforderungen betrifft nur das User Interface und hat keine Auswirkungen auf die Anforderungen an das Gesamtsystem.

Die Anforderungen werden in einem iterativ inkrementellen Prozess zusammen mit weiteren Aufgaben der Softwareentwicklung stufenweise erstellt und verfeinert (siehe Punkt 2.1.1). Es werden die Anforderungen an das User Interface formuliert und danach mit den Stakeholdern diskutiert. Zusätzlich fließen Erkenntnisse aus dem Architekturentwicklungs- und Implementierungsprozess ein und fügen gleichfalls neue Anforderungen hinzu oder verfeinern die Bestehenden.

3.1. Ist-Analyse

Die Ist-Analyse steht am Anfang der Anforderungsanalyse. Die in der Ist-Analyse ermittelten Daten stellen die Randbedingung für die Anforderungsanalyse dar. Nach [Poh07, S.26] werden zwei Formen der Ist-Analyse unterschieden:

Eine Form der Ist-Analyse, die sogenannte strukturierte Ist-Analyse, kann auf Basis eines Prototypen, der den implementierten Stand des Gesamtsystems zu Beginn der Anforderungsanalyse darstellt, durchgeführt werden. Da nicht alle bereits spezifizierten Anforderungen auch im Prototypen implementiert sind, außerdem die Gefahr der Vorgabe von Entwurfsentscheidungen besteht, ist dieser Weg jedoch eher ungeeignet.

Einen weiteren Ausgangspunkt für die Ist-Analyse bietet der bereits spezifizierte Stand der Anforderungen. Dieser essenzielle Ansatz der Systemanalyse betrachtet die tatsächlichen Anforderungen unabhängig von der technischen Implementation und liefert auf diese Weise eine stabilere Grundlage für Entwurfsentscheidungen. Hierbei spielt zunächst das fachliche Modell eine Rolle. In diesem Modell sollen alle vom System geforderten Anwendungsfälle durch Interaktionen umgesetzt werden können. [PBG07, S. 63]. Mit den bereits vorhanden System- und User-Interface-Spezifikationen [Sto08] [EDC07b] [EDC07a] ist bereits ein fachliches Modell gegeben. Dieses Modell wird im Folgenden als Ist-Modell bezeichnet.

Zusammen mit allen weiteren Informationen zum geplanten System gelingt nun die Entwicklung des Soll-Modells. Die Änderungsdefinition von Ist- zu Soll-Modell stellen die noch zu spezifizierenden Anforderungen dar.

3.2. Ziele

Das oberste Ziel des Entwicklungsvorhabens, die sogenannte Vision, beschreibt das angestrebte Entwicklungsprodukt unabhängig vom Weg der Umsetzung. So kann als Vision für den Architekturentwurf ein konsistentes, nutzerfreundliches und skalierbares User Interface für das „Digitale Filmarchiv“ definiert werden. Verfeinert wird diese Vision durch Ziele. Tabelle 3.1 listet die mit dem User Interface des Digitalen Filmarchivs verbundenen Ziele Z_x auf.

Z_1	Erweiterbarkeit, Modularität	Das User Interface soll leicht erweiterbar sein. Komponenten sollen dynamisch eingebunden werden können.
Z_2	Konfigurierbarkeit	Das User Interface soll Konfigurationsmöglichkeiten für die Anwendung und deren Darstellung bieten.
Z_3	Dialogmodalität	Das User Interface soll verschiedene Interface-Typen unterstützen. Der zentrale User-Interface-Typ soll eine GUI sein, als Alternative ist auch ein CLI vorgesehen. Möglichkeiten zur Umsetzung eines VUI sollen ebenfalls vorhanden sein.
Z_4	Bedienbarkeit	Das User Interface soll nutzerfreundlich gestaltet sein. Es soll möglich sein, die Ziele des Usability-Konzeptes aus [Sto08] umzusetzen. Weiterhin soll das User Interface einem größtmöglichen Nutzerkreis zugänglich sein.
Z_5	Nutzertypen	Das User Interface soll verschiedene Nutzertypen entsprechend [Sto08] abgrenzen können.
Z_6	Archivfunktionen	Das User Interface soll einen Zugang zu den Funktionen des „Digitalen Filmarchivs“ bieten. Die Aufgabenbewältigung soll gemäß den spezifizierten Workflows [Sto08] [EDC07b] [EDC07a] möglich sein. Auch die Funktionen erweiterter Komponenten, wie etwa der Administration und Konfiguration oder der Validierung von Archivpaketen, sollen über das User Interface zugänglich sein.
Z_8	Mehrsprachigkeit	Das User Interface soll Mehrsprachigkeit unterstützen.

Tabelle 3.1.: Zieldefinitionen des User Interfaces

Für Ziele, die im Konflikt zueinander stehen, müssen Lösungen gefunden oder Prioritäten gesetzt werden. Die größten Risiken sollten dabei zuerst bearbeitet werden.

[RH06, S.67] Markant sind beispielsweise die Punkte „Bedienbarkeit“ und „Erweiterbarkeit“. Ein komplexes System mit großem Funktionsumfang führt schnell zu einem unübersichtlichen User Interface. Hier müssen Anforderungen formuliert werden, die sicherstellen, dass beides zuverlässig umgesetzt werden kann.

3.3. Anwendungsszenarien

Nach der Beschreibung der Ziele empfiehlt es sich nun, eine Reihe von Anwendungsszenarien (Use-Cases UC-X) aufzustellen. Aufgrund der Komplexität und der Erweiterbarkeit des User Interfaces beschränken sich die Anwendungsszenarien jedoch auf ein paar Beispielfälle. Sie beruhen auf den bereits vorhandenen Use-Cases (vgl. [Sto08] [EDC07b] [EDC07a]) und den erweiterten Use-Case-Diagrammen im Anhang Punkt A.2. Ein Use-Case wird in Tabelle 3.2 exemplarisch dargestellt, alle weiteren Use-Cases befinden sich im Anhang Punkt A.1.

UC-1:	Abrufen der Item-Informationen
Vorbedingung:	-
Nachbedingung:	-
Auslösendes Ereignis:	Auswahl eines Items
Paket:	Repository Management
Beschreibung:	Eine Übersicht mit Informationen über das Videoitem wird angezeigt. In einem Videoplayer kann das Video im Preview-Format abgespielt werden. Weiterhin können Zusatzinformationen eingesehen werden.
Erweiterung:	Ein externer Nutzer kann zudem das Item in den Warenkorb legen und somit den Bestellvorgang starten. Dem internen Archivmitarbeiter wird sowohl eine Bearbeitungsfunktion als auch eine Exportfunktion für das Item bereitgestellt. Die Bearbeitung bezieht sich sowohl auf Metadaten als auch auf den Videoinhalt. Weiterhin kann der interne Archivmitarbeiter Informationen zum zugehörigen DCP abrufen und von diesem Punkt weitere Funktionen zum DCP (z.B. DCP Validation) nutzen.

Tabelle 3.2.: Beispielszenario für das User Interface

3.4. Spezifizierte Anforderungen

Die endgültige Spezifikation der Anforderungen aus Vision, Zielen und Szenarien stellt nun die letzte Stufe der Anforderungsanalyse dar. Sie lässt sich im klassischen Sinn auch mit einer Übertragung des „Lastenhefts“ in das „Pflichtenheft“ vergleichen [Poh07].

Da bereits die Notwendigkeit bestimmter Architekturkomponenten und Schnittstellen sowie deren Eigenschaften bzw. Funktionen angegeben werden, erfolgt mit der Anforderungsspezifikation der erste Grobentwurf für die User-Interface-Architektur.

Spezifizierte Anforderungen helfen im späteren Entwurfsprozess, die Softwarearchitektur den Erfordernissen anzupassen und somit die Softwarequalität zu sichern. Die Definition von Softwarequalität in [DIN] nennt eine Reihe von Qualitätsmerkmalen, worunter auch die funktionalen Merkmale gezählt werden. [Poh07] trennt die Merkmale deutlicher in funktionale Anforderungen, qualitative Anforderungen sowie Rahmenbedingungen für die Softwareentwicklung. Diese werden nun im folgenden Teil beschrieben.

3.4.1. Funktionale Anforderungen

In der hier aufgeführten Tabelle 3.3 werden die funktionalen Anforderungen des User Interfaces wiedergegeben.

R_1	Das User Interface soll in verschiedene Komponenten mit modularem Charakter untergliedert werden. Eine Modulverwaltung, welche alle Module koordiniert und die Kommunikation zwischen Modulen ermöglicht, ist notwendig.
R_2	Das User Interface muss vielfältige aber konsistente Ansichtstypen und übersichtliche Menü- sowie Navigationsstrukturen zur Verfügung stellen. Ansichtstypen können neben den Standardansichten beispielsweise eine Videoplayer-Ansicht, eine Timeline-Ansicht oder eine Baumstrukturansicht für die explorative Darstellung von XML-Metadaten sein.
R_3	Das User Interface muss vielfältige aber konsistente UI-Elemente für verschiedene Anwendungsdomänen (bspw. Archivfunktionen, Videobearbeitung, Shop) bereitstellen. Diese Elemente sollen von Modulen genutzt und erweitert werden können. Die UI-Elemente sollen in einer standardisierten UI-Beschreibungssprache definiert werden. UI-Teile verschiedener Module sollen mit Hilfe bestimmter Transformationsvorschriften zu einer einheitlichen Oberfläche verbunden werden.
R_4	Das User Interface soll eine einheitliche Konfigurationsoberfläche bieten, welche zur Konfiguration von Einstellungen des User Interfaces und der User-Interface-Module genutzt werden kann.
R_5	Verschiedene Interface-Typen sollen als UI-Module für die Realisierung der Dialogmodalitäten bereitgestellt werden.
R_6	Die Anwendungsschnittstellen der User-Interface-Module sollen über WebServices und den direkten Zugriff realisiert werden.
R_7	Das User Interface soll eine Nutzerverwaltung zur Autorisierung der am System angemeldeten Nutzer vorsehen. Access-Control-Lists spannen eine Matrix über die Zugriffsberechtigungen von Nutzern auf Programmmodule auf.
R_8	Der Zugriff auf das User Interface soll in Sessions erfolgen. Während eines Zugriffs können Sessiondaten gesammelt oder verworfen werden. Das Wiederherstellen einer Session soll ebenfalls möglich sein. Auch soll geregelt werden, welche Daten einer Session beim Übergang zwischen zwischen verschiedenen Nutzerstufen beibehalten bzw. verworfen werden.

Tabelle 3.3.: Funktionale Anforderungen an das User Interface

3.4.2. Qualitätsanforderungen

Nachdem der Funktionsumfang des zu entwickelnden Softwareproduktes beschrieben wurde, widmet sich dieser Teil den qualitativen Maßstäben und den Randbedingungen, die das System erfüllen soll, sowie den Rahmenbedingungen, welche die Entwicklung begleiten. Qualitätsanforderungen bilden die Voraussetzung für eine frühzeitige Bewertung des Architekturentwurfes. Für eine Bewertung ist im Gegensatz zu den funktionalen Anforderungen meist keine vollständige Implementierung notwendig. Qualitätsanforderungen werden oft unterspezifiziert. Daher ist im Folgenden darauf zu achten, dass eine formulierte Anforderung mit erfüllbaren Szenarien oder quantitativ bewertbaren Zielen verknüpft wird, die eine rein subjektive Bewertung ausschließen [PBG07, S. 80 f.]. Die in [Bos00], [PBG07] und [DIN] beschriebenen Qualitätsanforderungen sind wegweisend für die folgenden Erläuterungen.

Benutzbarkeit

Der vom Nutzer benötigte Aufwand zur Nutzung des Systems soll möglichst gering gehalten werden. Eine einfache, intuitive Bedienung, auch für Nutzer mit unterschiedlichem Erfahrungshorizont, soll in einem variablen Verwendungskontext ermöglicht werden. Die Oberfläche soll verständlich gestaltet sein, Wege zum Ziel der Benutzung sollen einfach erlernbar sein.

Die Richtlinien zur Benutzbarkeit lassen sich in der bereits erfolgten Usability-Studie zur Benutzeroberfläche des „Digitalen Filmarchivs“ [Sto08] nachlesen. Wird die Umsetzung des dort genannten Usability-Konzeptes ermöglicht, gilt diese Anforderung als erfüllt.

Leistung

Eine gute Performance, sichtbar durch schnelle Reaktionen auf Benutzereingaben und weitere Ereignisse, beeinflusst unter anderem die Benutzbarkeit und soll deshalb in einem akzeptablen Rahmen liegen. Es muss darauf geachtet werden, dass das User Interface im Gesamtsystem nicht den Leistungsengpass darstellt und unter verschiedensten Belastungsstufen konstante und akzeptable Antwortzeiten aufweist.

Änderbarkeit

> „Änderungen sind nahezu wie ein Naturgesetz in der Softwareentwicklung“ [PBG07]

Gerade Benutzerschnittstellen sind häufigen Änderungen unterworfen, deswegen sollte der Aufwand für Änderungen im User Interface möglichst gering sein. Schon im Entwicklungsprozess der Software (siehe Punkt 2.1.1) spielt die Änderbarkeit eine große Rolle, da Anforderungen überarbeitet oder ergänzt werden müssen, was Änderungen der Software zur Folge hat. Nach der Inbetriebnahme sind häufige Änderungen sicherlich vor allem im Design des User Interfaces zu erwarten. Diese Art der Änderung soll möglichst einfach durchgeführt werden können. Es soll dabei kein Eingriff in den Quellcode notwendig sein, wenige Konfigurationsschritte in einer Konfigurationsverwaltung müssen zur gewünschten Änderung führen. Änderungen im Quellcode sowie Tests der Änderungen sollen möglich sein, ohne die Verfügbarkeit des User Interfaces zu beeinflussen oder die Stabilität der Software zu gefährden. Vorteilhaft ist es, wenn wahrscheinliche Änderungen jeweils nur einen Baustein der Architektur beeinflussen. Änderungen der Architektur sollten nach der Inbetriebnahme nicht mehr notwendig sein. Die Änderbarkeit lässt sich auch als „Wartbarkeit“ des User Interfaces definieren, oft wird sie ebenfalls als „Flexibilität“ bezeichnet.

Effizienz und Skalierbarkeit

Der Gebrauchsumfang des User Interfaces kann variieren, zum einen in der Zahl der Nutzer, zum anderen im Umfang der genutzten Funktionalitäten. Deswegen ist eine Modularität des Subsystems „User Interface“ durch Untergliederung in weitere Komponenten unumgänglich. Es kann sowohl der Funktionsumfang skaliert werden, als auch eine konstante Effizienz der Software erzielt werden, z.B. durch Clustering oder Auslagerung der Komponenten. Komponenten des User Interfaces sollten also als Plugin-Module integrierbar sein. Bei Anwendung des Client-Server-Modells ist es im Interesse der Skalierbarkeit, möglichst viele Aufgaben zur clientseitigen Verarbeitung abzugeben.

Übertragbarkeit und Interoperabilität

Es soll möglich sein, das User Interface in eine andere Hardware- oder Softwareumgebung zu portieren. Gerade weil diese technologischen Faktoren für das zu erstellende System noch nicht vollständig spezifiziert sind und später variieren können, ist auf eine Plattformunabhängigkeit und eine Vermeidung von technologischen Abhängigkeiten zu achten. Plattformabhängige Faktoren können in einer Komponente gekapselt werden, so dass eine konkrete Plattform dann eine Implementation dieser Komponente darstellt [PBG07, S.79]. Zur Verwirklichung dieses Merkmals trägt die Standardkonformität bei, denn durch die Verwendung von Standards kann in der Regel Interoperabilität und damit erhöhte Kompatibilität gewährleistet werden.

Sicherheit

Sicherheit bedeutet die Abwehr und Vorsorge potentieller Gefahren. Die größte Gefahr besteht im unautorisierten Zugriff auf Anwendungsfunktionen. Im Sinne des Daten- und Urheberrechtsschutzes muss dafür gesorgt werden, dass vor allem Nutzerdaten und das Archivmaterial nur berechtigten Personen zugänglich ist. Da eine Hauptaufgabe des Archivsystems die Bewahrung der Filme ist, und das User Interface den Zugriff auf und die Veränderung des Materials möglich macht, muss sichergestellt werden, dass auch bei einer Fehlfunktion der Software archiviertes Material nicht verloren gehen oder beschädigt werden kann.

Testbarkeit

Um Entwickler und Tester bei der Fehlererkennung und Fehlerbehebung zu unterstützen, ist es angebracht, Schnittstellen und Strukturen in die Architektur zu integrieren, welche Debug-, Fehler- und Zustandsinformationen der Anwendung über das User Interface sichtbar machen.

Zuverlässigkeit

Das User Interface soll kontinuierlich für den Nutzer verfügbar sein. Da es sich um die Benutzerschnittstelle für ein komplexes System handelt, welches unter Umstän-

den im Sinne der Skalierbarkeit auch global mehreren tausend Nutzern rund um die Uhr verfügbar gemacht werden soll, sind Ausfallzeiten oder temporäre Einschränkungen bei der Benutzung indiskutabel. Voraussetzungen für Zuverlässigkeit sind Fehlertoleranz und eine ausgeprägte Reife des Systems. Diese lässt sich durch die die Unterstützung von Fehlererkennung und Fehlerbehebung erhöhen, ist also eng mit dem Merkmal „Testbarkeit“ verknüpft. [PBG07] Sollte dennoch ein Versagen der Software eintreten, ist es wichtig, eine komplette Wiederherstellbarkeit des vorherigen Zustandes zu garantieren. Beispielsweise sollten bei einem Ausfall der Datenbank die Nutzerprofile nicht verloren gehen, sondern sofort reproduzierbar sein.

3.4.3. Rahmenbedingungen

Organisatorische Faktoren

Da die Entwicklung des Gesamtsystems noch nicht vollzogen ist und der Bedarf an Komponenten noch nicht feststeht, sollen Überlegungen zum User Interface mit Bedacht einer, gegenüber diesen Faktoren, generischen Lösung stattfinden. Auch über das Entwicklungsteam und die bevorzugte Plattform bei der Implementierung lassen sich noch keine detaillierten Aussagen treffen. Hier muss ebenfalls mit einer Umsetzung auf Basis veränderlicher Parameter gerechnet werden. Als einziger organisatorischer Faktor kann ein Meilenstein im Zeitplan genannt werden: Im ersten Quartal 2009 soll ein Demonstrator des „Digitalen Filmarchivs“ einschließlich des User Interfaces fertiggestellt sein.

Technologische Faktoren

Detaillierte Aussagen zur technischen Realisierung sind zum Zeitpunkt der Anforderungsanalyse noch nicht möglich, deswegen ist auch hier auf eine generische Ausprägung der Umsetzung gegenüber diesen Faktoren zu achten. Einige Anhaltspunkte können jedoch schon als Anforderung formuliert werden: Das User Interface soll als Weboberfläche mit Ajax-Unterstützung umgesetzt werden, auch soll die Software im Sinn der Skalierbarkeit auf verteilten Systemen betrieben werden können. Weiterhin soll eine Struktur genutzt werden, welche die Trennung von Daten, Funktionen und Darstellung ermöglicht.

Kapitel 4.

Architektur des User Interfaces

Die Anforderungsanalyse des vorangegangenen Kapitels bereitet den Weg für den nun folgenden Teil des Architekturentwurfs. Hier wird zunächst viel Wert darauf gelegt, die Herangehensweise an den Architekturentwurf zu klären und die geeigneten Mittel für die Umsetzung zu finden. Schritt für Schritt wird dabei der Entwurf erarbeitet, ausführlich beschrieben und abschließend bewertet.

4.1. Vision

Bereits zu Beginn der Anforderungsanalyse im Abschnitt 3.2 wird eine Vision formuliert, welche elementare Ansprüche an das User Interface stellt. Nun lässt sich die Vision nicht nur für das Softwareprodukt definieren, sondern weitergehend als Systemidee [PBG07, S. 66] für den Architekturentwurf konkretisieren.

Im Mittelpunkt steht also der Entwurf einer User-Interface-Architektur für die interaktiven Anwendungskomponenten des Softwaresystems „Digitales Filmarchiv“, welche bestehende Schnittstellen im Softwaresystem nutzt, eine modular erweiterbare Grundstruktur des User Interfaces unter Berücksichtigung der Ermöglichung von Usability- und Accessibility-Ansprüchen vorsieht und insgesamt den Grundsatz der Skalierbarkeit erfüllt.

4.2. Vorbetrachtungen zum Entwurf

Eine gute Architektur lässt sich nicht allein durch das bloße Zusammenstellen der Komponenten zu einem Softwareprodukt umsetzen. Da es immer eine Vielzahl an mehr oder weniger geeigneten Wegen und Lösungen für die Architektur einer Software gibt, müssen vor dem eigentlichen Entwurf ausführliche Vorbetrachtungen stattfinden, welche eine grobe Richtung vorgeben und Entwurfsentscheidungen erleichtern sollen. Mit Architekturprinzipien, -stilen, -mustern und Referenzarchitekturen werden theoretische Erkenntnisse sowie der Erfahrungsschatz von Softwarearchitekten und -entwicklern mit bestehenden Systemen in die Vorbetrachtungen eingebracht. Um die geeigneten Hilfsmittel auszuwählen, erfolgt ein Abgleich mit den Anforderungen und der Vision. Stufenweise werden die Überlegungen im Laufe der Vorbetrachtungen ausgehend von allgemeinen Prinzipien der Softwarearchitektur soweit konkretisiert, dass am Ende bereits grundlegende Entwurfsentscheidungen für die Architektur getroffen sind.

4.2.1. Architekturprinzipien

Zunächst folgt ein kurzer Überblick über allgemeine Prinzipien der Architekturentwicklung nach [PBG07, S. 102 ff.]. Alle diese Prinzipien haben die Aufgabe, komplexe Systeme in eine übersichtliche und verständliche Struktur zu bringen. In Architekturstilen, -mustern und Referenzarchitekturen werden verschiedene dieser Prinzipien angewendet.

Abstraktion

Wichtig bei der Modellierung einer Architektur ist die Festlegung einer bestimmten Abstraktionsebene, auf der die Betrachtung stattfindet. Tiefere Ebenen beinhalten detailreichere Darstellungen, Ebenen höherer Abstraktion betrachten das Ganze im Gesamtzusammenhang. Abstraktion bezieht sich auf die äußere Sicht auf den Architekturbaustein und wird angewendet, wenn die Beschreibung eines Architekturbausteins auf seine Funktion und seine Schnittstellen begrenzt oder eine Architektur in verschiedenen Modellen und Sichten betrachtet wird. Die Architektur des „Digitalen Filmarchivs“ (siehe Abschnitt 2.3.2) und dessen User Interfaces wurde beispielsweise

auf mehrere Betrachtungsebenen abstrahiert: Das Softwaresystem „Digitales Filmarchiv“ und dessen 3-Schichten-Architektur. Eine Betrachtungsebene tiefer wird die innere Architektur einer Schicht sichtbar, auf der sich auch die Komponente „User Interface Management“ befindet. Bildet diese Komponente ein Subsystem, wird eine Abstraktionsebene des nächsten Grades erschlossen.

Hierarchie

Um viele Abstraktionsebenen in eine Rangfolge zu bringen, wird die Methode der Hierarchiebildung verwendet. Eine Abstraktionsebene bildet Elemente ab, auf einer untergeordneten Abstraktionsebene werden Teile oder Spezialisierungsformen der einzelnen Elementes dargestellt. Bei komplexen Systemen wie dem „Digitalen Filmarchiv“ lassen sich mehrere Betrachtungsebenen mit unterschiedlichen Architekturgrundsätzen nicht vermeiden, daher werden diese in einer strukturellen Hierarchie angeordnet. So ist zum Beispiel das Subsystem „User Interface Management“ Teil der Anwendungsschicht, welche wiederum Teil des Softwaresystems „Digitales Filmarchiv“ ist.

Kapselung

Kapselung beschreibt das Verstecken von Details nach dem Prinzip des „Information Hiding“. Die inneren Details eines Architekturbausteins werden dabei unsichtbar. Im Softwaresystem „Digitales Filmarchiv“ wird die Implementierung der Anwendungslogik des User Interfaces in der Komponente „User Interface Mangagement“ gekapselt, die Implementierung der Präsentation des User Interfaces in der Präsentationsschicht. Durch die Organisation weiterer Komponenten in tieferen Betrachtungsebenen der Komponente „User Interface Management“ erfolgt eine Kapselung weiterer User-Interface-bezogener Implementierungsdetails.

Modularität

Modularität beschreibt die Unterteilung eines Systems in mehrere Module. Innere Elemente eines Moduls sollen dabei eng gekoppelt, die Module untereinander lose gekoppelt sein. Wichtig ist es, dabei die richtige Granularität der Module zu defi-

nieren. Ziel ist es, neue Module so zu strukturieren, dass möglichst keine weiteren Subsysteme entstehen und eine übersichtliche Zahl etwa gleichwertiger Module geschaffen wird. Das User Interface soll modular gestaltet werden. Die Kopplung wird sich dabei durch die Abgrenzung der User-Interface-Teile gegenüber verschiedenen Anwendungsbereichen und -funktionalitäten ergeben. So könnte es zum Beispiel ein Archivmodul, ein Online-Shop-Modul,ein Videobearbeitungsmodul, ein Modul für die Erzeugung eines GUI-Modells und ein Modul zur Erzeugung eines CLI-Modells geben.

Konzeptuelle Integrität

Die konzeptuelle Integrität beschreibt die durchgängige Anwendung von Entwurfsentscheidungen und sichert somit ab, dass Architekturüberlegungen und deren Auswirkungen letztendlich im System widergespiegelt werden. Auf einer Abstraktionsebene sollte damit jeweils nur ein Architekturstil oder -muster eingesetzt werden und es muss außerdem beachtet werden, dass sich die Architekturgrundsätze verschiedener Ebenen nicht vermischen. Insbesondere bei der Implementierung ist die Umsetzung der Abstraktionsebenen und der zugeordneten Architekturvorstellungen zu beachten, da sich sonst leicht ungewollte Effekte im System ergeben können.

4.2.2. Architekturstile

Betrachtet man die globale Anordnung der Komponenten eines Softwaresystems, bieten Architekturstile bereits einige strukturelle Organisationsprinzipien. Sie sind zumeist mit grundlegenden Anforderungen verknüpft und können so je nach Eignung ausgewählt werden. Um konzeptuelle Integrität zu verwirklichen, sollte man sich beim Entwurf einer Betrachtungsebene möglichst auf einen Stil beschränken. [PBG07, S. 206 f.] Im Architekturentwurf des User Interfaces werden die Stile Layers, Pipes & Filters und Objektorientierung verwendet, welche im Folgenden näher betrachtet werden.

Layers

Der Layers-Stil (vgl. [SG96, S. 25] [Bos00, S. 119 ff.]) teilt ein System in mehrere Schichten. Dabei nutzt jede Schicht die Dienste der darunterliegenden Schicht und stellt der darüberliegenden Schicht Dienste zur Verfügung. Der Datenfluss bewegt sich in vertikaler Richtung zwischen den verschiedenen Schichten.

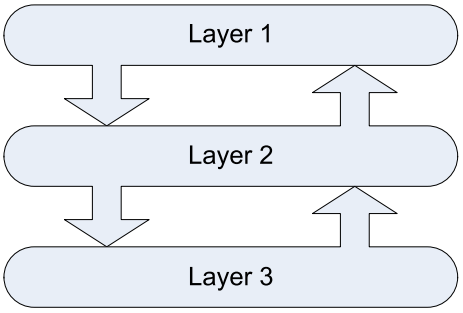

Abbildung 4.1.: Anordnung der Komponenten im Stil Layers

Diese Einteilung führt dazu, dass Änderungen meist nur eine Schicht betreffen. Einzelne Schichten können somit einfach ausgetauscht und wiederverwendet werden, da Anpassungen maximal in der jeweils oberen bzw. unteren Schicht relativ zur Bezugsschicht vorgenommen werden müssen. Die Schichtenarchitektur wirkt sich also positiv auf Änderbarkeit und Übertragbarkeit aus, der erhöhte Kommunikationsaufwand über mehrere Schichten kann allerdings zu Leistungseinbußen führen.

Das Softwaresystem „Digitales Filmarchiv“ orientiert sich mit seiner Gesamtarchitektur am Layers-Stil (siehe Punkt 2.3.2), das eigentliche User Interface befindet sich dabei in der obersten Schicht der 3-Schichten-Architektur. Es existiert jedoch auch eine „User Interface Management“-Komponente in der Anwendungsschicht, auch die Datenschicht wird genutzt, beispielsweise um Einstellungen für das User Interface zu hinterlegen. Weitere Anwendungen des Schichtenstils kommen später in tieferen Betrachtungsebenen zur Diskussion.

Pipes & Filters

Bei der Pipes & Filters-Architektur (vgl. [SG96, S. 21 f.] [Bus07, S. 31 ff.], siehe Abb. 4.2) sind die Komponenten eines Systems als Filter zu verstehen, welche einen Datenstrom aufnehmen, ihn verarbeiten und danach wieder bereitstellen. Die Filter

agieren dabei völlig unabhängig voneinander und sind lediglich über Pipes, welche den Datenstrom an den nächsten Filter weitergeben, miteinander verbunden.

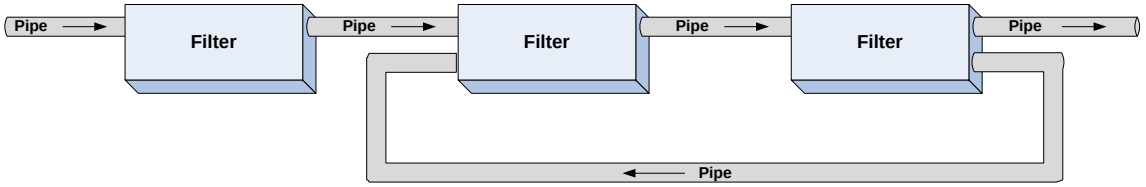

Abbildung 4.2.: Anordnung der Komponenten im Stil der Pipes & Filters

Ein großer Vorteil liegt mit der Modularisierung und Entkopplung der Komponenten in der einfachen Verteil- und Skalierbarkeit des Systems, einzelne Bausteine können problemlos verteilt, aktiviert bzw. deaktiviert, ausgetauscht oder wiederverwendet werden [PBG07, S. 208]. Weitere Vorteile ergeben sich für die Entwicklung und Fehlerbehebung, da zu Analysezwecken an jedem Baustein der In- und Output und somit die korrekte Funktionsweise geprüft werden kann. Nachteile der Pipes & Filters-Architektur liegen in der Leistung, welche durch den sequenziellen Datentransport und dem wiederholten Parsen der Daten in jedem einzelnen Filter gemindert wird. Speziell in Bezug auf interaktive Systeme existieren jedoch Nachteile dieses Architekturstils [SG96, S. 22]. Schrittweise Updates der Ausgabe sind aufgrund des Transformationscharakters der Filter, die eine vollständige Transformation der Eingabe- in Ausgabedaten durchführen, nur schwer möglich. Beziehungen zwischen Datenströmen verschiedener Pipes sind nur schwer realisierbar. Zuverlässigkeit und Sicherheit sind ebenfalls Risikofaktoren, da das schwächste Glied in der Kette die Anfälligkeit des Systems bestimmt [PBG07, S. 208].

Die genannten Vorteile stehen konform zum Anforderungsprofil des „User Interfaces“, womit dieser Stil zum Favoriten für den Architekturentwurf wird. Speziell die auf interaktive Systeme bezogenen Nachteile machen jedoch deutlich, dass ein allein auf diesem Stil beruhender Architekturentwurf für Komponenten des User Interfaces große Einschränkungen mit sich bringen würde. Daher müssen hier weitere Lösungen gefunden werden.

Objektorientierung

Im Stil der Objektorientierung (vgl. [SG96, S. 22 f.], siehe Abb. 4.3) stellen die Komponenten abstrahierte Objekte dar, in denen die weiteren Funktionen gekapselt

sind. Diese Objekte können wahllos über mehrere Schnittstellen Daten miteinander austauschen.

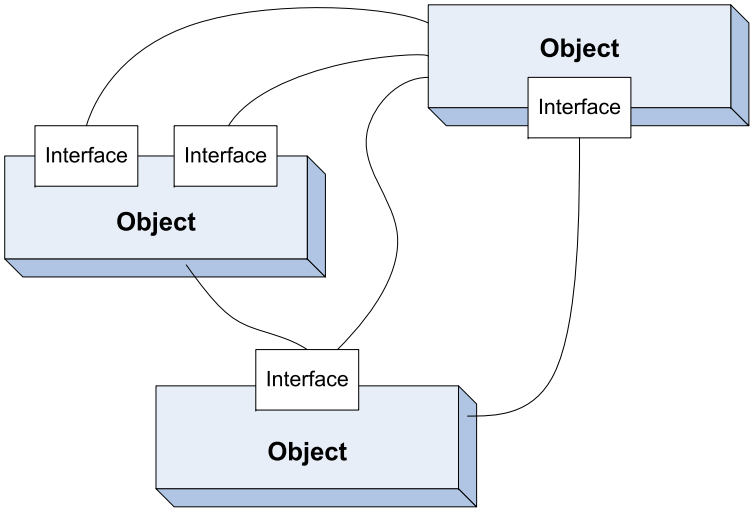

Abbildung 4.3.: Anordnung der Komponenten im Stil der Objektorientierung

Hauptproblem hierbei ist, dass die Änderung eines Objektes alle anderen Objekte beeinflussen kann. Objekte müssen also ihre gegenseitige Identität kennen. Auch kann die gleichzeitige Nutzung eines Objekts durch mehrere verknüpfte Objekte unerwünschte Nebeneffekte hervorrufen, die Fehlersuche und -behebung gestaltet sich dementsprechend ebenfalls schwieriger.

Die Objektorientierung eignet sich hervorragend, um die Nachteile bezüglich der Flexibilität des Datenflusses und der Transformationsschritte in der starren Pipes & Filters-Architektur auszugleichen. Somit ist beim Architekturentwurf eine Anwendung dieser beiden Stile in zwei übereinanderliegenden Schichten sinnvoll. Empfehlenswert wäre beispielsweise eine Anordnung der Komponenten als Objekte, in denen jeweils die Bestandteile der Pipes & Filters-Architektur gekapselt sind.

4.2.3. Architekturmuster für verteilte Systeme

Im Gegensatz zu Architekturstilen sind Architekturmuster (vgl. [Bus07]) auch global wirksam, beziehen sich jedoch auf bestimmte Funktionsbereiche des Systems. Sie bieten Lösungsmuster vor allem für die technische und strukturelle Gestaltung. Lösungsmuster haben sowohl in der Architektur, als auch in der Softwareentwicklung bereits Tradition. So wurden Muster für den Städtebau konzipiert [VM60] [AISJ05],

für Problemstellungen der objektorientierten Softwareentwicklung wurden sogenannte Entwurfsmuster („Design Patterns“) [GHJV07] geschaffen. Oft findet sich in Architekturmustern eine Kombination verschiedener, lokal wirksamer Entwurfsmuster wieder. Auf Entwurfsmuster soll jedoch hier nicht näher eingegangen werden, da sie sich mit der Lösungen von Problemen auf einer tieferen Detaillierungsebene befassen.

REST

Nachdem in den Betrachtungen der Architekturstile bereits die Anordnung von Komponenten in einem Softwaresystem geklärt wurde, wird hier hauptsächlich auf die Gestaltung der Kommunikation zwischen den Komponenten Wert gelegt. Das grafische User Interface des „Digitalen Filmarchivs“ soll in erster Linie als webbasiertes Hypermediasystem umgesetzt werden, womit sich die Anwendung der REST-Architektur empfiehlt (vgl. [RR07] [Fie00]). Definiert eine Komponente über die Schnittstelle eine Ressource (Resource), fungiert sie damit als Server und nimmt Anfragen (Requests) von anderen Komponenten (Clients) entgegen, welche diese Schnittstelle nutzen. Die Requests sind in Form einer URI an eine eindeutige Resource adressiert, deren Repräsentation als Antwort (Response) vom Server and den Client zurückgegeben wird. Die gesamte Kommunikation findet dabei auf Basis des HTTP-Protokolls statt. Die Dienste, welche der Server dem Client zur Verfügung stellt, werden auch „RESTful Webservices“ genannt. Eine Beschreibung der Dienste kann in der „Web Application Description Language“ (WADL) [Had06] erfolgen.

Die REST-Architektur bringt erhebliche Vorteile in der Skalierbarkeit mit sich, da die Kommunikation zustandslos ist und sich damit einfach auf mehrere Server verteilen lässt. Die eindeutige Adressierbarkeit der Ressourcen in eindimensionaler Form und die Repräsentation in beliebiger Form ermöglicht ebenfalls die Umsetzung eines Command Line Interfaces auf Basis der REST-Architektur.

Neben der REST-Architektur und RESTful Webservices existieren eine Reihe weiterer Webservice-Typen, zum Beispiel SOAP oder XML-RPC auf Basis einer RPC-Architektur [HL04]. Diese bieten allerdings für die Umsetzung der Anforderungen keinen Mehrwert und würden die Komplexität der Schnittstellen unnötig steigern. Auch fällt bei komplexeren Webservices ein erheblicher Kommunikationsoverhead an, welcher die Leistung des Systems merkbar beeinträchtigen würde.

Ajax

Die Idee der Ajax-Architektur (vgl. [Gar05]) bezieht sich auf die GUIs dynamischer Webanwendungen und ermöglicht ein neues Modell der Kommunikation zwischen dem Browser des Anwenders und dem Webserver über das HTTP-Protokoll. Für eine normale Webseite wird vom Webbrowser ein HTTP-Request an den Server gesendet, der daraufhin den kompletten HTML- und CSS-Code zur Anzeige der Webseite überträgt. In der Ajax-Architektur wird die auf JavaScript basierende „Ajax Engine“ in den vom Webserver übermittelten Code eingebunden und vom Browser ausgeführt. Die „Ajax Engine“ ermöglicht nun eine Kommunikation mit dem Server und eine Veränderung der angezeigten Webanwendung, ohne dass die komplette Seite vom Browser neu geladen werden muss. So können im Hintergrund dynamisch neue Daten geladen und Teile der Darstellung entsprechend verändert werden.

Dieses Modell hat große Vorteile für die Benutzbarkeit einer Webanwendung, da schrittweise Anpassungen der GUI ohne Verzögerungen durch einen Neuaufbau der Seite vorgenommen werden können. Durch das gezielte Nachladen von Daten wird ein Kommunikations- und Informationsoverhead vermieden und dadurch die Effizienz des Systems gesteigert, was sich wiederum positiv auf die Skalierbarkeit, Leistung und Usability des Systems auswirkt. Auf das webbasierte User Interface des „Digitalen Filmarchivs“ soll aus diesen Gründen eine Ajax-Architektur angewendet werden. Die dafür benötigten Technologien lassen sich problemlos in eine REST-Architektur integrieren [RR07, S. 315 ff.].

4.2.4. Architekturmuster für interaktive Systeme

In diesem Abschnitt werden Architekturmuster betrachtet, welche speziell für den Bereich User Interface Lösungen bieten. Die Hauptproblematik der User Interface-Architektur liegt in der Trennung der letztendlichen Darstellung des User Interfaces von den darzustellenden Inhalten und den Interaktionsmöglichkeiten zwischen Benutzer und System. Diese Problematik wurde von verschiedenen Modellen aufgegriffen, welche damit zu Architekturmustern für das User Interface werden. Die ersten Modelle, wie etwa das Seeheim Modell, kamen mit den User Interface Management Systems (UIMS, vgl. [Dix04, S. 306 ff.] [GSR05]) in einem frühen Stadium der User-Interface-Architektur auf und wurden später durch objektorientierte Modelle, wie zum Beispiel Model-View-Controller oder Presentation-Abstraction-Control,

fortgeführt. Letztere lassen Komponenten zu interagierenden Objekten, die zwischen Nutzer und Anwendung vermitteln, werden [HC97]. Alle Modelle sind hauptsächlich für traditionelle WIMP (Window-Icon-Menu-Pointer)-User Interfaces begründet worden [BCK02, S. 142].

Seeheim-Modell

Die einzelnen Ebenen des Seeheim-Modells (vgl. [BCK02, S. 131 ff.], siehe Abb. 4.4) entsprechen den Ebenen einer linguistischen Interaktion [HC97] [Dix04, S. 308 f.]. In der Linguistik unterscheidet man die semantische Ebene, die der Bedeutung des eigentlichen Inhalts entspricht, die syntaktische Ebene, welche Formprinzipien wiedergibt, und die lexikalische Ebene mit dem konkreten Wortschatz, verantwortlich für die Textrepräsentation. Übertragen auf die drei Bereiche des Seeheim-Modells bedeutet das folgendes: „Presentation“ (Darstellung) bezieht sich auf den lexikalischen Teil mit der Darstellung der für den Nutzer verfügbaren Ein- und Ausgabemöglichkeiten, „Dialogue Control“ (Dialogsteuerung) als Kommunikationsmedium zwischen Anwendung und Darstellung auf den syntaktischen Teil, und das „Application Interface“ (Anwendungsschnittstelle) entspricht mit den Anwendungsdaten und -funktionen der Semantik. Die Einordnung in diese drei Komponenten adressiert die Qualitätsattribute Änderbarkeit und Übertragbarkeit. Um Leistungseinbußen zu vermeiden, wurde im Seeheim-Modell die Möglichkeit einer Überbrückung der Dialogsteuerung geschaffen, was aber wiederum die Skalierbarkeit und Änderbarkeit negativ beeinflusst. Das Modell liefert grundlegende Gedanken für die Trennung von Anwendungs-, Dialog- und Präsentationsmodell, gehört an sich jedoch aufgrund der Weiterentwicklungen im User-Interface-Bereich der Vergangenheit an [UB96].

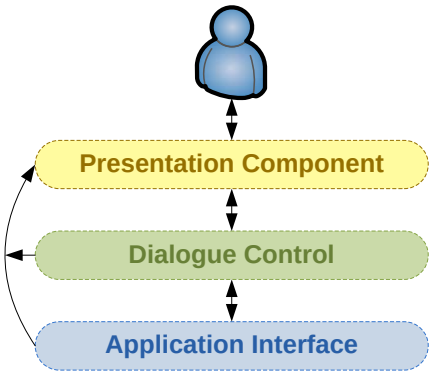

Abbildung 4.4.: Seeheim-Modell nach [BCK02]

Arch/Slinky-Modell

Das Arch/Slinky-Muster (vgl. [BCK02, S. 136ff] [Dix04, S.309], siehe Abb. 4.5) ähnelt dem Seeheim-Modell, fügt jedoch noch zwei weitere Schichten hinzu. Der Präsentationsschicht wird eine abstrakte Präsentationsschicht, „Virtual Toolkit“ genannt, vorangestellt. Zwischen Anwendungsschicht und Dialogmodell wird ebenfalls eine Schicht mit dem Namen „Virtual Application„ eingefügt. Damit wird gewährleistet, dass bei Änderungen in der Anwendungs- oder Präsentationsschicht auch das Dialogmodell unberührt bleibt und umgekehrt.

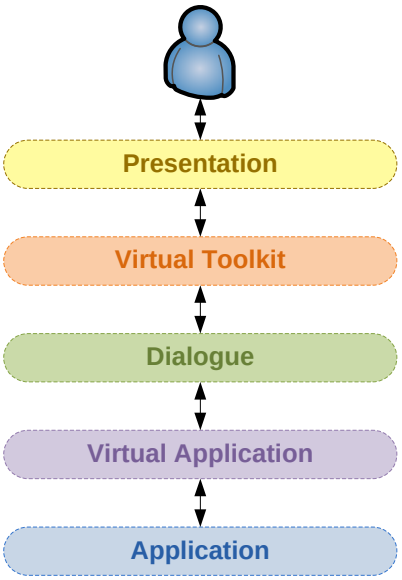

Abbildung 4.5.: Arch/Slinky-Muster nach [BCK02]

Model-View-Controller (MVC)

Das Model-View-Controller-Muster (vgl. [KP88] [HC97] [BCK02, S. 136ff], siehe Abb. 4.6) wird in Verbindung mit der Architektur von Benutzerschnittstellen am häufigsten genannt und eingesetzt. Das MVC-Muster gliedert ein System zunächst in mehrere Komponenten, im Gegensatz zum Seeheim-Modell werden jedoch zuerst unterschiedliche Bereiche der Anwendungsfunktionalität in sogenannte „Interaktionsobjekte“ (Interaction Objects) gekapselt. Auf einer tieferen Abstraktionsebene befinden sich die eigentlichen Bestandteile des MVC-Musters: Ein Model (Modell) repräsentiert den Anwendungsstatus und beinhaltet die Kernfunktionalität und die

Daten. Außerdem gibt es Views (Ansichten), welche Daten und Funktionen aus dem Model präsentieren und Eingabemöglichkeiten darstellen. Views verkörpern die Perzeption des Models durch den Nutzer. Ein View erfragt üblicherweise den Anwendungsstatus beim Model, auch kann das Model die Views über Änderungen des Anwendungsstatus benachrichtigen und damit eine Aktualisierung der Präsentation durchführen. Als dritter Bestandteil sind Steuerungskomponenten (Controller) zu nennen, welche Eingaben verarbeiten und Ereignisse steuern. Controller nehmen Aktualisierungen des Models vor und wählen Views für die Darstellung aus. Der Controller wird vom View über Eingabeereignisse informiert.

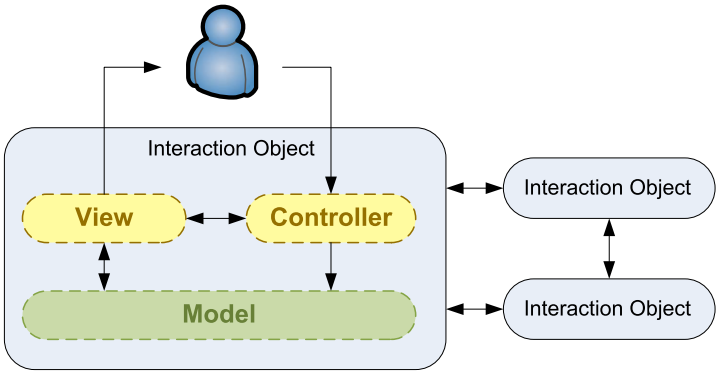

Abbildung 4.6.: Das Model-View-Controller-Muster

Wie auch das Seeheim-Muster ist die Hauptmotivation für die Anwendung des MVC-Musters in den Qualitätsattributen Änderbarkeit und Übertragbarkeit begründet, hier allerdings zusätzlich für die Komponenten der verschiedenen Anwendungsdomänen des Systems. Zuverlässigkeit und Sicherheit werden ebenfalls positiv bewertet. Die Verteilung domänenspezifischer Funktionalität auf verschiedene Komponenten trägt allerdings zur Erhöhung der Komplexität des Systems bei, ebenso beeinflusst die hohe Zahl an Aktualisierungsbotschaften zwischen den MVC-Teilen die Leistung des Systems negativ. [Bos00] [BCK02] Um die Skalierbarkeit zu erhöhen, können die MVC-Interaktionsobjekte im sogenannten HMVC-Muster (vgl. [CKP00]) auch hierarchisch in einer Schichtenarchitektur angeordnet werden.

Auf das User Interface des „Digitalen Filmarchivs“ lässt sich das MVC-Muster in seiner ursprünglichen Form nicht anwenden. Die Umgestaltung der Komponenten des „Digitalen Filmarchivs“ zu Interaktionsobjekten mit jeweils eigenen View- und Controller-Komponenten in der Präsentationsschicht würde sich negativ auf die Änderbarkeit, Übertragbarkeit und Interoperabilität des User Interfaces auswirken. Die fehlende zusätzliche Abstraktion der Anwendungslogik und des Präsentationsmodells

sowie die Kommunikationsbeziehungen zwischen Model und View machen die Nutzung generischer Anwendungsschnittstellen und eines generischen User-Interface-Modells unmöglich.

Presentation-Abstraction-Control (PAC)

Im PAC-Muster (vgl. [Dix04, S.310] [BCK02, S. 139 ff.] [HC97], siehe Abb. 4.7) wird das System ebenfalls in mehrere Interaktionsobjekte, hier auch Agenten genannt, unterteilt. Jeder Agent hat dabei einen eigenen Präsentations-, Abstraktions- und Kontrollteil. Die Agenten sind in diesem Muster hierarchisch angeordnet, die jeweiligen Kontrollteile benachrichtigen die verbundenen Agenten über Änderungen ihres Zustands. Zwischen Anwendungslogik und Präsentationslogik bestehen damit im Gegensatz zum MVC-Muster keine Abhängigkeiten mehr.

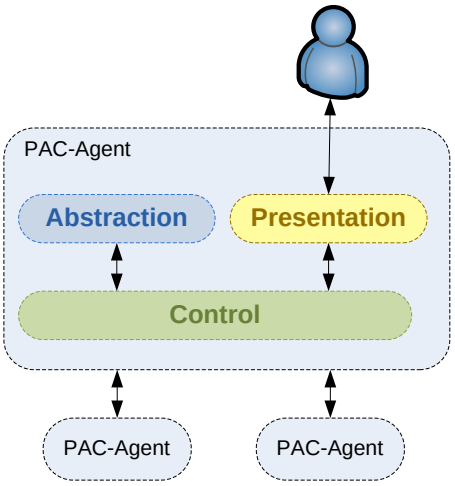

Abbildung 4.7.: Das PAC-Muster

Der Hauptnutzen des Musters liegt in der Skalierbarkeit und der Änderbarkeit. Um nun die Kapselung der Anwendungslogik im PAC-Agenten aufzuheben und damit auch die Übertragbarkeit und Interoperabilität zu steigern, wird dieses Modell in das Dialogmodell des Arch/Slinky-Musters integriert [NC91] [BCK02, S.140]. Das Resultat, genannt PAC-AMODEUS, kombiniert die beiden Architekturstile Layers und Objektorientierung. Unterschiede zum ähnlich strukturierten HMVC-Muster aus Abschnitt 4.2.4 ergeben sich darin, dass hier die gesamte Kommunikation zwi-

schen Anwendungs- und Präsentationsmodell immer über den Controller abgewickelt wird.

Mit der Anwendung des PAC-AMODEUS-Musters ist eine Erfüllung der Anforderungen für das User Interface des „Digitalen Filmarchivs“ umfassend möglich, es ergibt sich ein übereinstimmendes Profil in den Qualitätsattributen. Skalierbarkeit, Änderbarkeit und Übertragbarkeit stehen damit an erster Stelle. Die Idee zusätzlicher, von der Anwendungslogik und der Präsentation abgeleiteter Abstraktionsschichten trifft die Anforderungen der Nutzung generischer Anwendungsschnittstellen und eines generischen User-Interface-Modells. Das hierarchische Modell der PAC-Agenten ermöglicht zudem eine Verwirklichung modularer Komponenten, indem beispielsweise auf zwei Abstraktionsebenen eine übergeordnete Modulverwaltung untergeordnete User-Interface-Module verwaltet.

4.2.5. Model-Driven-Architecture (MDA)

Nach den Bestrebungen zu User Interface Management Systems (UIMS) hat sich im Zuge der Objektorientierung in der Softwareentwicklung die modellgetriebene Architektur einen Namen gemacht. Dies gilt auch für User Interfaces: Auf dem Gebiet des MBUID (Model-Based User Interface Development) wurden viele verschiedene MBUIDEs (Model-Based User Interface Development Environments) entwickelt [GSR05], von denen sich jedoch bis heute keine wirklich durchgesetzt hat. Ein gutes Modell sowie eine standardisierte User-Interface-Beschreibungssprache wie z.B. UML fehlen für User Interfaces leider. [Mol04] nennt dafür einige Gründe, beleuchtet jedoch auch die erheblichen Vorteile eines einheitlichen Modells.

Dennoch lassen sich Trends für die Organisation eines grundlegenden Musters zur modellgetriebenen User-Interface-Entwicklung erkennen [BC05] [GSR05] [GH00]. Dabei wird in ein Aufgaben-, Nutzer-, Anwendungs-, Dialog- und Präsentationsmodell unterschieden. Das Aufgabenmodell definiert Aktionen zur Erreichung des Ziels. Das Nutzermodell beschreibt die Fähigkeiten, Kenntnisse und Präferenzen des Nutzers. Das Präsentations-, Anwendungs- und Dialogmodell entspricht den Vorstellungen der bereits analysierten Architekturmuster für interaktive Systeme. Ähnlich wie im Arch/Slinky-Muster wird das Präsentationsmodell in ein abstraktes und ein konkretes Präsentationsmodell unterteilt. Das abstrakte Präsentationsmodell gibt das User Interface in einer abstrakten User-Interface-Beschreibungssprache wieder,

über eine Modelltransformation wird der Code für das letztendlich zur Anzeige gebrachte User Interface generiert.

Aktuell lässt sich sagen, dass nahezu alle aktuellen Versuche der modellgetriebenen User-Interface-Entwicklung auf dem XML-Standard beruhen [MML01]. Die User-Interface-Beschreibung wird dabei in einer XML-basierten Sprache verfasst, das Metamodell dazu als Document Type Definition (DTD) oder XML Schema Definition (XSD) hinterlegt. Modelltransformationen werden z.B. über XSLT vollzogen. Beispiele für XML-basierte User-Interface-Beschreibungssprachen sind Projekte wie Mozilla XUL, Microsofts XAML, Adobes MXML, OpenLaszlo LZX oder W3C XForms.

Die vollständige Transformation eines abstrakten Modells in ein konkretes Modell bedeutet einen hohen Leistungsaufwand für das System. Das konkrete Modell sollte daher nur für grundlegende Änderungen in der Konfiguration des User Interfaces aktualisiert werden. Als Alternative wäre die Vermeidung einer vollständigen Transformation durch die alleinige Nutzung eines konkreten Modells für die User-Interface-Beschreibung denkbar. Dieses konkrete Modell müsste alle Informationen zur vollständigen User-Interface-Darstellung beinhalten und eine Basispräsentation ohne weitere Transformation wiedergeben können. Ändern sich nun die Faktoren eines Teilmodells, beispielsweise der Darstellungstyp oder Nutzertyp, wird nur eine partielle Transformation vollzogen, die eine angepasste Version des konkreten Modells durch Veränderung eines Teilbereiches des ursprüngliches Modells erzeugt. Je höher die Wahrscheinlichkeit ist, dass ein bestimmtes angepasstes Modell für die Darstellung benötigt wird, desto geringer sollte die Zahl der Transformationsschritte zur Erreichung des angepassten Modells sein. Aus dem konkreten Modell sollen alle notwendigen angepassten Modelle über partielle Transformationen generierbar sein.

Die Nutzung eines User-Interface-Modells und verschiedener Transformationen zur Anpassung des Modells ermöglicht die Umsetzung grundlegender Anforderungen für das generische User Interface, vor allem bezogen auf Usability, Accessibility, die Testbarkeit und die Interoperabilität. So kann unter anderem die Anpassung an eine Dialogmodalität mit Hilfe von Modelltransformationen vorgenommen werden. Die Transformationsregeln ergeben sich beispielsweise aus der Definition verschiedener User-Interface-Arten, Nutzermodelle, Gestaltungsgundsätze oder Wörterbücher.

4.2.6. Referenzarchitekturen

XML-Pipeline

Die Pipeline (vgl. [SG96, S. 21]) ist eine Spezialform der in Abschnitt 4.2.2 erläuterten Pipes & Filters-Architektur. Der Datenstrom durchläuft dabei eine lineare Abfolge an Filterkomponenten. Das Architekturkonzept der XML-Pipeline ist im Rahmen der Entwicklung des Apache Cocoon Frameworks [The08] entstanden. Ein Datenstrom im XML-Format wird dabei durch mehrere Filter bearbeitet, welche in Generatoren, Transformatoren und Serializer kategorisiert werden (siehe Abb. 4.8). Generatoren kommunizieren mit Datenquellen und Anwendungsfunktionen und bereichern so den XML-Datenstrom, während Transformatoren Transformationsfunktionen nach bestimmten Transformationsregeln, vorrangig definiert über XSLT [Cla99], auf die XML-Daten anwenden. Serializer bringen den XML-Datenstrom dann in ein beliebiges Ausgabeformat.

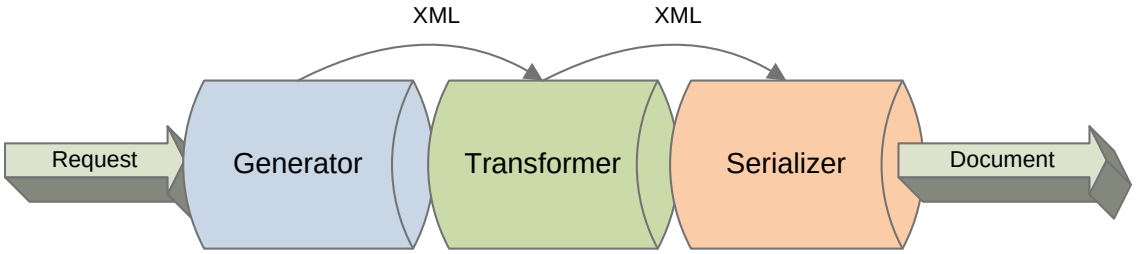

Abbildung 4.8.: Aufbau der XML-Pipeline

In Abschnitt 4.2.2 wurde eine positive Auswirkung der Pipes & Filters-Architektur auf die Qualitätsattribute des User Interfaces festgestellt, wenn mit ihr die innere Architektur eines Objektes in einem objektorientierten Modell verwirklicht wird. Daher eignet sich die XML-Pipeline für die innere Architektur der Agenten des PAC-AMODEUS-Musters. Die durch den Request beeinflusste Pipelinesteuerung stellt dabei den Controller dar. Über Generatoren und Transformatoren wird auf die abstrakte Schnittstelle, die User-Interface-Teile und Transformationsregeln zugegriffen und so das Präsentationsmodell erzeugt. Serializer geben am Ende der Pipeline das Präsentationsmodell, angepasst an die jeweilige Ausgabemodalität, aus. In der Pipelinesteuerung können Subpipelines aufgerufen werden, so dass eine hierarchische Anordnung der PAC-Agenten ebenfalls umgesetzt werden kann. Das XML-Format des Datenstroms entspricht den Bedingungen der modellgetriebenen User-Interface-

Architektur in Abschnitt 4.2.5 und wird damit zum Träger des Präsentationsmodells.

Portlets

Das Konzept der Portlets [Kla] [AH03] entstand aus den Bestrebungen der Java-Community, den interoperablen Komponenten komplexer Systeme ein einheitliches System zur Darstellung in einem gemeinsamen User Interface, dem Portal, zu schaffen. Die Portlets selbst sind dabei separate User-Interface-Teile, die von einem Portlet-Container verwaltet und angesprochen werden. Dieser wiederum kommuniziert mit dem Portlet Server, welcher die Aufgabe hat, die User-Interface-Teile zum Portal zusammenzuführen und für die Darstellung bereitzustellen.

Portlets vereinen die User-Interface-Teile mittels einer einfachen Aggregation. Das ermöglicht zwar eine freie Gestaltung der Portlet-Bereiche, gefährdet aber damit gleichzeitig ein konsistentes Verhalten und eine einheitliche Darstellung des User Interfaces. Somit sollten die UI-Teile nur in Form von Portlets integriert werden, wenn eine Komponente tatsächlich ein völlig unabhängiges User Interface benötigt. In allen anderen Fällen ist eher eine Komposition des User-Interfaces zu bevorzugen, welche die UI-Teile in eine einheitliche Struktur integriert.

Plugin-Architekturen

Mit einer Plugin-Architektur wird in dieser Arbeit das Ziel verfolgt, die Erweiterbarkeit eines Systems so zu gestalten, dass zusätzliche Module hinzugefügt und deren Funktionen genutzt werden können ohne Änderungen im Quellcode der Software durchführen zu müssen. Als erstes wird dazu die Eclipse Plugin-Architektur [Bol03] betrachtet. Jedes Plugin-Modul besitzt hier ein Manifest mit grundlegenden Informationen. Unter anderem wird darin auch beschrieben, um welchen Plugin-Typ es sich handelt und welche Abhängigkeiten zu weiteren Plugin-Modulen bestehen. Dieses Manifest wird von der übergeordneten Plugin-Verwaltung (Registry) eingelesen und das Plugin der Anwendung zur Verfügung gestellt. Es existieren zwei Plugin-Typen, Host-Plugins und Extender-Plugins, welche folgende Beziehung zueinander haben: Host-Plugins können sogenannte „Extension-Points“ besitzen, an denen „Extender-Plugins“ ihre Funktionen einbringen können. Einen ähnlichen Weg geht die Mozilla Chrome Plugin-Architektur [Sme06]. Allerdings gibt es dort keine verschiedenen

Plugin-Typen und das Zusammenfügen des XML-basierten Plugin-Codes wird mittels einer Overlay-Technik vollzogen.

Da beim User Interface des „Digitalen Filmarchivs“ die Teile des User-Interface-Modells in den Plugin-Modulen ebenfalls im XML-Format vorliegen sollen, kann die Overlay-Technik zur Erzeugung des gesamten User-Interface-Modells angewandt werden. Die Idee des Manifests, der Plugin-Verwaltung und der Verwaltung von Abhängigkeiten wird ebenfalls in die Architektur für das User Interface übernommen.

4.3. Strategien und Kernbestandteile der Architektur

In diesem Abschnitt vereinen sich nun die in den Vorbetrachtungen ausgewählten Entwurfsentscheidungen zu einer Gesamtstrategie. Damit wird der Architekturentwurf geschaffen. Im Rahmen der Erläuterung der Organisationsstruktur werden auch die Funktion und die Bedeutung der einzelnen Komponenten näher beleuchtet.

Die grundlegende Struktur für die Einordnung des User Interfaces in die dreischichtige Architektur des Systems „Digitales Filmarchiv“ (siehe Abschnitt 2.3.2) wird vom PAC-AMODEUS-Muster bestimmt (siehe Abb. 4.9). Die oberste Schicht des PAC-AMODEUS-Musters „Presentation“ gleicht der Präsentationsschicht des „Digitalen Filmarchivs“. Die drei mittleren Schichten des PAC-AMODEUS-Musters („Virtual Toolkit, „Dialogue“ und „Virtual Application“) sind in der Komponente „User Interface Management“gekapselt. Die unterste Schicht des PAC-AMODEUS-Musters („Application“) ist mit den weiteren Anwendungskomponenten des „Digitalen Filmarchivs“ in der Anwendungsschicht gegeben. An den Grenzen der obersten und untersten Schicht zur mittleren Schicht des PAC-Amodeus-Musters findet damit ein Sprung zwischen den Betrachtungsebenen des Systems (siehe Tabelle 4.1) statt.

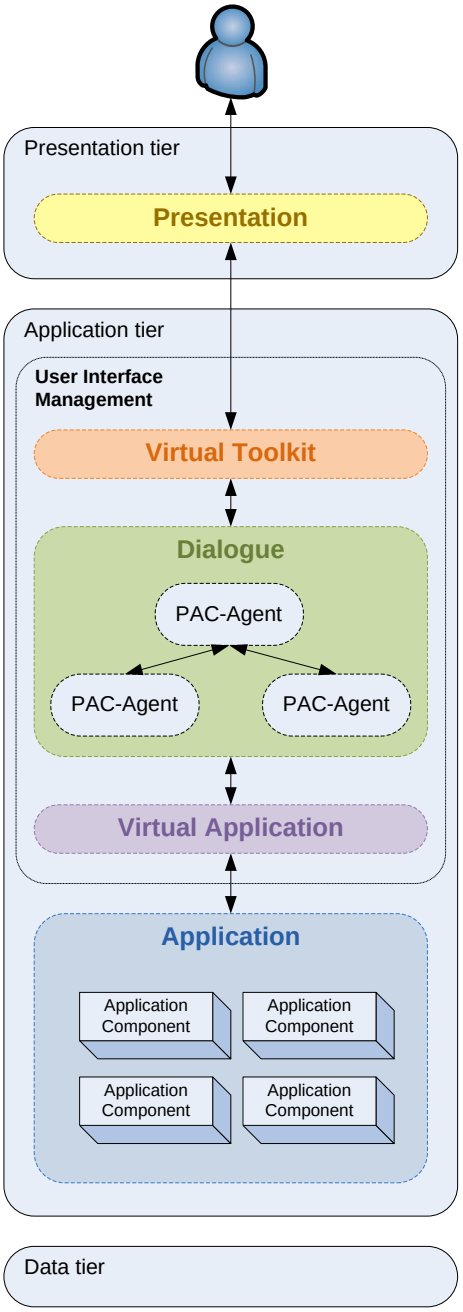

Abbildung 4.9.: Einordnung der PAC-AMODEUS-Komponenten in die dreischichtige Systemarchitektur des „Digitalen Filmarchivs“

4.3.1. Präsentationsschicht

Die Präsentationsschicht bildet die oberste Schicht in der Architektur des „Digitalen Filmarchivs“. Sie enthält das eigentliche User Interface, welches die Oberfläche der Anwendung präsentiert und die Interaktionen des Nutzers registriert. Die Präsenta-

tion erfolgt für eine spezifische Dialogmodalität, nach bestimmten Gestaltungsprinzipien und angepasst an konkrete Nutzereigenschaften. Den User-Interface-Code dafür liefert eine auf der REST-Architektur basierende Schnittstelle der „User Interface Management“-Komponente in der Anwendungsschicht des „Digitalen Filmarchivs“. Benutzerereignisse werden in Form von Anfragen an eindeutige Ressourcen der „User Interface Management“-Komponente weitergegeben. Diese gibt die Repräsentation der REST-Ressourcen in Form eines XML-Dokuments zurück. Die Persistenz wird dabei über HTTP-Sessions erzielt. Für die webbasierende GUI des „Digitalen Filmarchivs“ wird die Präsentationsschicht mit einem Webbrowser als nutzerseitiger Client umgesetzt. Dieser Webbrowser sollte eine Ajax-Technologie ähnlich der „Ajax Engine“ unterstützen, um dynamische Aktualisierungen der Darstellung und des Anwendungsmodells zu erlauben.

4.3.2. Das Subsystem „User Interface Management“

Das Subsystem „User Interface Management“ (siehe Abb. 4.10) befindet sich in der Schicht der Anwendungslogik des „Digitalen Filmarchivs“. Die Komponenten dieses Subsystems realisieren die drei mittleren Schichten des PAC-AMODEUS-Musters. Das „Virtual Toolkit“ wird zum „Presentation Toolkit“, die Funktion der „Dialogue“-Schicht übernimmt der „User Interface Container“ und aus der Idee der „Virtual Application“ ist der „Interface Mapper“ entstanden.

Die Komponenten des Subsystems „User Interface Management“ übernehmen Aufgaben sowohl der Erzeugung und Verwaltung des User Interfaces, als auch der Steuerung der Kommunikation zwischen Präsentationsschicht und weiteren Anwendungskomponenten des „Digitalen Filmarchivs“. Sämtliche Schnittstellen sowie die Kommunikation zwischen den Komponenten innerhalb des Subsystems „User Interface Management“ folgen in ihrer Gestaltung der REST-Architektur. Zusätzliche Caching-Mechanismen sollen an den Schnittstellen Requests ohne eine Verarbeitung beantworten können. Es schließt sich jetzt eine nähere Betrachtung der Komponenten an.

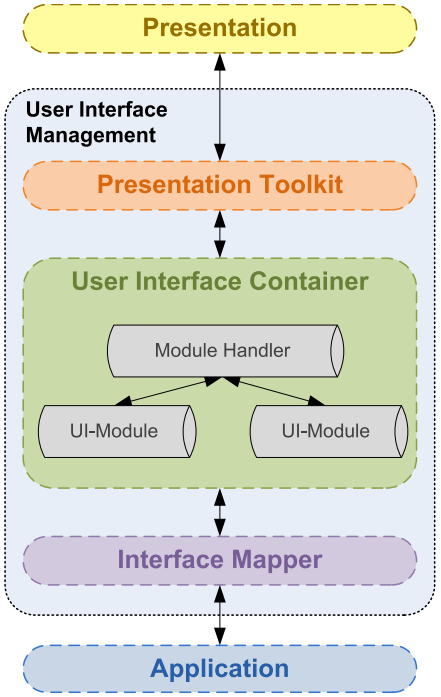

Abbildung 4.10.: Architektur des Subsystems „User Interface Management“

Presentation Toolkit

Das Presentation Toolkit hat die Aufgabe, aus dem User-Interface-Modell den User-Interface-Code für die Präsentationsschicht zu generieren. Je nach User-Interface-Typ und Ausgabemodalität kann dieser Verarbeitungsschritt auch auf der Nutzerseite erfolgen. Für das grafische, webbasierte User Interface ist hier ein Rich Internet Application Framework angebracht, welches den User-Interface-Code für den Browser oder ein Browser-Plugin erzeugt. Im User-Interface-Code befindet sich bei der Verwendung der Ajax-Technologie mit der „Ajax Engine“ ein zweites Presentation Toolkit, welches dynamische Aktualisierungsfunktionen der Darstellung und Requests an den User Interface Container auf der Nutzerseite verarbeiten kann. Für das Command Line Interface müsste hier der Code für ein Programm erzeugt werden, welches das Präsentationsmodell für den Command-Line-Interpreter übersetzt.

Das Presentation Toolkit definiert damit die Schnittstelle des Subsystems „User Interface Management“ für die Kommunikation mit der Präsentationsschicht. Es sendet den Code für das User Interface und empfängt Benutzerereignisse

User Interface Container

Der User Interface Container übernimmt die Aufgabe des Zusammenführens verschiedener User-Interface-Teile und der Erstellung und Transformation des User-Interface-Modells. Er definiert die Schnittstelle, über die das User Interface-Modell an das Presentation Toolkit gegeben wird. Außerdem stellt er die Verbindung zu den abstrakten Anwendungsschnittstellen her. Da der User Interface Container die Dialogschicht des PAC-AMODEUS-Musters umsetzt, wird mit ihm ein neues Subsystem erschlossen (siehe Tab. 4.1). Dieses ist durch das objektorientierte PAC-Muster hierarchisch angeordneter Agenten geprägt. Die PAC-Agenten folgen in ihrer inneren Struktur der XML-Pipeline-Architektur. Auf der obersten Ebene dieser Hierarchie findet sich der Controller mit Sessionverwaltung (Session Handler) und Modulverwaltung (Module Handler) wieder, eine Ebene tiefer die einzelnen Plugin-Module. Jedes Plugin-Modul muss eine Pipeline definieren und ein Manifest besitzen, in dem grundlegende Informationen zum Modul sowie dessen Abhängigkeiten enthalten sind. Weitere Modulinhalte können Teile des User-Interface-Modells, Elemente des User Interfaces, Konfigurationseinstellungen, Wörterbucheinträge und Transformationsregeln für Transformationen des User-Interface-Modells sein. Die Module des User Interfaces sind in einer Konfigurationsoberfläche der Modulverwaltung sichtbar und können dort aktiviert, deaktiviert und konfiguriert werden. Die Modulverwaltung steuert zudem, wie die Teile des User Interfaces zusammengeführt werden. Dafür werden starre Transformationsregeln genutzt, welche User-Interface-Beschreibungen in eine einheitliche Struktur mit gemeinsam genutzten Elementen bringen und so ein einheitliches, konsistentes User Interface-Modell erzeugen.

Den vorangegangenen Gedanken zur modellbasierten Entwicklung des User Interfaces im Abschnitt 4.2.5 entsprechend, wird auf ein abstraktes Präsentationsmodell des User Interfaces verzichtet. Durch die dynamische Einbindung der verschiedenen Plugin-Module können neben dem konkreten Präsentationsmodell verschiedene angepasste Modelle des User Interfaces direkt erzeugt werden. So erfolgt bereits hier eine Anpassung, unter anderem an die Dialogmodalität sowie weitere Accessibility- und Usabilityfaktoren des User Interfaces.

Betrachtungs-ebene	Architekturstil	Architektur-muster	Referenz-architektur
Softwaresystem „Digitales Filmarchiv“	Layers	Client-Server	3-Schichten-Architektur
Anwendungs-schicht des „Digitalen Filmarchivs“	Objekt-orientierung	/	/
Subsystem „User Interface Management“ in der Anwendungsschicht	Layers	Arch/Slinky (PAC-AMODEUS), REST, Ajax	/
Komponente User Interface Container des Subsystems „User Interface Management“	Objekt-orientierung, Hierarchie	PAC	Plugin-Architektur, Portlets
PAC-Agenten des User Interface Containers	Pipes & Filters	/	XML-Pipeline

Tabelle 4.1.: Betrachtungsebenen des „Digitalen Filmarchivs“ und deren innere Architektur

Interface Mapper

Die Anwendungskomponenten des „Digitalen Filmarchivs“ definieren Schnittstellen, über die sie den Zugriff auf die Funktionalität und das Datenmodell der Anwendungskomponente bereitstellen. Dabei spielen verschiedene Schnittstellentypen eine Rolle. So kann eine Schnittstelle zum Beispiel als SOAP-basierter Webservice oder als Java API zur Verfügung gestellt werden. Diese Schnittstellen verkörpern die Verbindung zwischen Anwendungsschicht und abstrakter Anwendungsschicht. Die abstrakte Anwendungsschicht wird nun mit dem Interface Mapper umgesetzt. Der Interface Mapper besitzt im Wesentlichen drei Aufgaben: Erstens soll mit ihm der Zugriff auf verschiedene Schnittstellentypen anderer Anwendungskomponenten ermöglicht werden. Zweitens soll eine einheitliche, abstrahierte Schnittstelle auf Basis der REST-Architektur definiert werden, welche vom User Interface Container

des Subsystems „User Interface Management“ angesprochen werden kann. Drittens soll eine Zuweisung der Funktionen der abstrahierten Schnittstelle auf die jeweilige Anwendungsschnittstelle stattfinden. Außerdem übernimmt der Interface Mapper weitere Proxyfunktionalitäten, wie etwa das Caching oder Firewalling. Der Interface Mapper ist Client für das Anwendungsmodell und Server für das abstrahierte Anwendungsmodell gleichzeitig. Damit wird die gesamte Anwendungsschnittstelle in einer Komponente gekapselt und das Hinzufügen oder der Austausch von Schnittstellentypen erleichtert.

4.4. Beispiel: GUI als Webanwendung

Dieser Abschnitt soll den Architekturentwurf bezogen auf den speziellen Fall der grafischen Benutzeroberfläche einer dynamischen, Ajax-getriebenen Webanwendung darstellen und liefert damit bereits einen kurzen Vorgeschmack auf die Implementierung in Kapitel 5.

In der Präsentationsebene befindet sich nun der Browser, der die Webanwendung darstellt und Benutzereingaben empfängt. Der Browser kommuniziert über das HTTP-Protokoll mit einem Webserver. Der Webserver ist identisch mit dem Subsystem „User Interface-Management“, ein RIA-Framework wird als Presentation Toolkit genutzt und befindet sich gemeinsam mit dem User Interface Container und dem Interface Mapper auf dem Webserver. Besonders interessant ist die Abfolge von Kommunikationsschritten zwischen den Komponenten des User Interfaces, welche in Abb. 4.11 veranschaulicht wird.

Zunächst wird die Webanwendung im Browser vom Benutzer aufgerufen und damit ein HTTP-Request an den Webserver, gerichtet an das RIA-Framework, erzeugt. Mit dem Request wird der Code für die Webanwendung angefragt, zusätzlich werden in den HTTP-Header-Daten oder der Request-URI verschiedene Parameter, wie etwa der Browsertyp, die eingestellte Sprache oder eine bestimmte Zugriffsdomäne, übergeben.

Der Server des RIA-Frameworks prüft nun zuerst, ob für diesen Request eine Cache-Version des User-Interface-Codes vorhanden ist und ob diese für die Antwort genutzt werden kann. Falls die Antwort nicht sofort mit der Cache-Version des User-Interface-Codes erfolgt, leitet das RIA-Framework den Request an den User-Interface-Container

weiter. Hier wird das User-Interface-Modell für diesen Request erzeugt und an das RIA-Framework zurückgegeben. Das RIA-Framework erzeugt daraus den User Interface-Code, speichert diesen im Cache und sendet ihn an den Browser.

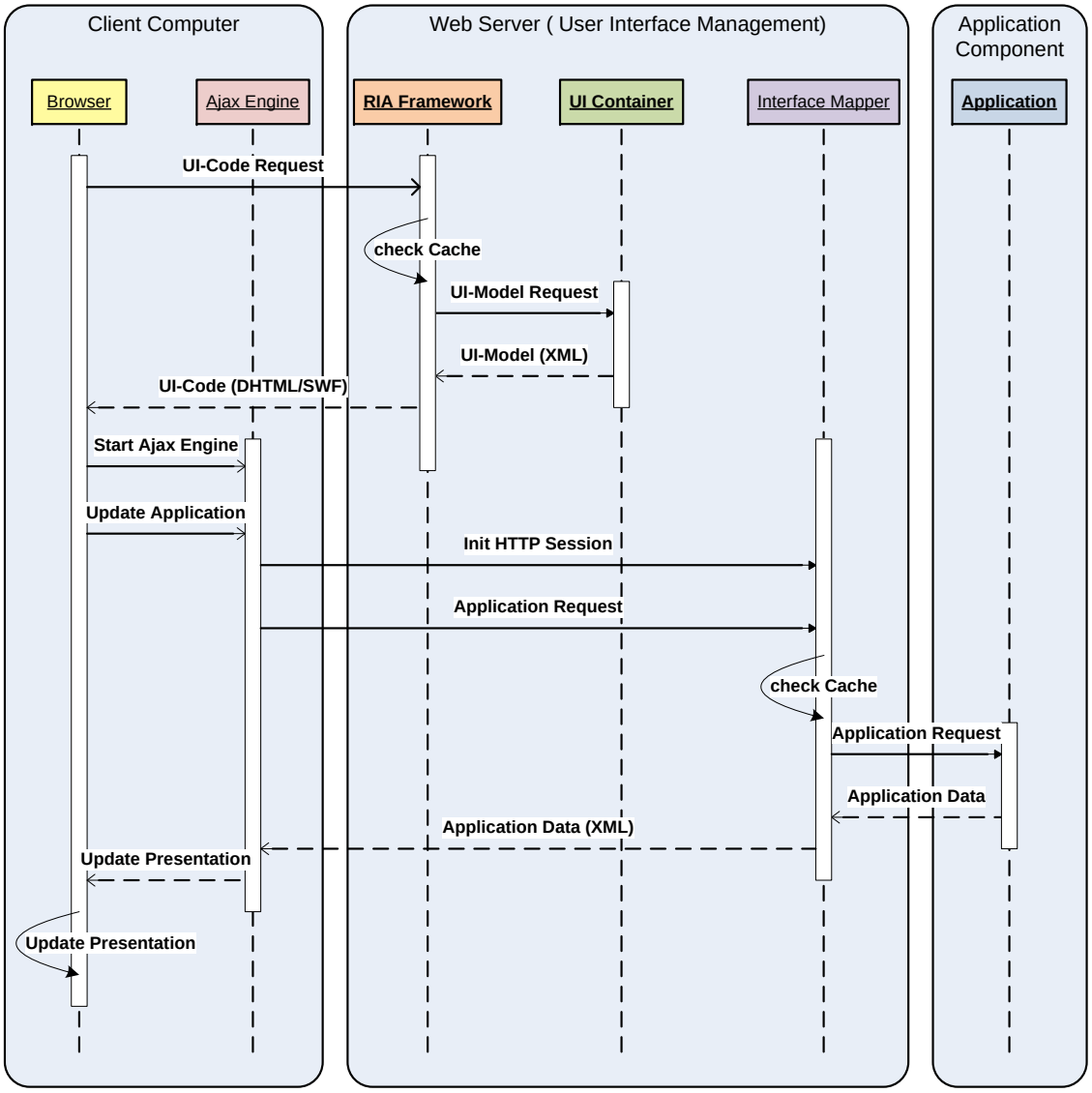

Abbildung 4.11.: Abfolge der Kommunikationsschritte für den Aufbau und die Aktualisierung des webbasierten User Interfaces

Nachdem der Browser den Code des User Interfaces interpretiert hat, wird dem Nutzer nun die Webanwendung dargestellt. Im UI-Code werden die „Ajax-Engine“ sowie weitere Technologien, welche die dynamische Aktualisierung der Darstellung und des in der Darstellung enthaltenen Anwendungsmodells ermöglichen, eingebunden. Änderungen der Darstellung können also im Browser ausgeführt werden, ohne

dass eine Kommunikation mit dem Webserver notwendig ist. Ist das in der Darstellung enthaltene Anwendungsmodell von einer Änderung betroffen, kann diese Aktualisierung gezielt mit einem Request an den Webserver vorgenommen werden. Die „Ajax-Engine“ macht ein Überspringen des Presentation Toolkits und des User Interface Containers möglich, Teile des Anwendungsmodells können über eine direkte Kommunikation mit dem Interface-Mapper separat vom gesamten Modell des User-Interfaces abgeglichen werden.

4.5. Bewertung des Architekturentwurfs

Mit der Bewertung des Architekturentwurfes erfolgt eine ausführliche Rückbesinnung auf die Anforderungen des User Interfaces aus Kapitel 3. Eine Erfüllung der ursprünglichen Ziele (Z_x) sowie der spezifizierten Anforderungen (R_x, Qualitätsattribute) soll verifiziert und ein Bezug zu Anwendungsszenarien (UC-x) hergestellt werden. Nachdem in den Vorbetrachtungen zum Architekturentwurf (Abschnitt 4.2) Bezug auf die Anforderungen aus Sicht der Hilfsmittel genommen wurde, soll nun aus Sicht der Anforderungen auf den Architekturentwurf geblickt werden.

4.5.1. Erweiterbarkeit und Modularität

Das User Interface kann jederzeit um konfigurierbare Plugin Module erweitert werden (Z_1, UC-3, R_1). Der Inhalt der Plugin-Module, und damit das Spektrum erweiterbarer Bestandteile des User Interfaces, ist umfassend. Neue Anwendungskomponenten können ihren Teil zum User-Interface-Modell hinzufügen (R_3), verschiedene Dialogmodalitäten lassen sich ebenfalls in Plugin-Module packen (R_5). Ein User-Interface-Modell für Archivfunktionen (Z_6) oder eine Konfigurationsoberfläche (Z_2, R_4) kann also genauso als Plugin-Modul in das User Interface integriert werden wie ein eine GUI, ein CLI oder ein VUI (Z_3, R_5).

4.5.2. Accessibility und Dialogmodalität

Die Nutzung eines User-Interface-Modells macht es möglich, über Transformationen Anpassungen an die Dialogmodalität (Z_3) und weitere Accessibilityfaktoren, wie

zum Beispiel die Mehrsprachigkeit, vorzunehmen (Z_4, Z_8). Die Präsentationsschicht kann geändert werden, ohne dass die Dialogschicht oder tiefere Schichten angepasst werden müssen. Entsprechende Transformationsregeln können mit Plugin-Modulen ergänzt werden, bei Bedarf kann ein angepasstes oder ein zusätzliches Presentation Toolkit eingesetzt werden.

4.5.3. Usability

Einen wesentlichen Beitrag zur Usability des Systems (Z_4) leistet der Module Handler, da hier die Teile des User Interfaces zu einem gemeinsamen, konsistenten User-Interface-Modell vereint werden. Dieses zentrale Element garantiert, dass auch bei einer Vielzahl von Erweiterungen die Benutzerfreundlichkeit des User Interfaces erhalten werden kann. Die Nutzung einer einheitlichen Beschreibungssprache für das User-Interface-Modell soll eine Vielzahl standardisierter Navigationsstrukturen, Ansichtstypen und User-Interface-Elemente möglich machen (UC-4, R_2, R_3). Die Nutzung der Ajax-Technologie für die webbasierte GUI trägt ebenfalls zur Usability der Oberfläche bei.

Weitere Usabilityziele werden mit Persistenz (R_8) erreicht. Der Status der Benutzerinteraktion lässt sich auf Ebene der User-Interface-Präsentation, des User-Interface-Modells, des Interface Mappers und der Anwendung speichern und wiederherstellen. Zwischen Präsentationsschicht und dem Subsystem „User Interface Management“ lassen sich die jeweiligen Zustände über HTTP-Sessiondaten identifizieren. Zwischen Interface Mapper und Anwendungskomponenten können die eigenen Schnittstellen und Persistenzmechanismen der jeweiligen Anwendungskomponenten eingesetzt werden. Die Nutzerverwaltung selbst (Z_5, UC-2, R_7) kann im User Interface Container oder als Anwendungskomponente umgesetzt werden, eine Oberfläche zur Nutzerverwaltung lässt sich als Plugin-Modul in das User-Interface-Modell einbringen.

4.5.4. Umsetzung der Qualitätsattribute

Mit der Einführung des Interface Mappers wurde die Wahl der Schnittstellen zu den Anwendungskomponenten des „Digitalen Filmarchivs“ bewusst offen gelassen. Dies erzeugt maximale Interoperabilität des Subsystems „User Interface Management“. Das gleiche Prinzip gilt für die Einführung des Presentation Toolkits und eines auf

dem XML-Standard basierenden User-Interface-Modells als Vorstufe zur letztendlichen Präsentation. Mit dem hohen Grad an Interoperabilität wird auch eine einfache Übertragbarkeit des Subsystems „User Interface Management“ gewährleistet. Für die Komponenten dieses Subsystems wird Interoperabilität und Übertragbarkeit durch die Verwendung offener Standards, wie XML und XSLT, erzielt.

Auf die Änderbarkeit wirken sich die beiden zusätzlichen Schichten um den User Interface Container des Subsystems „User Interface Management“ ebenfalls positiv aus. Änderungen in der Erzeugung des User-Interface-Modells betreffen nur den User Interface Container und können vorgenommen werden, ohne dass Anwendungskomponenten oder die Präsentation angepasst werden müssen. Der Skalierbarkeit des Systems kommt die feingranulare Untergliederung der User-Interface-Komponenten bis hin zu dynamisch aktivierbaren Modulen, die leicht verteilbare XML-Pipeline-Architektur der PAC-Agenten im User Interface Container, sowie die Verwendung von REST als zustandslose und damit ebenfalls leicht verteilbare Kommunikationsarchitektur zugute.

Die Leistung wurde beim Entwurf mit niedrigster Priorität betrachtet, dennoch ist es wichtig, dass das User Interface möglichst effizient und mit konstanten und akzeptablen Reaktionszeiten arbeitet. Darauf zielen vor allem Caching-Mechanismen ab. Diese sind zum einen am Interface-Mapper vorgesehen, wo gepufferte Anwendungsdaten wiedergegeben werden können, ohne dass Anwendungsschnittstellen beansprucht werden müssen. So kann vor allem bei Anwendungskomponenten mit sehr niedrigen Antwortzeiten Abhilfe geschaffen werden. Auch die angepassten User-Interface-Modelle sowie der daraus generierte User-Interface-Code sollen ein Caching durchlaufen. Benutzerinteraktionen, welche häufige Darstellungsänderungen und damit verbundene Aktualisierungen mit sich bringen, werden für die webbasierte GUI in der Präsentationsschicht mit Ajax-Technologie umgesetzt und so die Ressourcen des Systems geschont.

Die Testbarkeit des Systems wird durch den Architekturentwurf hervorragend unterstützt. Das User-Interface-Modell lässt sich im Laufe der Verarbeitung leicht mit Fehler- und Debuginformationen anreichern, ohne dabei die Präsentation zu gefährden. Die Erstellung des User-Interface-Modells geschieht innerhalb einer Pipeline-Architektur, womit der Ablauf der Verarbeitungsschritte eindeutig nachvollzogen und das Ergebnis nach jedem Schritt geprüft werden kann. Die Beschreibung des User-Interface-Modells sowie die Repräsentation der REST-Ressourcen erfolgt im menschenlesbaren XML-Format, die transportierten Informationen können an jeder

Stelle ohne Hilfsmittel eingesehen werden.

Die Zuverlässigkeit des User Interfaces wird durch die gute Testbarkeit positiv beeinflusst, mit der Architektur ist außerdem eine umfassende Fehlerbehandlung in verschiedenen Komponenten umsetzbar. Bei kritischen Systemfehlern können die Zustände wie bereits in Abschnitt (4.5.3) beschrieben mit Hilfe der Persistenzmechanismen wiederhergestellt werden.

4.5.5. Einfluss der Rahmenbedingungen

Die Rahmenbedingungen für den Architekturentwurf (siehe Abschnitt 3.4.3) zeichneten sich vor allem darin aus, dass sie keine genauen Aussagen über Einflussfaktoren liefern konnten. Eine mögliche spätere Anpassung an Einflussfaktoren wurde nun mit der generischen Architektur erreicht. Das User Interface lässt sich mit einer variierenden Zahl an Anwendungskomponenten mit unterschiedlichen Schnittstellentypen betreiben. Die hohe Interoperabilität und Übertragbarkeit des Systems machen eine relativ freie Gestaltung der technischen Umsetzung möglich. Für die Erstellung eines Demonstrators in Form einer Weboberfläche mit Ajax-Unterstützung sind alle Voraussetzungen erfüllt.

Kapitel 5.

Implementierung

Das Kapitel der Implementierung beschreibt die beispielhafte Umsetzung des Architekturentwurfs für den Spezialfall des grafischen User Interfaces einer dynamischen Webanwendung. Grundlegend dafür ist der bereits als Beispiel konkretisierte Architekturentwurf für diesen Fall in Abschnitt 4.4. Zuerst werden nach dem Stand der Technik geeignete Mittel für die Implementierung ausgewählt. Danach wird der, auf Basis des Architekturentwurfs entstandene, Prototyp erläutert und Beispielimplementierungen im Bereich der Transformation des User-Interface-Modells durchgeführt.

5.1. Stand der Technik

Zunächst wird der Stand der Technik in Bezug auf geeignete Software für die Umsetzung der Komponenten des User Interfaces betrachtet. Im Zuge dessen erfolgt auch die Auswahl einer Beschreibungssprache für das UI-Modell und des Formates für den UI-Code.

5.1.1. Webbrowser, Browser-Plugins und der User Interface Code

Die Präsentationsschicht des „Digitalen Filmarchivs“ beinhaltet den Webbrowser. Dieser interpretiert den User-Interface-Code, üblicherweise bestehend aus HTML oder XHTML, CSS und JavaScript, zusammen auch DHTML (Dynamic HTML)

genannt. Durch Browser-Plugins ist die Anzeige weiterer Formate, wie beispielsweise Shockwave Flash (SWF), Java oder Quicktime, möglich. Vor der Wahl des Formates für den User-Interface-Code müssen drei Aspekte geprüft werden:

I. Verbreitung der Fähigkeit zur Darstellung des Formates

Hierzu werden Studien herangezogen, welche Auskunft über die Verbreitung bzw. Marktdurchdringung der Webbrowser und Browser-Plugins mit dieser Fähigkeit zur Darstellung des Formates geben. Von den in [Sto08] genannten heterogenen Nutzergruppen kann nicht auf eine einheitliche Browserkonfiguration geschlossen werden, so dass von allgemeinen Statistiken zur Verbreitung von Webbrowsern und Browser-Plugins ausgegangen werden muss. Ist die Fähigkeit zur Darstellung eines Formates nicht weit verbreitet, sollte zumindest die nachträgliche Installation eines entsprechenden Browser-Plugins für den Nutzer kein Hindernis darstellen und keine Einschränkungen mit sich bringen.

II. Einheitliche Darstellung des Formates

Für eine zuverlässige Darstellung ist eine einheitliche Interpretation des Formates in verschiedenen Browserkonfigurationen und Plugin-Typen sowie deren Versionen und Einstellungen wichtig. Hier können zum Teil gravierende Differenzen auftreten, welche einschneidende Auswirkungen auf die Usability und Accessibility des User Interfaces haben.

III. Integration verschiedener Medientypen

Ein letztes, wichtiges Kriterium ist die Fähigkeit zur Darstellung verschiedener Medientypen. Die Webanwendung für das „Digitale Filmarchiv“ erfordert die Integration vielfältiger Medientypen. Sowohl Hypertext als auch multimediale Inhalte, beispielsweise das Streaming von MPEG4-codierten Videos, sollen integriert werden. Je mehr Medientypen ein Format darstellen kann, desto geringer ist die Notwendigkeit mehrere Formate zu kombinieren.

Analysiert man die zur Auswahl stehenden Formate, ergibt sich in keinem Format eine vollständige Erfüllung der drei Kriterien. Die für Webanwendungen übliche

Kombination aus HTML/XHTML, CSS und JavaScript wird zwar von fast allen Webbrowsern akzeptiert, jedoch von unterschiedlichen Browsertypen, -versionen und mit verschiedenen Einstellungen nicht einheitlich dargestellt. Für Multimediainhalte ist man auf die Unterstützung zusätzlicher Browser-Plugins oder weiterer Software angewiesen.

Das Browser-Plugin Adobe Flash Player zur Anzeige von SWF ist ebenfalls sehr weit verbreitet [Ado08b] und unterstützt zudem Multimediainhalte inklusive Streaming sowie Ajax-Technologie. Zwischen verschiedenen Versionen des Plugins gibt es Differenzen in der Darstellung, jedoch ist die Verbreitung der aktuellen Version und die Aufnahme von Aktualisierungen sehr gut [Ado08a]. Eine nachträgliche Installation führt nur zu einem geringen Installations- und Wartungsaufwand.

Die Verwendung von Java auf Seite des Clients ist trotz vielseitiger Fähigkeiten nicht empfehlenswert. Nachteile sind die geringere Verbreitung, vor allem der aktuellen Version [Rea06], und der hohe Installations- und Wartungsaufwand. Ähnliche Probleme würden bei einer Verwendung des Quicktime- oder RealPlayer-Plugins entstehen. [Ado08b]

Auch die Verwendung des XForms-Standards [Boy07] oder des gut dokumentierten Mozilla-XUL Schemas [GHHW01] als User-Interface-Code würde den Großteil der Nutzer ausschließen. XForms ist auf ein kaum verbreitetes Browser-Plugin angewiesen, XUL benötigt die lediglich in Mozilla-Webbrowsern benutzte Gecko-Layout-Engine.

5.1.2. Rich Internet Application Frameworks und das User-Interface-Modell

Rich Internet Application Frameworks erstellen aus dem Modell des User Interfaces den User Interface Code für eine dynamische Webanwendung. Eine dynamische Webanwendung im Stil der Rich Internet Application (RIA) vereinigt Ajax-Technologie mit einer Vielzahl einheitlicher UI-Elemente und Darstellungsstrukturen sowie verschiedener Kommunikationsschnittstellen. Die Verarbeitung von Aktualisierungen des Präsentationsmodells erfolgt clientseitig.

Für das User Interface des „Digitalen Filmarchivs“ eignet sich das OpenLaszlo- [Las06] oder ZK-Framework [Pot08]. Beide nutzen jeweils eine objektorientierte und

XML-basierte Beschreibungssprache für das User-Interface-Modell und sind quelloffen. Grundlage für den Betrieb beider Frameworks ist der auf plattformunabhängiger Java-Technologie basierende Apache Tomcat Server. Tabelle 5.1 zeigt die Unterschiede der beiden RIA-Frameworks auf.

RIA-Framework	UI-Modellsprache	Ausgabeformat	Lizenz
OpenLaszlo	LZX (ECMA-Script, Action-Script, Xpath)	SWF, DHTML	CPL 1.0
ZK	ZUML (XUL, XHTML, verschiedene Script-sprachen)	DHTML	GPL

Tabelle 5.1.: RIA-Frameworks im Vergleich

Als RIA-Framework für die Implementierung des User Interfaces wird OpenLaszlo gewählt. Hier kann das vorteilhafte SWF-Format für den User-Interface-Code genutzt werden, außerdem tragen die weniger restriktiven Lizenzbedingungen des OpenLaszlo-Frameworks zu dieser Entscheidung bei. Das User-Interface wird damit in Form einer SWF-Webanwendung im Flash-Player-Plugin des Webbrowsers betrieben.

Die XML-basierte Beschreibungssprache für das User-Interface-Modell ist somit OpenLaszlos LZX. User-Interface-Elemente können hier in einer objektorientierten Struktur als XML-Elemente mit Attributen angeordnet werden, zusätzlich ist die Einbindung von ECMA-Script zur Steuerung von dynamischen Veränderungen möglich. Eine Vielzahl von Events registrieren Nutzeraktionen. Auf REST-Schnittstellen kann dynamisch zugegriffen werden, die Auswertung der Antwort im XML-Format erfolgt mit Hilfe grundlegender XPath-Funktionalitäten.

5.1.3. Dialogue Controller und Interface Mapper mit DSpace Manakin

Die letzten beiden User-Interface-Komponenten, der Dialogue Controller und der Interface Mapper, werden nun mit DSpace Manakin umgesetzt. DSpace Manakin

ist die Standardsoftware für das User Interface des im digitalen Filmarchiv genutzten DSpace-Systems. Es beruht auf dem Apache Cocoon Framework und besitzt somit eine Architektur hierarchisch angeordneter XML-Pipelines. Auch die innere Architektur des Dialogue Controllers kann also vollständig mit DSpace Manakin verwirklicht werden. DSpace Manakin verfügt außerdem über ein REST-Interface, welches von den anderen User-Interface-Komponenten für die Kommunikation genutzt werden kann.

Den Part des Dialogue Controllers übernimmt eine konfigurierbare Pipeline (siehe Abb. 5.2). Diese wird über einen HTTP-Request aufgerufen und bindet abhängig von den Request-Parametern mehrere Subpipelines ein. Die Subpipelines generieren ein Präsentationsmodell, nehmen verschiedene Modelltransformationen mittels eines konfigurierbaren XSLT-Themes sowie eine I18n-Transformation für die Mehrsprachigkeit vor und serialisieren den XML-Datenstrom in das Ausgabeformat.

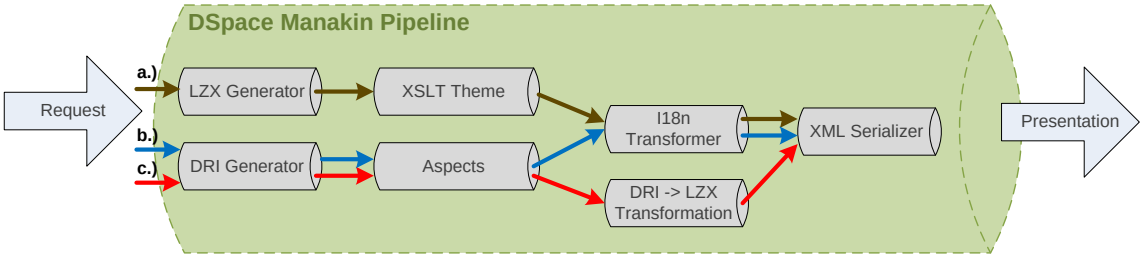

Abbildung 5.1.: DSpace Manakins Pipeline mit Subpipelines für verschiedene Request-Typen:
a.)Erzeugung des UI-Modells
b.)Aktualisierung des Anwendungsmodells
c.)Erzeugung der UI-Teilmodelle aus dem DRI-Modell

Auch sogenannte Aspect-Pipelines können in der Pipeline aufgerufen werden. Sie stellen die Verbindung zu den Anwendungsschnittstellen dar und verkörpern damit den Interface Mapper. Im DSpace Manakin sind bereits mehrere Aspects für den Zugriff auf die Java API des im „Digitalen Filmarchivs“ als Anwendungskomponente realisierten DSpace-Archivsystems implementiert.

Als Basisschema für den XML-Datenstrom nutzt DSpace Manakin ursprünglich das DRI-Format [Dig05]. Das DRI-Dokument stellt zwar auch eine Art Präsentationsmodell dar, enthält jedoch im Sinne der klassischen Webseite immer die Repräsentation einer statischen Seite einschließlich deren Anwendungsdaten. Außerdem ist das DRI-Schema domänenspezifisch, es umfasst nur die UI-Elemente eines herkömmlichen Archivsystems. Um die Architekturvorgaben für die Umsetzung des User In-

terfaces als dynamische Webanwendung zu erfüllen, muss DSpace Manakin also ein Präsentationsmodell auf Basis von OpenLaszlos LZX-Schema erzeugen.

DSpace, DSpace Manakin und Apache Cocoon sind unter Open-Source-Lizenzen zur freien Verwendung verfügbar und beruhen auf plattformunabhängiger Java-Technologie.

5.2. Erstellung des Prototypen

Auf Basis der entworfenen Architektur und mit den ausgewählten Softwarekomponenten lässt sich nun ein Prototyp des User Interfaces für das „Digitale Filmarchiv“ erarbeiten. Ziel des Prototypen ist es, demonstrativ einige grundlegende Funktionen des User Interfaces zu verwirklichen. Als Anwendungskomponente wird dabei vorerst das DSpace-Archivsystem ausgewählt. Mit dem Prototyp des User Interfaces soll es zunächst möglich sein, im Archiv zu stöbern, Filme zu suchen, Informationen und eine Videovorschau zu einzelnen Filmen anzuzeigen, und sich als Nutzer im DSpace-Archivsystem zu authentifizieren. Ein Teil dieser Funktionalität ist bereits im für die Usability-Evaluation erstellten Vorgänger-Prototyp vorhanden und wird nun als UI-Modul erneut eingebracht. Zusätzliche LZX-Teilmodelle des User Interfaces können im Prototyp automatisch aus dem DRI-Modell generiert und als UI-Modul zur Verfügung gestellt werden. Außerdem soll die Kommunikation zwischen SWF-Anwendung und DSpace Manakin nun direkt erfolgen, damit effizienter und schneller werden und eine echte Nutzerauthentifizierung ermöglichen.

5.2.1. Technisches Konzept

Das technische Konzept soll einen kurzen Überblick über das geplante Prototyp-System geben. Zentrales Element ist der Apache Tomcat Webserver. Im Apache Tomcat Server sind die Servlets von DSpace, DSpace Manakin und OpenLaszlo beheimatet. DSpace übernimmt die Archiv- und Nutzerverwaltung. Das um ein LZX-Theme erweiterte DSpace Manakin erzeugt das DRI-Modell sowie das LZX-basierte UI-Modell. OpenLaszlo wandelt das LZX-basierte UI-Modell in SWF-Bytecode um, welcher dann im Flash-Player-Plugin des Webbrowsers auf Nutzerseite interpretiert wird. Darstellungsänderungen werden vom Flash Player clientseitig verarbeitet. Um

das Anwendungsmodell zu aktualisieren, kommuniziert der Flash Player direkt mit DSpace Manakin.

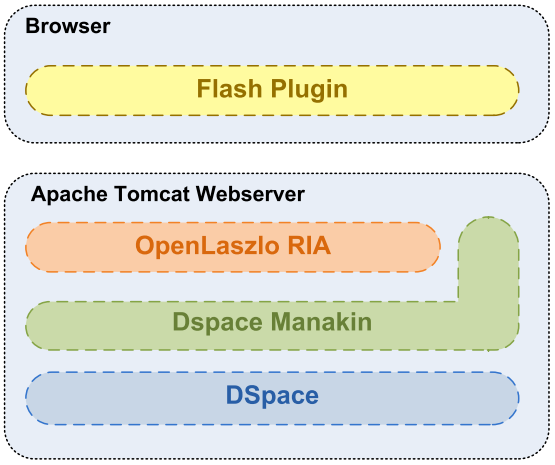

Abbildung 5.2.: Die Anordnung der Softwarekomponenten des Prototypen

5.2.2. Konzept für das User-Interface-Modell

Im LZX-basierten Modell des User Interfaces müssen alle Informationen zur vollständigen Darstellung des gesamten User Interfaces der Webanwendung enthalten sein.

Die Grundstruktur des LZX-basierten UI-Modells gestaltet sich folgendermaßen: UI-Elemente werden in Objekten beschrieben. Die Elemente der Grundstruktur beruhen auf den Klassendefinitionen für Standardobjekte des OpenLaszlo-Frameworks, viele Klassen werden direkt genutzt, manche Elemente ergeben sich aus Erweiterungen der bestehenden Klassen. Auf höchster Ebene befindet sich die Canvas. Sie enthält Elemente, die Auswirkungen auf die komplette Webanwendung haben: Globale Attribute (attributes) und Methoden (methods), Datenquellen (datasets), Ressourcen (resources), Style-Vorgaben (styles, stylesheets) und die Klassendeklarationen vordefinierter UI-Elemente (classes).

Ebenfalls auf der Canvas-Ebene befinden sich die Wurzelelemente der Seitenstruktur. Zum einen ist dies die Portalansicht (portal), welche das gesamte Browserfenster ausfüllt. Neben der Portalansicht existieren unabhängige Overlays, die es ermöglichen, die Portalansicht mit beliebigen Darstellungsformen innerhalb des Browserfensters zu überlagern. Zwei vorgefertigte Overlays, der Wizard (u.a. für den Bestellvorgang

konzipiert) und das Screenwindow (Element des Login-Fensters) sind Basisbestandteile der konzipierten Webanwendung.

Die Portalansicht untergliedert sich hierarchisch in immer feinere Strukturen. Innerhalb der Portalansicht wird in einen Header- und einen Tab-Bereich unterschieden. Im Header-Bereich befindet sich die Seitenkennung (Site-ID) und das Utilities-Menü. Im Tab-Bereich stehen in einem Tab-Menü mehrere Reiter (tabs) mit deren Fensterinhalten zur Auswahl. Diese stellen die primäre Navigation dar, eine sekundäre Navigation kann optional in einem Tab-Fenster (tabpane) als Submenü platziert werden. Innerhalb eines Tab- oder Submenü-Fensters stehen weitere Basisstrukturen für die Anzeige der Inhalte bereit. So können zum Beispiel frei anordbare Fenster (windows) und vertikal angeordnete Widgets genutzt werden.

Abbildung 5.3.: Die Struktur der Portalansicht nach [Sto08]

Die im Konzept für das webbasierte User Interface beschriebene Seitenstruktur wurde damit umgesetzt. Innerhalb der vorhandenen Struktur können neue Elemente definiert oder weitere Standardelemente des OpenLaszlo-Frameworks integriert werden.

Ein weiterer Punkt im Konzept des UI-Modells ist das dynamische Verhalten der Anwendung. Dynamische Änderungen der Darstellung sind über Methodenaufrufe möglich. Methodenaufrufe werden durch Events ausgelöst. Events sind Ereignisse, die unter anderem auch Benutzerinteraktionen beschreiben. Beispielsweise wird

durch einen Mausklick auf ein bestimmtes Element eine Methode aufgerufenen, deren Script die aktuelle Ansicht eines Elementes in der Seitenstruktur auswählt.

Auch das Anwendungsmodell soll dynamisch abgefragt und aktualisiert werden. Das Anwendungsmodell wird im UI-Modell durch Datasets und Datapaths repräsentiert. Jedem Dataset ist eine eindeutige HTTP-Request-URI zugewiesen, die beim Zugriff auf das Dataset abgefragt wird (siehe Listing 5.1 und 5.2). Als Antwort wird ein XML-Datenstrom empfangen und im Dataset gespeichert. Datapaths verweisen dann mit Hilfe eines XPath-Ausdrucks auf den Teil des Datasets, der die darzustellenden Informationen enthält. Im UI-Element ist also anstelle der tatsächlichen Information des Anwendungsmodells ein Datapath referenziert, der nur den Weg zu den Informationen beschreibt. Ändert sich das im Dataset gespeicherte XML-Dokument, aktualisieren sich automatisch auch die betroffenen Bereiche der Darstellung mit diesen Informationen.

Listing 5.1: Definition eines Datasets

```
1 <dataset type="http" proxied="false" request="true" src="${canvas.
      dri + 'browse-title'}" name="manakin_browse_title"/>
2 <dataset type="http" proxied="false" request="true" src="${canvas.
      dri + 'browse-subject'}" name="manakin_browse_subject"/>
```

Listing 5.2: Methode für die Aktualisierung des Datasets

```
1 <method args="context, refpage, refnodedataset" name="updateData">
2
3   // remove any path leftovers
4   context = context.split("?");
5   context = '?' + context[1];
6
7   // set the new source for the dataset
8   refnodedataset.setSrc(canvas.dri + refpage + context);
9   Debug.write(refnodedataset.src);
10
11  // refresh the dataset
12  refnodedataset.doRequest();
13
14 </method>
```

Auf den vollständigen Quellcode des LZX-basierten UI-Modells wird in Abschnitt C.2 im Anhang dieser Arbeit verwiesen.

5.2.3. Transformationen des User-Interface-Modells

Für die Transformation des User Interface Modells wurden zwei exemplarische Beispielvorgänge ausgewählt, welche für die Implementierung des User Interfaces von großer Bedeutung sind. An erster Stelle steht die komplette Transformation des User-Interface-Modells. Zwar sind im Architekturentwurf nur schrittweise partielle Transformationen angedacht, jedoch ist es für den konkreten Implementierungsfall von hohem Interesse, das DRI-Modell des DSpace Manakin User Interfaces in das LZX-Modell des OpenLaszlo RIA-Frameworks umzuwandeln. So sollen verschiedene DRI-Seiten in mehrere LZX-Teilmodelle konvertiert werden. Das Ergebnis dieser Transformation kann dann als Grundlage für die zweite Transformation genutzt werden. Diese soll exemplarisch eine Vereinigung verschiedener User-Interface-Teilmodelle aus den Plugin-Modulen darstellen. Die Transformation erfolgt in beiden Fällen mit Hilfe einer XSL Transformation.

Erzeugung des LZX-Modells aus dem DRI-Modell

Auf Basis des im DSpace Manakin erzeugten DRI-Dokuments (siehe Abschnitt C.1 im Anhang dieser Arbeit) wird nun ein eigenes Vorgehen für die Erstellung des LZX-basierten UI-Modells im Dialogue Controller entwickelt. Das DRI-Dokument eignet sich als Grundlage, da es die Informationen zwar für eine gesamte Seite wiedergibt, inhaltlich jedoch eine Trennung in drei Teile vornimmt: Ein Präsentationsteil, in dem UI-Elemente und die Seitenstruktur enthalten sind, einen Steuerungsteil mit den Eingabeoptionen des Nutzers und einen Anwendungsteil, der Metadaten des Archivs und der Archivinhalte widerspiegelt.

Das angestrebte LZX-basierte Modell unterscheidet sich in wesentlichen Punkten vom DRI-Modell. Das DRI-Modell repräsentiert den kompletten statischen Inhalt einer einzelnen Webseite. Im LZX-Modell jedoch soll das UI-Modell der kompletten Webanwendung mit der Möglichkeit des dynamischen Nachladens von Anwendungsdaten enthalten sein. Weiterhin soll mit dem LZX-Modell das bestehende Konzept für die Gestaltung des User Interfaces des „Digitalen Filmarchivs“ eingebracht werden. Somit ist eine Transformation nötig, welche neben der Konvertierung der Beschreibungssprache und der UI-Elemente die neue Seitenstruktur sowie die Möglichkeit der dynamischen Aktualisierung von Anwendungsmodell und Darstellung in das LZX-Modell einbringt.

Anhand einer DSpace-Seite soll diese Transformation beispielhaft implementiert werden. Als Vorlage dient das DRI-Modell der „Browse by title“-Seite des „Digitalen Filmarchivs“. Standardmäßig erzeugt DSpace Manakin mit Hilfe der XSL-Transformationen im XHTML-Theme eine XHTML-Seite (siehe Abb. 5.4) aus dem DRI-Modell.

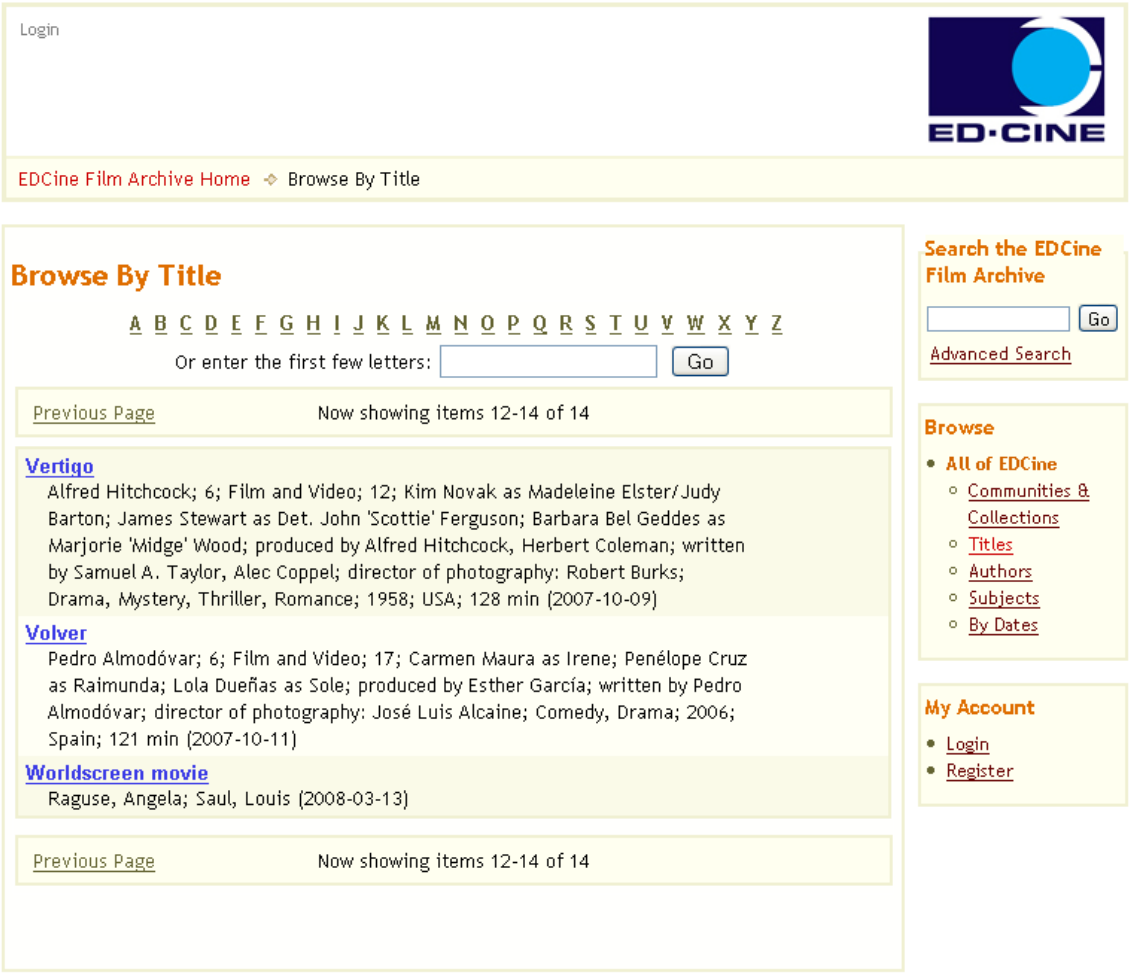

Abbildung 5.4.: Browserdarstellung der XHTML-Ausgabe der „Browse by title“-Seite des DSpace Manakin

Das XHTML-Theme dient als Vorlage für das neue LZX-Theme (siehe Abschnitt C.1 im Anhang dieser Arbeit). Die enthaltenen XSL-Transformationen mit deren XSL-Templateregeln und XPath-Ausdrücken können größtenteils weiter genutzt werden, um auf die verschiedenen Bereiche des DRI-Dokumentes zuzugreifen. Die Anordnung der Templateregeln sowie die Struktur und die Elemente des UI-Modells müssen an das LZX-basierte Modell angepasst und entsprechend erweitert werden.

Um den dynamischen Zugriff auf das Anwendungsmodell zu ermöglichen, wird ein

Dataset mit einem Verweis auf die HTTP-Request-URI des aktuellen DRI-Dokumentes gesetzt. Direkte Informationen aus dem Anwendungsmodell werden nicht in das UI-Modell übertragen, an deren Stelle steht ein Datapath mit einem XPath-Verweis auf den entsprechenden Bereich des Datasets.

Als Beispiel wird kurz die Auswahlliste des Anfangsbuchstabens auf der „Browse by title“-Seite betrachtet. Listing 5.3 zeigt die Liste (list) im entsprechenden DRI-Modell. Darin enthalten ist neben dem Buchstaben (item) der veränderte Teil der HTTP-Request-URI (target) für den Aufruf der neuen Seite.

Listing 5.3: Auswahlliste im DRI-Modell

```
<?xml version="1.0" encoding="UTF-8"?>
<document xmlns="http://di.tamu.edu/DRI/1.0/" xmlns:i18n="http://
    apache.org/cocoon/i18n/2.1" version="1.0">
  <body>
    <div rend="primary" n="browse-by-title" id="artifactbrowser.
        BrowseTitles.div.browse-by-title">
      <!-- ... -->
      <div interactive="yes" rend="secondary navigation" action="
          browse-title" n="browse-navigation" method="post" id="
          artifactbrowser.BrowseTitles.div.browse-navigation">
        <list rend="alphabet" type="simple" n="jump-list" id="
            artifactbrowser.BrowseTitles.list.jump-list">
          <item>
            <xref target="browse-title?startsWith=A">A</xref>
          </item>
          <item>
            <xref target="browse-title?startsWith=B">B</xref>
          </item>
          <!-- ... -->
        </list>
        <!-- ... -->
      </div>
      <!-- ... -->
    </div>
  </body>
  <!-- ... -->
</document>
```

Mit Hilfe des XSL-Templates in Listing 5.4 wird die DRI-Liste in das LZX-Modell umgewandelt. Der Inhalt der Auswahlliste selbst wird mit einem Datapath referenziert und der Event-Handler „onclick“ hinzugefügt, welcher ein Update der Quelle

des Datasets für die „Browse by title“-Seite bei Mausklick auf einen Buchstaben durchführt.

Listing 5.4: XSL-Template für die Transformation der Auswahlliste

```
<xsl:template match="dri:body/dri:div/dri:div/dri:list[@type='
    simple']" priority="5">
    <view align="center">
    <simplelayout axis="x" spacing="0"/>
    <text>
      <!-- create a text box for each text item matching the
          datapaths result -->
      <xsl:attribute name="datapath">
          <xsl:value-of select="$page_data"/>
          <xsl:text>:/document/body/div/div/list/item/xref/text()</
              xsl:text>
      </xsl:attribute>
      <!-- create an onclick event for this text box to update the
          page data -->
      <xsl:attribute name="onclick">
            <xsl:text>canvas.updateData(this.datapath.xpathQuery('
                @target'), '</xsl:text>
            <xsl:value-of select="$page_context"/>
        <xsl:text>', canvas.</xsl:text>
        <xsl:value-of select="$page_data"/>
        <xsl:text>)</xsl:text>
          </xsl:attribute>
    </text>
  </view>
</xsl:template>
```

Das LZX-Modell in Listing 5.5 enthält nun lediglich den Verweis auf das Anwendungsmodell. Für jeden einzelnen Buchstaben im Anwendungsmodell wird später ein Listeneintrag (text) erzeugt. Erfolgt die Auswahl eines Anfangsbuchstabens, wird mittels des Event-Handlers die HTTP-Request-URI des Datasets mit dem neuen Parameter aktualisiert (siehe updateData-Methode in Listing 5.2) und abgefragt. Alle mit diesem Dataset verknüpften Datapaths erhalten dann eine Aktualisierung ihrer Inhalte.

Listing 5.5: LZX-Modell der Auswahlliste

```
<view align="center">
  <simplelayout spacing="0" axis="x"/>
  <text datapath="manakin_browse_title:/document/body/div/div/list/
      item/xref/text()" onclick="canvas.updateData(this.datapath.
```

```
        xpathQuery('@target'), 'browse-title', canvas.
        manakin_browse_title)"/>
4 </view>
```

Auf den vollständigen Quellcode der DRI-Dokumente, der XSL-Transformation des LZX-Themes und zwei automatisch generierte LZX-Teilmodelle wird im Abschnitt C im Anhang dieser Arbeit verwiesen.

Merge-Transformation

Nach der Transformation des DRI-Modells in das LZX-Modell sind aus den einzelnen DRI-Seiten LZX-Teilmodelle entstanden, welche zum gesamten UI-Modell vereint werden müssen. Hierfür wird in diesem Abschnitt eine exemplarische XSL-Transformation mit dem Namen Merge-Transformation (siehe Abschnitt C.2 im Anhang dieser Arbeit) entworfen, mit deren Hilfe die UI-Teilmodelle der einzelnen UI-Plugins im Module Handler zusammengeführt werden können.

Nachdem im vorangegangenen Abschnitt bereits das Neuladen der aktuellen Seite durch dynamische Vorgänge ersetzt wurde, muss jetzt auch der Wechsel auf andere Seiten mit einer dynamischen Aktualisierung der Darstellung innerhalb der Webanwendung vollzogen werden. Üblicherweise wird sich ein Teilmodell in verschiedene Menüs auf unterschiedlichen Ebenen integrieren. Der Inhalt des Teilmodells wird dann über das Benutzerereignis „Menüpunkt ausgewählt“ aufgerufen und dargestellt. An bestimmten Stellen können aber auch Teilmodelle integriert werden, welche keinen Menüeintrag benötigen und über andere Ereignisse aufgerufen werden. Wird ein Menüpunkt ausgewählt, erfolgt die Darstellung des Inhalts und damit eine Aktualisierung üblicherweise im untergeordneten Teilbereich der Webanwendung. Es ist jedoch auch möglich, an jeder Stelle über ein Ereignis auf andere Ebenen zu springen, um eine abweichende Darstellung zu erzielen.

Das Zusammenführen der Teilmodelle findet also auf den verschiedenen Ebenen der Seitenstruktur der Webanwendung statt. Auf der Canvas-Ebene werden Attribute, Datenquellen, Style-Vorgaben, Ressourcen, Klassendeklarationen und Overlays mit unterschiedlichem Namen additiv vereint. In der Portalansicht lässt sich das Utilities-Menü um Einträge erweitern. Auch dem Tab-Menü oder einem Submenü können weitere Menüeinträge hinzugefügt werden.

In der Merge-Transformation (siehe Abb. 5.5) werden die Elementknoten (element nodes) auf ihren jeweiligen Hierarchieebenen (node levels) zusammengeführt. Das Zusammenführen (merge) der Elemente erfolgt nach verschiedenen Mustern, ausgehend vom Canvas-Knoten (canvas node): Elemente mit unterschiedlichem Bezeichner oder verschiedenem name-Attribut werden mitsamt ihrer Attribute separat kopiert. Elemente mit gleichem Bezeichner und ohne name-Attribut oder mit übereinstimmendem name-Attribut werden additiv vereint, d.h. fehlende untergeordnete Attribute werden ergänzt und erhalten einen gemeinsamen Elternknoten. Treten gleiche Attribute mit verschiedenen Werten auf, bleibt also der Wert des zuerst aufgetretenen Attributs im Ergebnisdokument. Für die untergeordneten Elemente muss entschieden werden, ob ein Element mitsamt des untergeordneten Baums kopiert werden kann (recursive copy) oder eine erneute Zusammenführung des Elementes auf der tieferen Knotenebene (subelement) stattfinden soll. Zwei Parameter der Transformation mit jeweils einer Liste an Elementbezeichnern bestimmen, mit welchem Element wie verfahren wird und in welcher Reihenfolge die Elemente abgearbeitet werden. Um mehr als zwei Teilmodelle vereinen zu können, wird das Ergebnis des Zusammenführens in der result-Variable gespeichert. Folgt ein weiteres Teilmodell, wird das Template mit der result-Variable und dem Teilmodell erneut durchlaufen. Mit dieser Transformation ist es jetzt möglich, das UI-Modell aus einer beliebigen Anzahl an Plugin-Modulen automatisch zu vereinen.

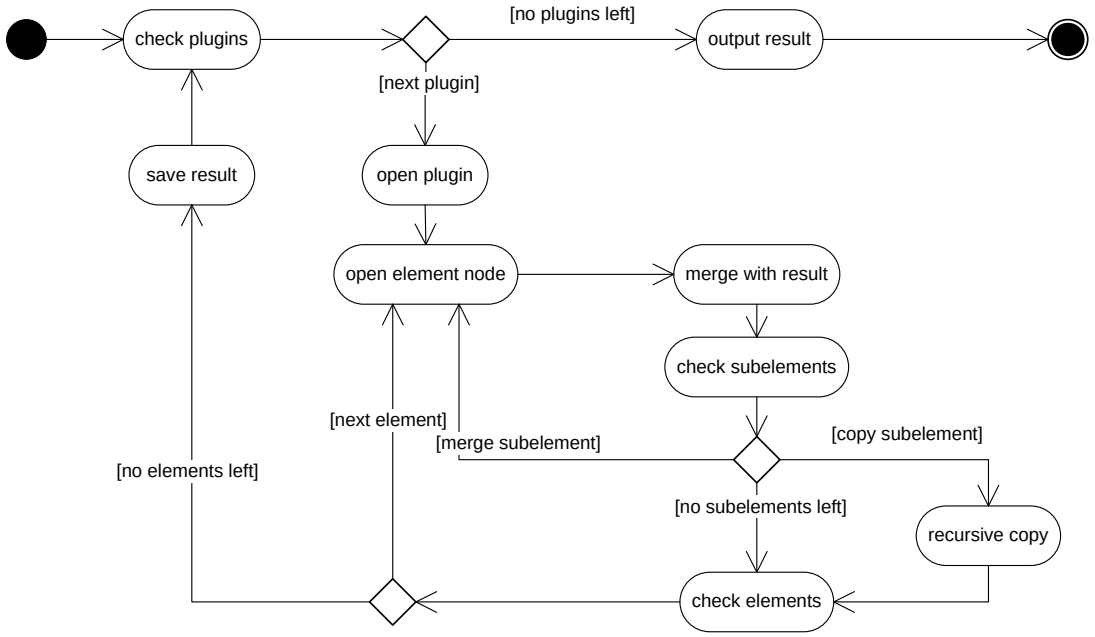

Abbildung 5.5.: Merge-Transformation zum Zusammenführen der UI-Teilmodelle aus den UI-Plugins

5.2.4. Nutzerauthentifizierung über HTTP-Sessions

Die Kommunikation zwischen SWF-Anwendung und DSpace Manakin soll in diesem Prototyp für den Austausch der Anwendungsdaten direkt erfolgen. Im Usability-Prototyp wurden HTTP-Requests an das DSpace Manakin über den Proxy des OpenLaszlo-Servers ausgeführt, was zu Leistungseinbußen und Schwierigkeiten bei der Verwaltung der HTTP-Session eines Nutzers führte. Nun wird der HTTP-Request ohne diesen Umweg an DSpace Manakin gesendet und die Antwort mit Hilfe der XPath-Funktionen des Flash-Players ausgewertet. Dadurch wird in diesem Prototyp auch eine echte Authentifizierung im DSpace-System möglich.

DSpace Manakin identifiziert einen Nutzer über eine Session-ID. Beim Initialisieren der Webanwendung werden Daten vom Anwendungsmodell geladen, damit erfolgt eine erste direkte Kommunikation zwischen der Webanwendung und DSpace Manakin. In Folge des ersten HTTP-Requests an DSpace Manakin innerhalb einer Browsersession erstellt DSpace-Manakin eine eindeutige Session-ID (JSESSIONID) und gibt diese in den HTTP-Response-Headern der Antwort zurück (siehe Listing B.1 im Anhang dieser Arbeit). Alle zukünftigen HTTP-Requests der Webanwendung werden mit Angabe der zugeteilten Session-ID im HTTP-Request-Header gestellt. Somit wird eine persistente Wiedererkennung des Nutzers im DSpace Manakin ermöglicht.

Die Kommunikation zwischen DSpace Manakin und der Webanwendung wird nun für das Beispiel der Anmeldung eines Nutzers im DSpace-System dargestellt. Klickt der Nutzer den Login-Button im Utility-Menü der Webanwendung, öffnet sich das Login-Fenster mit den Eingabefeldern für die Login-Daten (siehe Abb. 5.6).

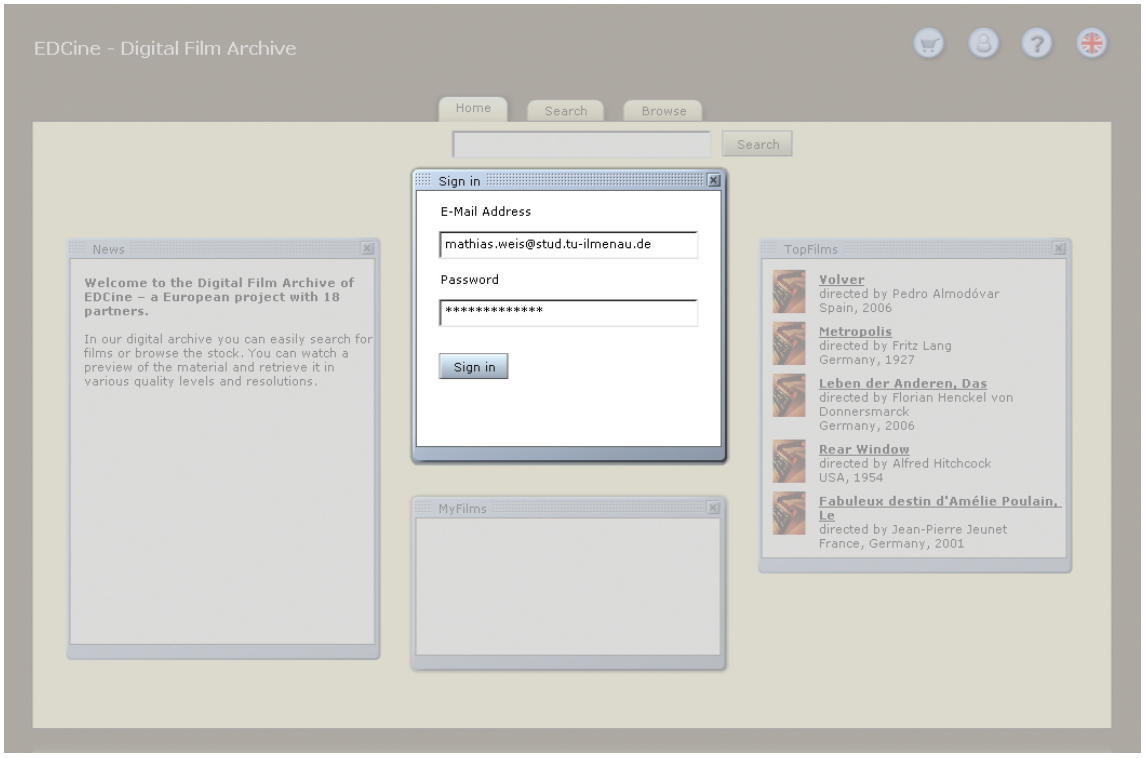

Abbildung 5.6.: Login-Fenster im User Interface des „Digitalen Filmarchivs“

Hat der Nutzer die entsprechenden Daten in die Felder eingetragen und das Formular abgesendet, veranlasst die dem Login-Fenster zugeordnete Login-Methode (siehe Listing 5.6) das Anmelden des Benutzers im DSpace Manakin.

Listing 5.6: Die Login-Methode

```
1  <method name="loginUser">
2  <![CDATA[
3
4    // set the query parameters for the datasets request URI
5    canvas.manakin_login.setQueryType('POST');
6    canvas.manakin_login.setQueryParam('login_email', this.
         login_email.getValue());
7    canvas.manakin_login.setQueryParam('login_password',this.
         login_password.getValue());
8
9    // update the dataset with a request
10   canvas.manakin_login.doRequest();
11
12   // check if the user is logged in now
13   if (manakin_login.datapath.xpathQuery('meta/userMeta/
         @authenticated') == "yes"){
14     parent.close();
```

```
15      canvas.portal.head.buttons.login.button.ani.doStart();
16      this.login_password.clearText();
17    }
18 ]]>
19 </method>
```

In dieser Methode werden die Formularfelder (login_email, login_password) ausgelesen und als POST-Parameter dem HTTP-Request des Login-Datasets zugewiesen. Danach wird das Login-Dataset (manakin_login) aktualisiert und damit ein HTTP-Request (Headerdaten siehe Listing B.2 im Anhang dieser Arbeit) mit den entsprechenden Parametern durchgeführt. Nach erfolgreicher Anmeldung wird das Login-Fenster geschlossen. Der Nutzer wird nun anhand der Session-ID und des vom DSpace Manakin gespeicherten Authentifizierungsstatus für diese Session-ID als angemeldeter Nutzer wiedererkannt.

Der vollständige Quellcode für die Nutzerauthentifizierung ist im Hauptteil des UI-Modells zu finden, auf das im Abschnitt C.2 im Anhang dieser Arbeit verwiesen wird.

5.3. Weitere Aufgaben der Implementierung

Da die Implementierung im Rahmen dieser Arbeit nur beispielhaft erfolgte, wird nun ein Überblick über weitergehende Schritte in der Implementierung gegeben. Um alle Funktionen des DSpace-Systems im User Interface zu unterstützen, müssen weitere Seiten des DRI-Modells in LZX-basierte Teilmodelle konvertiert werden. Mit jeder neuen Seite erfolgt dabei eine schrittweise Optimierung und ein Test der vorhandenen XSL-basierten Modelltransformation. Gleichzeitig kann jedes neu erzeugte LZX-Teilmodell mit der Merge-Transformation in das gesamte UI-Modell integriert werden. Auch dabei sind eine schrittweise Optimierung sowie wiederholte Tests der Transformation notwendig.

Für das Plugin-System des User Interfaces wurde bisher nur der wesentliche Teil des Zusammenführens der UI-Modelle betrachtet. Somit bleibt noch die Umsetzung zu einem echten Plugin-System mit einer Konfigurationsoberfläche, Konfigurationseinstellungen, dynamischen Plugin-Registry-Pipelines und einem erweiterbaren Wörterbuch noch offen.

Ist das Plugin-System ausgereift, können weitere Anwendungskomponenten integriert werden. Hierfür ist eventuell eine Optimierung des Interface Mappers für weitere Schnittstellen neben der DSpace API notwendig. Möglicherweise müssen auch die momentan im DSpace integrierten und gemeinsam genutzten User-Interface-bezogenen Anwendungsfunktionen, wie etwa die Nutzerauthentifizierung, die Speicherung des Status und die Versionisierung von UI-Aktivitäten und die Personalisierung erweitert oder in eine neue Komponente ausgelagert werden.

Kapitel 6.

Zusammenfassung

In dieser Arbeit wurde eine Softwarearchitektur für das User Interface des „Digitalen Filmarchivs“ entwickelt. Dabei wurden verschiedene Phasen des Softwareentwicklungsprozesses durchlaufen. Zuerst wurde geklärt, welche Ziele mit der Entwicklung des User Interfaces erreicht werden sollen. Aus den Zielen wurden dann die Anforderungen für das User Interface abgeleitet. Einige exemplarische Anwendungsszenarien für das User Interface wurden ebenfalls aufgestellt.

Der Schwerpunkt der Arbeit lag jedoch auf dem Entwurf der Architektur für das User Interface. Im Rahmen der Vorbetrachtungen zum Entwurf wurde gezeigt, wie verschiedene Architekturstile, -muster und Referenzarchitekturen auf mehreren Abstraktionsebenen eingesetzt werden können, um das User Interface in mehrere Komponenten zu unterteilen, diese in einer vorteilhaften Struktur anzuordnen, und so einen optimalen Architekturentwurf für das User Interface zu gestalten. Für die Kommunikation zwischen den Komponenten wurde die Eignung der REST-Architektur in Verbindung mit Ajax-Technologie begründet. Es wurde zudem gezeigt, welche positiven Auswirkungen auf die Erfüllung der Anforderungen bei Anwendung des PAC-AMODEUS-Musters auf die Architektur des User Interfaces entstehen. Mit der Nutzung eines XML-basierten UI-Modells und der Einbindung von XSL-Transformationen zur Anpassung des UI-Modells wurden die Vorstellungen von einem generischen User Interface weiter konkretisiert. Auf tieferen Architekturebenen erwiesen sich Referenzarchitekturen wie die XML-Pipeline oder verschiedene Plugin-Architekturen als hilfreich. Es erfolgte eine ausführliche Beschreibung des Architekturentwurfes, Funktion und Anordnung der Komponenten wurden umfassend dokumentiert. Auch konnte die Architektur des User Interfaces erfolgreich in die Systemarchitektur eingebunden werden. Eine kurze Architekturbewertung bescheinigte die Erfüllung der Anforderungen und eine Eignung der Architektur für das User Interface des „Digi-

talen Filmarchivs“.

Im Rahmen der beispielhaften Implementierung wurde ein Prototyp-System mit dem User Interface einer Anwendungskomponente des „Digitalen Filmarchivs“ umgesetzt. Einige Funktionen der Anwendungskomponente wurden in das User Interface eingebracht und getestet. Mit zwei XSL-Transformationen wurde der Grundstein für die automatische Konvertierung und Zusammenführung des UI-Modells auf dem Prototyp-System gelegt.

6.1. Kritische Reflexion

Bei der Erarbeitung der Softwarearchitektur traten auch einige Probleme auf. Die Definition der Anforderungen konnte durch die gewünschte generische Ausprägung des User Interfaces an vielen Stellen nur oberflächlich erfolgen. So wurde nur ein kleiner Teil der konkreten Funktionen beispielhaft erwähnt und in Anwendungsszenarien übertragen, detaillierte und messbare Angaben zur Ausprägung der Qualitätsattribute konnten ebenfalls kaum in den Anforderungen festgehalten werden. Die Erfüllung der Anforderungen in einem generischen System zu garantieren und zu überprüfen stellt sich gegenüber einem spezifischen System wesentlich schwieriger dar. Die Architektur in einem generischen System tendiert dazu, den Charakter einer Referenzarchitektur für eine Produktlinie anzunehmen. Gleiches gilt für die nicht genau definierbaren organisatorische Rahmenbedingungen. Hätte beispielsweise ein Entwicklungsteam mit bestimmten Fähigkeiten für die Umsetzung festgestanden, hätte dies sicherlich auch Auswirkungen auf den Architekturentwurf gehabt.

Weitere Hürden waren bei der Betrachtung der verschiedenen Architekturmuster für interaktive Systeme zu überwinden. So existieren vor allem im Umfeld des MVC- und des PAC-Musters eine Vielzahl von Variationen abgeleiteter Modelle. Die Nachvollziehbarkeit dieser Modelle wird wird jedoch oft dadurch gemindert, dass in den Herleitungen unterschiedliche Auffassungen vom Ursprungsmodell vertreten werden. Auch wurden manche in ihrer Struktur übereinstimmende Modelle entdeckt, welche keinen Bezug zueinander herstellten und unter verschiedenen Namen existieren. Daher wurden diese, oft aktuelleren Modelle, nicht näher betrachtet und sich lediglich auf das Ursprungsmodell berufen. Für komplexe interaktive Systeme existieren leider kaum geeignete Architekturmuster. Vor allem Muster, welche User Interfaces für verschiedene Interface-Typen jenseits der GUI einschließen waren nicht vorhan-

den.

Die Komplexität des in dieser Arbeit betrachteten Systems ist recht umfangreich. Dies führte zu einer sehr beispielhaften Implementierung. Als Anwendungskomponente war zum Implementierungszeitpunkt lediglich das Archivsystems DSpace vorhanden. Damit waren die Möglichkeiten zur Demonstration der generischen Eigenschaften des Systems gegenüber weiteren Anwendungskomponenten in der Implementierung stark eingeschränkt. Für die noch geplanten Anwendungskomponenten konnte nur eine theoretische Bewertung auf Basis des Architekturentwurfs stattfinden.

6.2. Ausblick

Standardisierte UI-Beschreibungssprachen und Referenzmodelle sind zumeist für grafische User Interfaces ausgelegt. Hier wäre ein internationaler Standard für eine allgemeinere UI-Beschreibungssprache wünschenswert, die sich auch auf andere Typen von User Interfaces ausweiten lässt. Zudem müssten plattformunabhängige und weit verbreitete Werkzeuge zur Darstellung und Bedienung des User Interfaces existieren, welche in der Lage sind, ohne großen Aufwand und in einheitlicher Form ein in der standardisierten Beschreibungssprache verfasstes User-Interface-Modell zu interpretieren. So könnte die aufwändige Transformation des UI-Modells in den UI-Code umgangen werden. Neben der Entkopplung des Anwendungsmodells und des Style-Modells vom eigentlichen UI-Modell könnte in weitere Modelle, wie etwa in ein Nutzer- oder Aufgabenmodell, unterschieden werden. In diesen Bereichen wären dann ebenfalls unabhängige und gezielte Aktualisierungen der Teilmodelle möglich.

Literaturverzeichnis

[Ado08a] Adobe Systems Incorporated (Hrsg.): *Adobe Flash Player Version Penetration*. http://www.adobe.com/products/player_census/flashplayer/version_penetration.html. Version: 2008, Abruf: 20.04.2008

[Ado08b] Adobe Systems Incorporated (Hrsg.): *Flash Player Penetration: Flash content reaches over 98% of Internet viewers*. http://www.adobe.com/products/player_census/flashplayer/. Version: 2008, Abruf: 20.04.2008

[AH03] Abdelnur, Alejandro ; Hepper, Stefan ; Sun Microsystems Inc. (Hrsg.): *Java(TM) Portlet Specification Version 1.0*. http://jcp.org/aboutJava/communityprocess/final/jsr168/index.html. Version: 2003, Abruf: 15.05.2008

[AISJ05] Alexander, Christopher ; Ishikawa, Sara ; Silverstein, Murray ; Jacobson, Max: *Center for Environmental Structure series*. Bd. 2: *A pattern language: Towns, buildings, construction*. 27. Aufl. New York : Oxford Univ. Press, 2005

[BC05] Bergh, J. Van d. ; Coninx, K.: Towards integrated design of context-sensitive interactive systems. In: *Proceedings of the 3rd International Conference on Pervasive Computing and Communications Workshops* (2005), S. 30–34

[BCK02] Bass, Len ; Clements, Paul ; Kazman, Rick: *Software architecture*

in practice. 9. Aufl. Boston : Addison–Wesley, 2002

[BJ01] Bass, L. ; John, B.E.: Supporting usability through software architecture. In: *Computer* Nr. 34 (2001), Nr. 10, S. 113–115

[Bol03] Bolour, Azad: *Notes on the Eclipse Plug–in Architecture*. http://www.eclipse.org/articles/Article-Plug-in-architecture/plugin_architecture.html. Version: 2003, Abruf: 15.05.2008

[Bos00] Bosch, Jan: *Design and use of software architectures: Adopting and evolving a product–line approach*. Harlow : Addison–Wesley, 2000

[Boy07] Boyer, John M. ; W3C (Hrsg.): *XForms 1.0 (Third Edition): W3C Recommendation 29 October 2007*. http://www.w3.org/TR/2007/REC-xforms-20071029/. Version: 2007, Abruf: 20.04.2008

[Bun02] Bundesministerium der Justiz (Hrsg.): *BITV: Verordnung zur Schaffung barrierefreier Informationstechnik nach dem Behindertengleichstellungsgesetz*. http://www.gesetze-im-internet.de/bitv/index.html. Version: 2002, Abruf: 13.04.2008

[Bus07] Buschmann, Frank: *Wiley series in software design patterns*. Bd. 1: *A system of patterns*. Reprinted. Chichester : Wiley, 2007

[CCRV08] Caldwell, Ben ; Cooper, Michael ; Reid, Guarino L. ; Vanderheiden, Gregg ; W3C (Hrsg.): *Web Content Accessibility Guidelines 2.0: W3C Candidate Recommendation 30–April–2008*. http://www.w3.org/TR/2008/CR-WCAG20-20080430/. Version: 2008, Abruf: 15.05.2008

[Chl06] Chlebek, Paul: *User Interface–orientierte Softwarearchitektur: Bauentwurfslehre für interaktive Softwareoberflächen – Kompass für die Entwicklung dialogintensiver Anwendungen – Leitfaden für erlebbare User Interfaces*. 1. Aufl. Wiesbaden : Vieweg, 2006

[CKP00] Cai, Jason ; Kapila, Ranjit ; Pal, Gaurav: *HMVC: The layered pattern for developing strong client tiers*. http://www.javaworld.com/javaworld/jw-07-2000/jw-0721-hmvc.html. Version: 2000, Abruf: 13.04.2008

[Cla99] Clark, James ; W3C (Hrsg.): *XSL Transformations (XSLT) Version 1.0: W3C Recommendation*. http://www.w3.org/TR/1999/REC-xslt-19991116.html. Version: 1999, Abruf: 13.04.2008

[Con02] Consultative Committee for Space Data Systems (CCSDS) (Hrsg.): *Reference Model for an Open Archival Information System (OAIS): CCSDS 650.0–B–1, Blue Book*. 2002. (14721)

[Dig05] Digital Initiatives (Hrsg.): *DRI Schema Reference*. http://di.tamu.edu/projects/xmlui/schemaReference. Version: 2005, Abruf: 24.04.2008

[DIN] Norm DIN 66272: 1994-10 . *Beurteilen von Softwareprodukten*

[Dix04] Dix, Alan J.: *Human–computer interaction*. 3. Aufl. Harlow : Pearson Prentice–Hall, 2004

[DSp] DSpace (Hrsg.): *dspace.org – About DSpace*. http://www.dspace.org, Abruf: 04.03.2008

[EDC07a] EDCine (Hrsg.): *D1.2 Specification of Systems and Workflows*. http://www.edcine.org/documents/public/edcine_tcf_ds_d1_2_v3_ucl.pdf/view. Version: 2007, Abruf: 13.04.2008

[EDC07b] EDCine (Hrsg.): *D6.5 Definition of detailed workflow and format specifications for T6.3*. 2007

[ETS02] ETSI: *Human Factors (HF): Guidelines for ICT products and services: "Design for All"*. V1.2.1. 2002

[ETS03] ETSI: *Human Factors (HF): Multimodal interaction, communication and navigation guidelines*. V1.1.1. 2003

[Fie00] Fielding, Roy T.: *Architectural Styles and the Design of Network–based Software Architectures: Dissertation*. Irvine, University of California, Diss., 2000. http://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf, Abruf: 15.05.2008

[Gar05] Garrett, Jesse J. ; Adaptive Path Inc. (Hrsg.): *Ajax: A New Ap-*

proach to Web Applications. http://www.adaptivepath.com/ideas/essays/archives/000385.php. Version: 2005, Abruf: 14.03.2008

[GH00] Grundy, John ; Hosking, John: Developing Adaptable User Interfaces for Component-Based Systems. In: *AUIC '00: Proceedings of the First Australasian User Interface Conference*. Washington, DC, USA : IEEE Computer Society, 2000, S. 17

[GHHW01] Goodger, Ben ; Hickson, Ian ; Hyatt, David ; Waterson, Chris ; mozilla.org (Hrsg.): *XML User Interface Language (XUL) 1.0*. http://www.mozilla.org/projects/xul/xul.html. Version: 2001, Abruf: 20.04.2008

[GHJV07] Gamma, Erich ; Helm, Richard ; Johnson, Ralph ; Vlissides, John: *Design patterns: Elements of reusable object–oriented software*. 34. Aufl. Boston : Addison–Wesley, 2007

[GSR05] Gomaa, M. ; Salah, A. ; Rahman, S. ; Computer Science Department, North Dakota State University (Hrsg.): *Towards a better model based user interface development environment: A comprehensive survey*. 2005

[Had06] Hadley, Marc J. ; Sun Microsystems Inc. (Hrsg.): *Web Application Description Language (WADL)*. https://wadl.dev.java.net/. Version: 2006, Abruf: 13.04.2008

[HC97] Hussey, A. ; Carrington, D.: Comparing the MVC and PAC architectures: a formal perspective. In: *Software Engineering. IEE Proceedings* Nr. 144 (1997), Nr. 4, S. 224–236

[HL04] Hauser, Tobias ; Löwer, Ulrich M.: *Web–Services: Die Standards*. 1. Aufl. Bonn : Galileo Press, 2004

[IEE00] IEEE Computer Society (Hrsg.): *IEEE Std 1471-2000. IEEE Recommended practice for architectural description of software-intensive systems*. 2000

[Kla] Klaene, Michael ; Jupitermedia Corporation (Hrsg.): *Under-*

standing the Java Portlet Specification. http://www.developer.com/java/web/article.php/3366111, Abruf: 13.04.2008

[KP88] Krasner, G. ; Pope, S.: A Description of the Model–View–Controller User Interface Paradigm in the Smalltalk–80 system. In: *Journal of Object Oriented Programming* (1988), Nr. 3, S. 26–49

[Kru01] Kruchten, Philippe ; Rational Software Canada (Hrsg.): *What is the Rational Unified Process*. http://www.ibm.com/developerworks/rational/library/content/RationalEdge/jan01/WhatIstheRationalUnifiedProcessJan01.pdf. Version: 2001, Abruf: 13.04.2008

[Las06] Laszlo Systems, Inc. (Hrsg.): *OpenLaszlo: An Open Architecture Framework for Advanced Ajax Applications*. http://www.openlaszlo.org/whitepaper/LaszloWhitePaper.pdf. Version: 2006, Abruf: 20.04.2008

[Mic08] Michel, Colin ; EDCine Project (Hrsg.): *The EDCINE vision*. http://www.edcine.org/documents/public/edcine_the_vision.pdf. Version: 2008, Abruf: 13.04.2008

[MML01] Mueller, A. ; Mundt, T. ; Lindner, W.: Using XML to semi–automatically derive user interfaces. In: *User Interfaces to Data Intensive Systems, 2001. UIDIS 2001. Proceedings.* (2001), S. 91–95

[Mol04] Molina, Pedro J. ; CARE Technologies S.A. (Hrsg.): *A Review to Model–Based User Interface Development Technology*. http://ftp.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-103/molina-moreno.pdf. Version: 2004, Abruf: 13.04.2008

[NC91] Nigay, L. ; Coutaz, J.: Building User Interfaces: Organizing Software Agents. In: *Esprit'91 Conference Proceedings* (1991)

[NF07] Nowak, Arne ; Fössel, Siegfried: Digitale Filmarchive – Chancen und Risiken. In: *Tagungsband, Europäische Tagung zur Medienproduktion, 25.1. – 28.1.2007, Ilmenau* (2007)

[NFNS07] Nowak, Arne ; Fössel, Siegfried ; Nunes, Luís (MOG Solutions Por-

tugal) ; SANTOS, Ernesto (MOG Solutions Portugal): A System Architecture for Digital Film Archives using JPEG2000 and MXF. In: *Proceedings, International Broadcast Convention (IBC), Sept. 6 – 11, 2007, Amsterdam, The Netherlands* (2007)

[PBG07] POSCH, Torsten ; BIRKEN, Klaus ; GERDOM, Michael: *Basiswissen Softwarearchitektur: Verstehen, entwerfen, wiederverwenden*. 2. Aufl. Heidelberg : dpunkt–Verl., 2007

[Poh07] POHL, Klaus: *Requirements engineering: Grundlagen, Prinzipien, Techniken*. 1. Aufl. Heidelberg : dpunkt.verl., 2007

[Pot08] POTIX CORPORATION (Hrsg.): *ZK: The Product Overview*. http://www.zkoss.org/doc/ZK-wp-prodovw.pdf. Version: 2008, Abruf: 20.04.2008

[Ras01] RASKIN, Jef: *Das intelligente Interface: Neue Ansätze für die Entwicklung interaktiver Benutzerschnittstellen*. München : Addison–Wesley, 2001

[Rea06] REALCHAT SOFTWARE (Hrsg.): *Java vs. Flash: Which technology dominates on the client side?* http://www.realchat.com/blog/java-vs-flash/. Version: 2006, Abruf: 20.04.2008

[RH06] REUSSNER, Ralf ; HASSELBRING, Wilhelm: *Handbuch der Software–Architektur*. 1. Aufl. Heidelberg : dpunkt–Verl., 2006

[RL04] RAYMOND, Eric . ; LANDLEY, Rob W.: *The Art of Unix Usability*. http://catb.org/~esr/writings/taouu/html/. Version: 2004, Abruf: 13.04.2008

[RR07] RICHARDSON, Leonard ; RUBY, Sam: *RESTful web services*. Beijing : O'Reilly, 2007

[SG96] SHAW, Mary ; GARLAN, David: *Software architecture: Perspectives on an emerging discipline*. Upper Saddle River, NJ : Prentice Hall, 1996

[Sme06] SMEDBERG, Benjamin ; MOZILLA.ORG (Hrsg.): *Configurable*

Chrome. http://www.mozilla.org/xpfe/ConfigChromeSpec.html. Version: 2006, Abruf: 13.04.2008

[Sto08] Stoll, Corinna: *Digitale Filmarchive: Konzeption und Usability–Evaluation einer Benutzeroberfläche: Diplomarbeit*. Ilmenau, 2008

[The08] The Apache Software Foundation (Hrsg.): *The Apache Cocoon Project*. http://cocoon.apache.org/. Version: 2008, Abruf: 24.04.2008

[UB96] Unterweger, D. ; Brenner, E.: Architecture model for a user interface software tool supporting application independence. In: *Computer–Human Interaction, 1996. Proceedings., Sixth Australian Conference on* (1996), S. 205–212

[VM60] Vitruvius, Pollio ; Morgan, Morris .: *Vitruvius: The ten books on architecture*. New York : Dover Publications, 1960

[WB93] Wilhelm, Henry . ; Brower, Carol: *The permanence and care of color photographs traditional and digital color prints, color negatives, slides, and motion pictures*. 1. Aufl. Grinnell, Iowa : Preservation Publ. Co, 1993

Abkürzungsverzeichnis und Glossar

AIP	Archival Information Package, Archivpaket Für die Langzeitarchivierung erstelltes Informationspaket.
Ajax	Asynchronous JavaScript and XML Fasst verschiedene Technologien zur Erstellung dynamischer Webseiten mittels clientseitiger Verarbeitung des Präsentationsmodells und asynchroner Server-Remote-Kommunikation zusammen.
API	Application Programming Interface Programmschnittstelle eines Softwaresystems
CLI	Command Line Interface Kommandozeileninterpreter eines Systems.
CPL	Common Public License Von IBM veröffentlichte Open-Source-Lizenz für freie Software.
CSS	Cascading Style Sheets Vom W3C verwaltete Spezifikation für die Definition der Präsentationseigenschaften eines in einer Beschreibungssprache verfassten Dokumentes.
DCP	Digital Cinema Package Format für ein verschlüsseltes digitales Filmpaket zur Distribution an Kinos.

DHTML — Dynamic HTML, dynamisches HTML
Durch eine clientseitige Skriptsprache (meist JavaScript) um dynamische Funktionen erweitertes HTML.

DIP — Dissemination Information Package
Aus dem Archivpaket (AIP) für die Distribution erstelltes Informationspaket.

DRI — Digital Repository Interface
Schema für die XML-Repräsentation einer Manakin DSpace-Seite.

DTD — Document Type Definition
Sprache zur Deklaration eines XML-Schemas.

EDCine — Enhanced Digital Cinema
Projekt der Europäischen Union zur Förderung der Entwicklung von Technologien im Bereich "Digitales Kino" für das europäische Umfeld

Event Handler
Methode, die bei Auftreten eines Ereignisses (z.B. eines Benutzerereignisses) aufgerufen wird.

GPL — GNU General Public License
Von der Free Software Foundation herausgegebene Open-Source-Lizenz für freie Software.

GUI — Graphical User Interface
Grafische Benutzerschnittstelle eines Systems.

HMVC — Hierarchical Model View Controller
Objektorientiertes Architekturmuster für ein interaktives System, bei dem die MVC-Interaktionsobjekte hierarchisch angeordnet sind

HTML — HyperText Markup Language
Auf SGML basierender W3C-Standard zur Beschreibung eines Hypermedia-Dokumentes.

HTTP — Hypertext Transfer Protocol
Vom W3C verwaltete Spezifikation eines Protokolls zum Übertragen von Informationen über ein Netzwerk.

HTTP-Header
Datenblock mit Metadaten am Anfang einer über das HTTP-Protokoll übertragenen Nachricht.

Hypermediasystem
System miteinander verknüpfter Hypertext- und Multimediainhalte.

I18n — Internationalization, Mehrsprachigkeit

IAP — Intermediate Archive Package
Für die Online-Bereitstellung optimiertes Paket mit verlustbehaftet komprimiertem Filmmaterial.

ISO — International Organization for Standardization
Weltgrößter Entwickler internationaler Standards.

JPEG200 — Internationaler Standard für die Bildkomprimierung, bietet die Möglichkeit ohne Beeinträchtigung des Originals Bilder in verschiedenen Auflösungen und Darstellungsqualitäten zu extrahieren.

LZX — In OpenLaszlo verwendete, XML-basierte UI-Beschreibungssprache.

MAP — Master Archive Package
Für die Langzeitarchivierung erstelltes Archivpaket mit verlustlos komprimiertem Filmmaterial.

MBUID — Model-Based User Interface Development
Modellgetriebene Entwicklung von Benutzerschnittstellen.

MBUIDE — Model-Based User Interface Development Environments
Software für die modellgetriebene Entwicklung von Benutzerschnittstellen.

MDA	Model Driven Architecture Modellgetriebene Softwarearchitektur.
MFX	Material Exchange Format Objektbasiertes Containerformat für die gemeinsame Aufbewahrung von Audio-, Video-, und Metadaten.
MPEG4	Internationaler Standard für die Kompression von Video- und Audiodaten.
MVC	Model View Controller Objektorientiertes Architekturmuster, welches ein interaktives System in Interaktionsobjekte mit separater Betrachtung von Darstellung, Benutzerinteraktionen und Anwendungslogik unterteilt.
MXML	In Adobe Flex verwendete, XML-basierte UI-Beschreibungssprache.
PAC	Presentation Abstraction Control Objektorientiertes Architekturmuster für ein interaktives System, welches Interaktionsobjekte als PAC-Agenten in einen Anwendungs-, Steuerungs- und Präsentationsteil untergliedert.
PAC-AMODEUS	PAC - Assimilating Models of Designers, Users and Systems Eine im Rahmen des AMODEUS-Projektes entstandene Erweiterung des PAC-Musters.
Quicktime	Plattformübergreifendes Multimedia-Framework von Apple.
RealPlayer	Plattformübergreifender Media Player von RealNetworks.
REST	Representational State Transfer Architekturmuster für ein verteiltes Hypermediasystem.
RIA	Rich Internet Application Dynamische Webanwendung mit erweiterten Darstellungsmöglichkeiten und integrierter Ajax-Technologie.

RPC	Remote Procedure Call Technik zum Aufruf einer Funktion in einem entfernten Adressraum.
RUP	Rational Unified Process Iteratives Vorgehensmodell zum Softwareentwicklungsprozesses.
SGML	Standard Generalized Markup Language SGML ist der internationale Standard einer Metasprache für Beschreibungssprachen.
SIP	Submission Information Package Informationspaket zur Erstellung eines Archivpaketes (AIPs).
SOAP	Simple Object Access Protocol Netzwerkprotokoll für den Austausch XML-basierter Nachrichten.
Stakeholder	Projektbeteiligte und von der Entwicklung eines Systems betroffene Personen
SWF	Shockwave Flash Offenes Dateiformat für Rich Internet Applications. Kann Text, Video, Vektorgrafiken und Ton enthalten.
Theme	DSpace Manakin Theme Enthält eine Abfolge von XSL-Transformationen für die Umwandlung des DRI-Dokuments in das Ausgabeformat.
UI	User Interface Benutzerschnittstelle eines Systems.
UIMS	User Interface Management System System zur Trennung der Präsentations- von der Anwendungslogik.
URI	Uniform Resource Identifier Eindeutiger Bezeichner einer Ressource.

VUI — Voice User Interface
Stimmliche Benutzerschnittstelle eines Systems.

W3C — World Wide Web Consortium
Internationales Konsortium zur Entwicklung von Web Standards.

Webservice, WS
Webdienst, der den Austausch XML-basierter Nachrichten über ein Netzwerkprotokoll nutzt.

XAML — Extensible Application Markup Language
Hauptsächlich von Microsoft Windows genutzte, XML-basierte UI-Beschreibungssprache.

XForms — W3C-Standard für webbasierte Formulare.

XHTML — Extensible Hypertext Markup Language
Auf XML-Syntax und HTML-Elementen basierender W3C-Standard zur Beschreibung eines Hypermedia-Dokumentes.

XML — Extensible Markup Language
Aus SGML abgeleitete Beschreibungssprache für den Austausch und die Veröffentlichung strukturierter Daten im World Wide Web.

XPath — Sprache zur Adressierung bestimmter Bereiche eines XML-Dokuments.

XSD — XML Schema Definition
W3C-Spezifikation einer Sprache zur Deklaration eines XML-Schemas.

XSLT — XSL Transformations
XML-basierte Sprache zur Transformation von XML-Dokumenten.

XUL — XML User Interface Language
Beschreibungssprache für das User Interface von Rich Internet Applications, entstammt dem Mozilla Projekt.

Use Cases

A.1. Anwendungsszenarien

UC-2:	Ein neuer Nutzer wird angelegt
Vorbedingung:	Nutzer „Archivmitarbeiter“ oder „Anwendungsadministrator“ authentifiziert
Nachbedingung:	Neuer Nutzer wurde erstellt
Auslösendes Ereignis:	Aufruf in der Benutzerverwaltung
Paket:	System
Beschreibung:	In einer Übersicht der Benutzerverwaltung wird wird ein neuer Nutzer hinzugefügt. Die Nutzergruppe wird ausgewählt und Benutzerdaten sowie erweiterte Zugriffsdaten eingetragen
Erweiterung:	-
UC-3:	Konfiguration der UI-Module
Vorbedingung:	Nutzer „Anwendungsadministrator“ authentifiziert
Nachbedingung:	-
Auslösendes Ereignis:	Aufruf in den Anwendungseinstellungen
Paket:	System
Beschreibung:	In einer Übersicht der Modulverwaltung werden die UI-Module angezeigt. Sie können einzeln aktiviert bzw. deaktiviert werden, außerdem können modulspezifische Einstellungen vorgenommen werden.
Erweiterung:	-
UC-4:	Anzeige von DCP-Metadaten
Vorbedingung:	Nutzer „Archivmitarbeiter“ authentifiziert
Nachbedingung:	-
Auslösendes Ereignis:	Aufruf in der Item-Ansicht
Paket:	System
Beschreibung:	Eine explorative Übersicht der DCP-Inhalte wird angezeigt. Einzelne Elemente können dabei geöffnet und der XML-basierte Inhalt in einer Baumstruktur angezeigt werden. Zusätzlich stehen weitere Funktionen zum DCP zur Verfügung.

UC-5:	Item importieren (Ingest)
Vorbedingung:	Nutzer „Archivmitarbeiter“ authentifiziert
Nachbedingung:	-
Auslösendes Ereignis:	Aufruf im Menü
Paket:	Item import
Beschreibung:	Eine Übersicht der Importfunktionen wird angezeigt. Der Nutzer kann das Zielformat auswählen und dort die Encodierparameter festlegen. Er kann das zu importierende Material (Bild, Ton, Zusatzmaterial) zusammenstellen. Schließlich kann er den Import freigeben.
Erweiterung:	Es können zusätzliche Metadaten eingegeben und hinzugefügt werden.
UC-6:	Videoschnitt
Vorbedingung:	Nutzer „Archivmitarbeiter“ oder „Professioneller Nutzer“ authentifiziert
Nachbedingung:	-
Auslösendes Ereignis:	Aufruf der Videoschnittfunktion
Paket:	video editing
Beschreibung:	Eine Übersicht der ausgewählten Videos wird angezeigt. Dazu beliebig viele Videoplayer (Vorschaufenster). Für das ausgewählte Fenster wird zusätzlich eine Timeline mit der Position des Videos und weiteren Metadaten (z.B. In-, Out-Punkte, Szenen, Kapitel) angezeigt. Weiterhin ist eine Werkzeugleiste für Videoschnittfunktionen vorhanden.

Tabelle A.1.: Beispielszenario für das User Interface

A.2. Erweiterte Use-Case-Diagramme

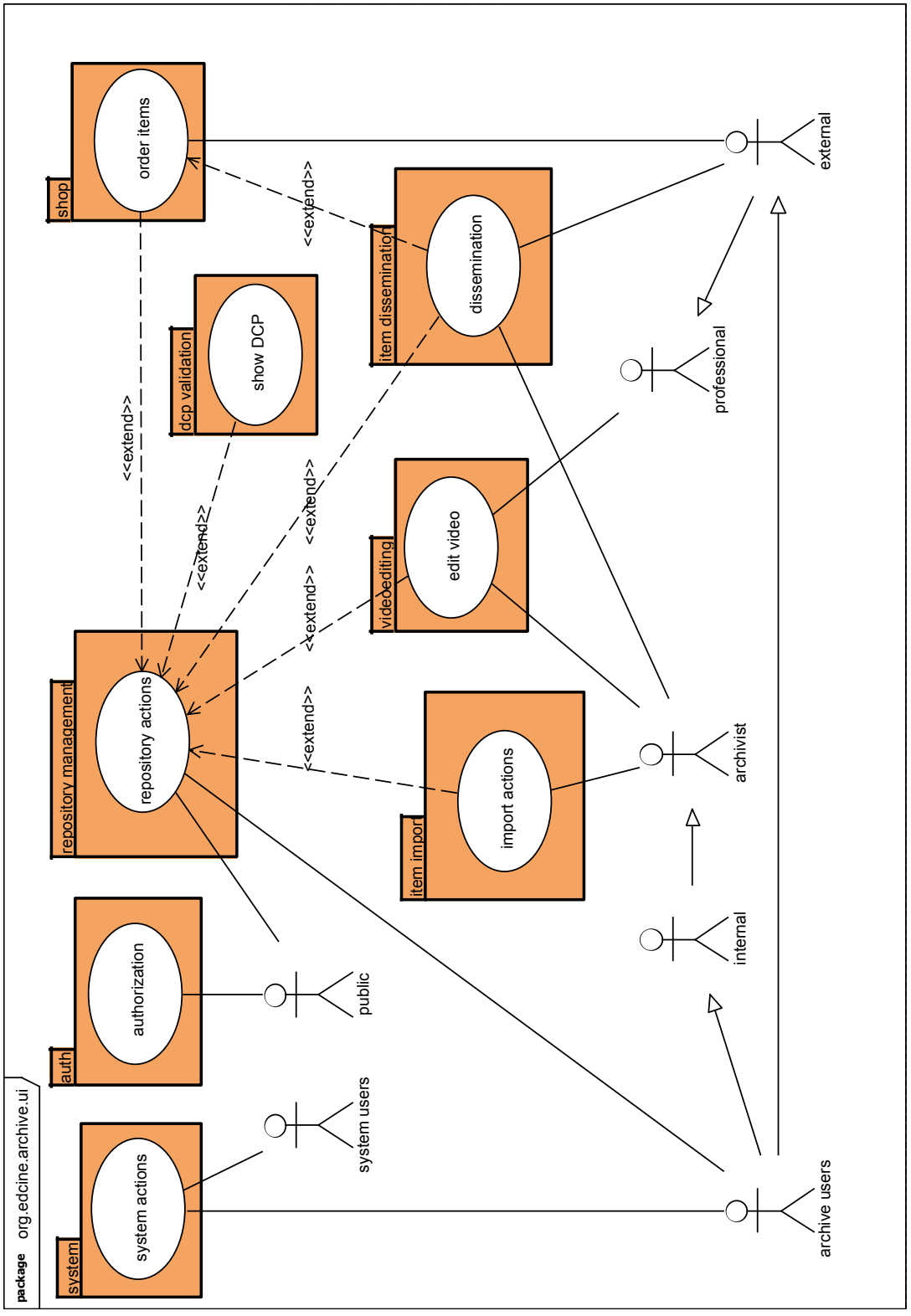

Abbildung A.1.: Use-Case-Diagramm Überblick User Interface

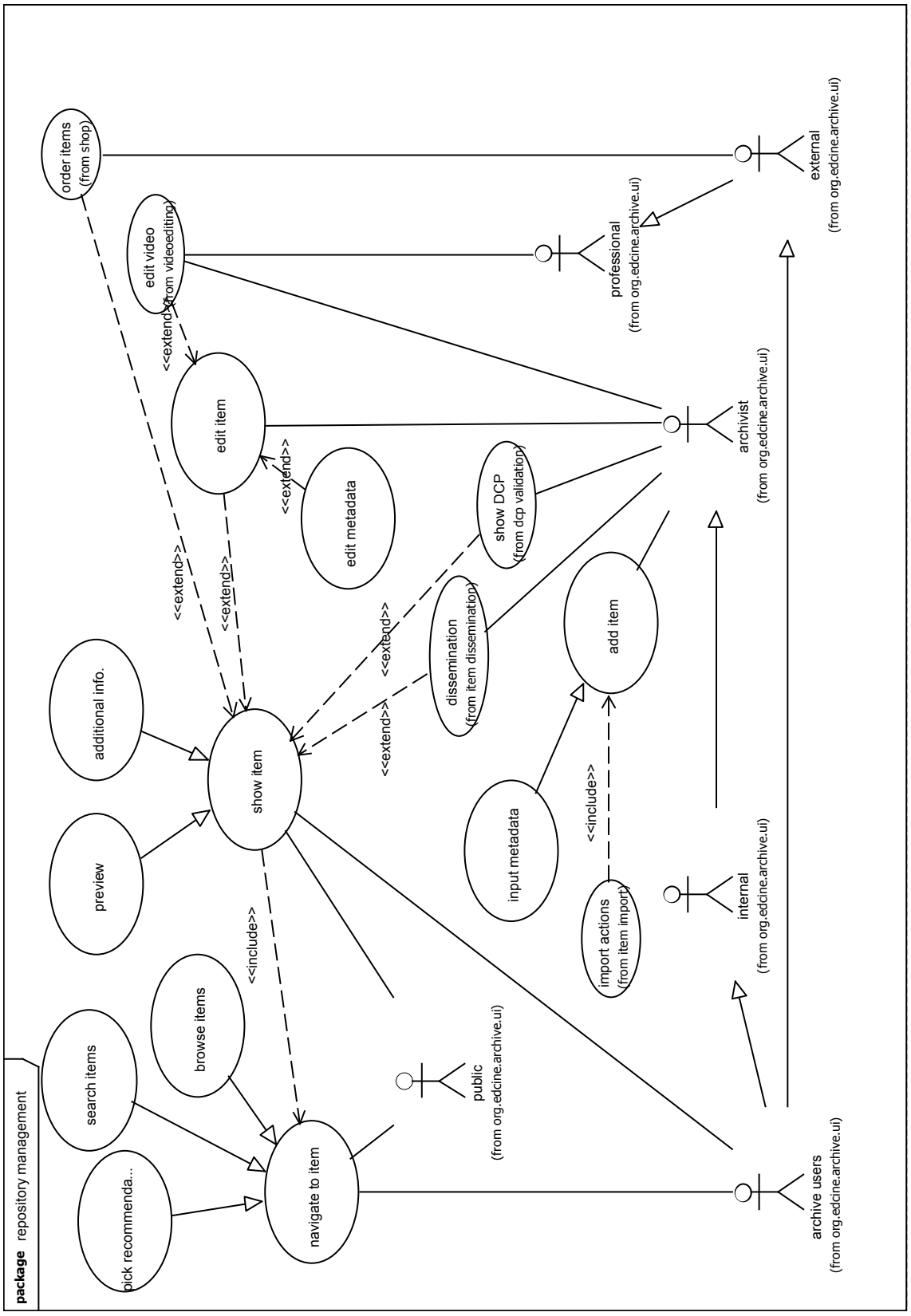

Abbildung A.2.: Use-Case-Diagramm der Repository-Komponente

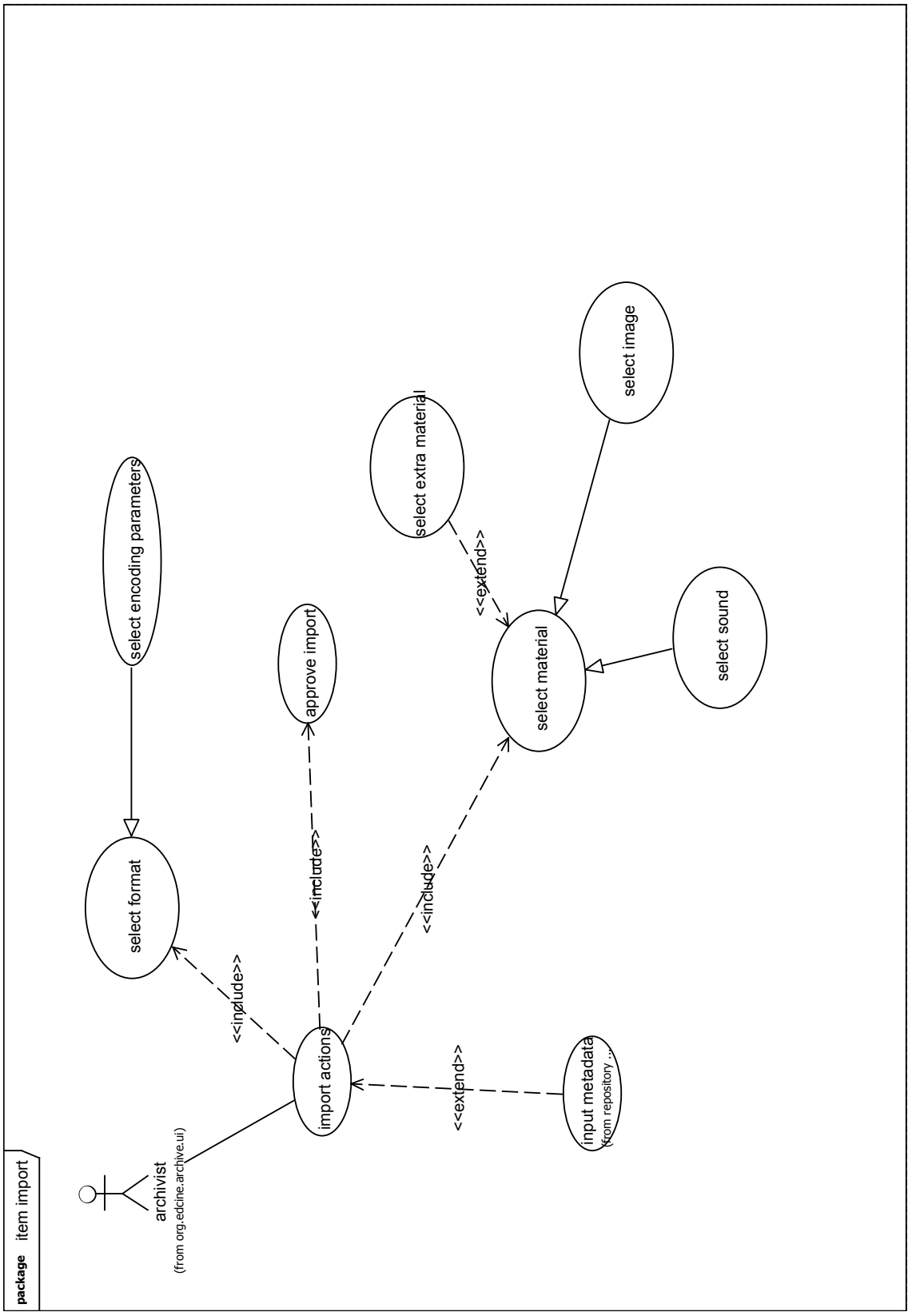

Abbildung A.3.: Use-Case-Diagramm der Ingest-Komponente

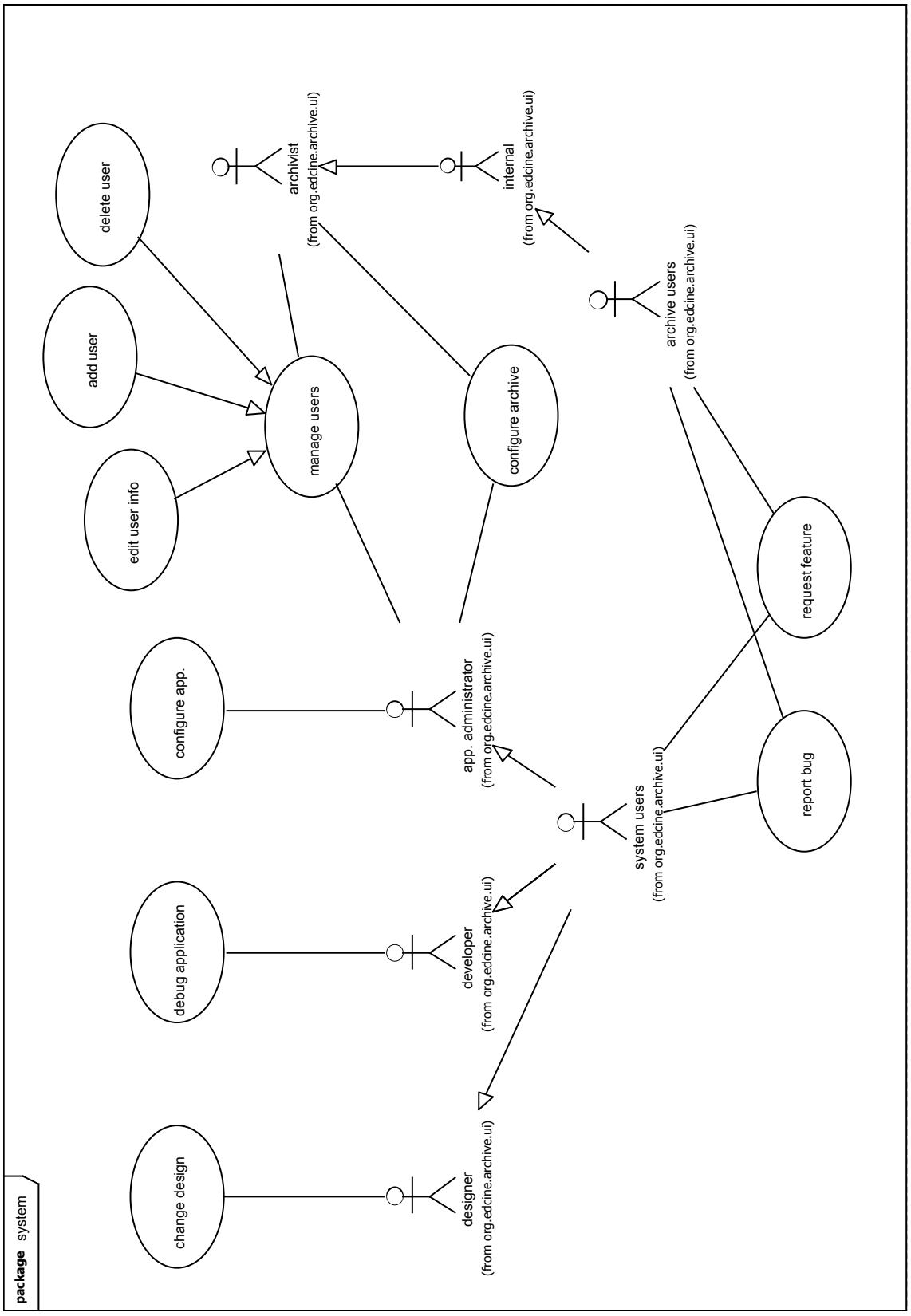

Abbildung A.4.: Use-Case-Diagramm der Systemkomponente

Anhang B.

HTTP-Headerdaten

Listing B.1: Headerdaten des ersten HTTP-GET Requests an DSpace Manakin in einer Browsersession inklusive Response-Header

```
14:59:42.435[1893ms][total 1893ms] Status: 200[OK]
GET http://edcine:8080/manakin/DRI/browse-subject?%5F%5Flzbc%5F%5F
    =1210337974664 Load Flags[LOAD_NORMAL] Content Size[8338] Mime
    Type[text/xml]
  Request Headers:
    Host[edcine:8080]
    User-Agent[Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US;
        rv:1.8.1.14) Gecko/20080404 Firefox/2.0.0.14]
    Accept[text/xml,application/xml,application/xhtml+xml,text/
        html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5]
    Accept-Language[en-us,en;q=0.5]
    Accept-Encoding[gzip,deflate]
    Accept-Charset[ISO-8859-1,utf-8;q=0.7,*;q=0.7]
    Keep-Alive[300]
    Connection[keep-alive]
  Response Headers:
    Server[Apache-Coyote/1.1]
    Set-Cookie[JSESSIONID=8289947B6A01B0A8D8AA318761ACDF30; Path
        =/manakin]
    X-Cocoon-Version[2.1.9]
    Last-Modified[Thu, 08 May 2008 19:41:10 GMT]
    Content-Type[text/xml;charset=utf-8]
    Content-Length[8338]
    Date[Fri, 09 May 2008 12:59:45 GMT]
```

Listing B.2: Headerdaten des HTTP-POST Requests an die Login-Ressource DSpace Manakins inklusive Response-Header

```
15:06:34.878[591ms][total 591ms] Status: 200[OK]
POST http://edcine:8080/manakin/DRI/login?__lzbc__=1210338390301
     Load Flags[LOAD_NORMAL] Content Size[4190] Mime Type[text/xml]
   Request Headers:
      Host[edcine:8080]
      User-Agent[Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US;
          rv:1.8.1.14) Gecko/20080404 Firefox/2.0.0.14]
      Accept[text/xml,application/xml,application/xhtml+xml,text/
          html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5]
      Accept-Language[en-us,en;q=0.5]
      Accept-Encoding[gzip,deflate]
      Accept-Charset[ISO-8859-1,utf-8;q=0.7,*;q=0.7]
      Keep-Alive[300]
      Connection[keep-alive]
      Cookie[JSESSIONID=8289947B6A01B0A8D8AA318761ACDF30]
   Post Data:
      POST_DATA[login%5Fpassword=dummypassword&login%5Femail=
          mathias%2Eweis%40stud%2Etu%2Dilmenau%2Ede]
   Response Headers:
      Server[Apache-Coyote/1.1]
      X-Cocoon-Version[2.1.9]
      Content-Type[text/xml;charset=utf-8]
      Content-Length[4190]
      Date[Fri, 09 May 2008 13:06:37 GMT]
```

Anhang C.

Quellcode

C.1. DSpace Manakin LZX-Theme

Das LZX-Theme befindet sich im elektronischen Anhang im Ordner lzxTheme. Einziger Bestandteil des Themes ist die XSL-Transformation in der Datei dri2lzx.xsl , welche aus einem DRI-Dokument ein LZX-Teilmodell erstellt.

Die mit dem LZX-Theme transformierten DRI-Dokumente sind ebenfalls im lzxTheme-Ordner hinterlegt. Das Ergebnis der Transformation ist im Ordner lzxUImodel in den entsprechenden LZX-Dateien zu finden.

Die Vorlage für das LZX-Theme ist das im DSpace Manakin als Standard-Theme verfügbare dri2xhtml-Theme im Ordner dri2xhtml mit der Datei structural.xsl.

C.2. LZX-basiertes UI-Modell und Merge-Transformation

Das LZX-basierte UI-Modell und die Merge-Transformation befinden sich im elektronischen Anhang im Ordner Quellcode unter dem Namen lzxUIModel.

Die Eingangsdaten für die Merge-Transformation sind in der Datei UIplugins.xml zu finden. Die drei zusammenzuführenden Teile des UI-Modells befinden sich in der Datei main.lzx und den beiden mit Hilfe des LZX-Themes automatisch generierten LZX-Teilmodellen in den Dateien browse-subject.lzx und browse-title.lzx.

Die Merge-Transformation in der Datei mergeUIModel.xsl vereint die drei genannten LZX-Dateien zu einem gemeinsamen UI-Model (result.lzx).

Erklärung

Hiermit erkläre ich, dass ich diese Diplomarbeit selbständig durchgeführt und abgefasst habe. Quellen, Literatur und Hilfsmittel, die von mir benutzt wurden, sind als solche gekennzeichnet.

_____

Thesen zur Diplomarbeit

1. Entwurfsentscheidungen im Architekturdesign müssen mit spezifizierten Anforderungen an das User Interface des „Digitalen Filmarchivs“ begründbar sein. Unbekannte oder variable Anforderungen erfordern eine gegenüber diesen Faktoren generische Architektur.

2. Eine optimale Architektur für das User Interface des „Digitalen Filmarchivs“ kann nur durch die Kombination der Anwendung verschiedener Architekturstile und -muster in mehreren Abstraktionsebenen erzielt werden.

3. Für die Softwarearchitektur des generischen User Interfaces für das „Digitale Filmarchiv“ empfiehlt sich auf oberster Ebene eine Organisation der Komponenten nach dem PAC-AMODEUS-Muster.

4. Vorteilhafte Muster für die Kommunikation zwischen den Komponenten des User Interfaces sind mit der REST-Architektur und der Ajax-Architektur gegeben.

5. Eine gute Erweiterbarkeit des User Interfaces lässt sich am besten über eine Plugin-Architektur und die Einbindung von User-Interface-Teilen in Plugin-Modulen erreichen.

6. XML-basierte User-Interface-Modelle können mittels einer XSL-Transformation in verschiedene Modellsprachen konvertiert werden.

7. Die Zusammenführung von Teilen eines XML-basierten User-Interface-Modells in eine geimeinsame Struktur kann mit einer XSL-Transformation vollzogen werden.

8. Für den Code des User Interfaces in der webbasierten Implementierung eignet sich das Shockwave Flash (SWF)-Format.

9. Für die Implementierung des webbasierten User Interfaces empfiehlt sich das Open Laszlo Rich Internet Application Framework als Presentation Toolkit.

10. Eine geeignete Beschreibungssprache für das User-Interface-Modell ist mit OpenLaszlo LZX gegeben.