Freislich Thomas, B.Sc.

Webportal zur Digitalisierung und Visualisierung von Werken der Landesbibliothek Steiermark

Masterarbeit

zur Erlangung des akademischen Grades
Diplomingenieur

eingereicht an der
Technischen Universität Graz

Betreuer
Univ.-Prof. Dipl-Ing. Dr.techn. Kappe Frank

Mitbetreuer
Dipl.-Ing. Leitner Helmut

Institut für Informationssysteme und Computer Medien

Graz, November 2015

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Graz, _____ _____

Date Signature

Eidesstattliche Erklärung[1]

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Graz, _____ _____

Datum Unterschrift

[1] Beschluss der Curricula-Kommission für Bachelor-, Master- und Diplomstudien vom 10.11.2008; Genehmigung des Senates am 1.12.2008

Kurzfassung

Das Anbieten von Büchern in digitaler Form ist mittlerweile zu einem großen Anliegen von vielen Bibliotheken geworden. Dies lässt sich vorwiegend durch zwei Dinge begründen. Die Bibliotheken wollen den Kunden einen einfacheren und schnelleren Zugang zu den Werken bieten. Oft scheitert die Ausleihe eines Werkes an der Bürokratie, die eine Buchausleihe mit sich bringt, oder schlicht an dem Weg von zu Hause in die Bibliothek. Da in Zeiten von Tablets der Webbrowser ein ständiger Begleiter vieler Menschen geworden ist, versuchen viele Bibliotheken und Organisationen diese Möglichkeiten zu nutzen und ihr Angebot auch digital anzubieten.

Bei manchen Werken, besonders bei sehr alten Exemplaren, lässt der materielle Zustand des Werkes eine Ausleihe und häufiges Benützen einfach nicht zu. Zu Schade wäre eine weitere Verschlechterung oder gar eine unwiederbringliche Zerstörung bzw. Verlust des Exemplars. Dies ist wohl der Zweite Grund warum viele Bibliotheken den digitalen Weg gehen wollen. Einmal digitalisiert, kann das Werk ohne jegliche Verschlechterung des Zustandes gelesen werden.

Im praktischen Teil dieser Arbeit wurde ein webbasiertes Administrationsportal zur Erfassung und Verwaltung von Werken für digitale Bibliotheken entwickelt. Gekoppelt mit einem Book-Viewer wurde ein Prototyp für die Landesbibliothek Steiermark bereitgestellt. Zum Einsatz kamen modernste Technologien der Java Enterprise Architektur. Viele Werke der Landesbibliothek Steiermark wurden bereits digitalisiert, sind aber ausschließlich

als Grafiken in einem Dateisystem abgelegt und sollen durch eine neue Webapplikation erfasst werden. Dazu ist es auch notwendig Metadaten aus einer existierenden Software im neuen webbasierten Portal zu nutzen. Im theoretischen Teil dieser Arbeit werden die aktuellen, sich am Markt befindlichen Lösungen für digitale Bibliotheken verglichen und sowohl Stärken als auch Schwächen gegenübergestellt.

Abstract

Offering literary works in digital form is a big challenge for modern libraries. This fact has two primary reasons. On the one hand libraries want an easier access to their literary works. The reason why a book loan fails is often founded by a huge bureaucracy or the distance between home and library. In times of portable computers and tablets where being connected to the internet is omnipresent a lot of libraries are trying to use this opportunity for presentig their literary works in the internet.

Especially very old works are in a bad condition. Loaning these works leads to a continuing deterioration. To avoid this deterioration it seems obvious to digitize affected literary works. This is the second main reason for the growth of digital libraries.

Basically this thesis consists of two major parts. The first part is the development of a state of the art webapplication to upload and maintain digitized works. This administration portal is then linked with a book viewer to display the literary works in the webbrowser. Coupled with metadata from an already existing datapool the webapplication should be act as a new digtal library platform for the styrian regional library. The second part is a comparison of existing solutions for digital libraries on the market.

Inhaltsverzeichnis

Kurzfassung **3**

Abstract **5**

1 Anforderungsprofil der Landesbibliothek Steiermark **15**

1.1 Funktionale Anforderungen 15

1.1.1 Administrationsportal 17

1.1.2 Book-Viewer 18

1.2 Technische Anforderungen 19

1.3 Eigenentwicklung vs. Drittsoftware 20

1.3.1 Vorteile einer Eigenentwicklung 21

1.3.2 Nachteile einer Eigenentwicklung 23

2 Vergleich von Lösungen am Markt **25**

2.1 Evaluierung für das Administrationsportal 25

2.1.1 GOOBI Production 26

2.1.2 Visual Library Manager (VLM) 28

2.2 Evaluierung für den Book-Viewer 30

2.2.1 Visual Library Server (VLS) 31

2.2.2 GOOBI Presentation 33

2.2.3 Intranda Viewer 35

2.2.4 DFG-Viewer 37

3 Beschreibung der Eigenentwicklung **41**
3.1 Administrationsportal 41
3.2 Book-Viewer 43
3.2.1 Vorteile dieses Book-Viewers 44
3.2.2 Nachteile dieses Book-Viewers 45

4 Eingesetzte Technologien **47**
4.1 Java Enterprise Edition (Java EE) 47
4.2 Hibernate 49
4.3 JavaServer Faces (JSF) 51
4.3.1 JSF Lifecycle 53
4.3.2 Managed Beans 56
4.3.3 Expression Language 58
4.3.4 Konfiguration von JSF Implementierungen 60
4.3.5 AJAX and JSF 62
4.3.6 Konvertierer 63
4.3.7 Validierung 66
4.3.8 UI Komponenten 67
4.3.9 Navigation 68

5 Implementationsdetails **75**
5.1 Architektur 75
5.2 Datenbankschema 76
5.2.1 WB_BOOK 77
5.2.2 WB_USER_BOOK_OPTIONS 77
5.2.3 WB_SOURCE 78
5.2.4 WB_SCOPE 78
5.2.5 WB_METADATA 79
5.2.6 WB_AUTHOR 80
5.2.7 WB_PUBLISHER 80
5.2.8 WB_METADATA_TO_AUTHOR 81

5.2.9 WB_CATEGORY 81
5.2.10 WB_CATEGORY_MEMBER 82
5.2.11 WB_CATEGORY_CHILDREN 82
5.2.12 WB_PAGE 83
5.2.13 WB_PAGE_META 84
5.2.14 WB_LAYER 84
5.2.15 WB_FORMAT 85
5.2.16 WB_LAYER_TO_FORMAT 86
5.2.17 WB_PROGRESS 86

6 Ausblick **89**

7 Resümee **91**

Literatur **95**

Abbildungsverzeichnis

1.1 Aktueller Zustand und Zielsystem 16
1.2 Schichtenmodell und Technologieübersicht 19

2.1 GOOBI Production 26
2.2 Visual Library Server 31
2.3 GOOBI Presentation 33
2.4 Intranda Viewer 35
2.5 DFG-Viewer 37

3.1 Bearbeitung von Metadaten im Admintool 42
3.2 Bearbeitung von Seiten im Admintool 43
3.3 Anzeige von Seiten im Book-Viewer 44
3.4 Übersicht aller Werke im Book-Viewer 45

4.1 Java EE Container 48
4.2 Hibernate 49
4.3 MVC 52
4.4 JSF Lifecycle 54
4.5 AJAX Request 63
4.6 AJAX Execute 64
4.7 AJAX Render 64
4.8 JSF Komponentenhierarchie 69

5.1 Architektur 76

5.2 Tabelle WB_BOOK 77
5.3 Tabelle WB_USER_BOOK_OPTIONS 78
5.4 Tabelle WB_SOURCE 78
5.5 Tabelle WB_SCOPE 79
5.6 Tabelle WB_METADATA 80
5.7 Tabelle WB_AUTHOR 81
5.8 Tabelle WB_PUBLISHER 81
5.9 Tabelle WB_METADATA_TO_AUTHOR 81
5.10 Tabelle WB_CATEGORY 82
5.11 Tabelle WB_CATEGORY_MEMBER 82
5.12 Tabelle WB_CHILDREN_MEMBER 83
5.13 Tabelle WB_PAGE 83
5.14 Tabelle WB_PAGE_META 84
5.15 Tabelle WB_LAYER 85
5.16 Tabelle WB_FORMAT 85
5.17 Tabelle WB_LAYER_TO_FORMAT 86
5.18 Tabelle WB_PROGRESS 86

Abkürzungsverzeichnis

AJAX Asynchronous JavaScript and XML 62
ALTO Analyzed Layout and Text Object 38
AS Application Server 47
DC Dublin Core 17
DFG Deutsche Forschungsgemeinschaft 37
EL Expression Language 58
GNU GPL GNU General Public License 26
MARCXML Machine-Readable Cataloging XML 17
METS Metadata Encoding & Transmission Standard 17
MODS Metadata Object Description Schema 26
MVC Model View Controller 51
OAI-PMH Open Archives Initiative - Protocol for Metadata Harvesting 34
OCR Optical Character Recognition 29
ORM Object-Relational Mapping 49
POJO Plain Old Java Object 56
TEI Text Encoding Initiative 33
URN Uniform Resource Name 17
VLS Visual Library Server 29
VSM Visual Library Manager 29

1 Anforderungsprofil der Landesbibliothek Steiermark

Die steirische Landesbibliothek sucht nach einer Webapplikation um ihre digitalisierten Werke der breiten Öffentlichkeit zu zeigen. Sowohl funktional wie auch technisch müssen die Anforderungen der Landesbibliothek erfüllt werden.

1.1 Funktionale Anforderungen

Basis aller Überlegungen der Landesbibliothek ist in Abbildung 1.1 dargestellt. Sie bildet den aktuellen Stand (grün) und das gewünschte System (blau) ab. Ziel der Landesbibliothek Steiermark ist es, eine eigene Lösung zu entwickeln, um eine Alternative zu der sich am Markt befindlichen Möglichkeiten zu erhalten.

Derzeit werden viele Werke digitalisiert und in einem Filesystem abgelegt. Die Scans der Werke liegen in den Formaten *jpeg* oder *tiff* vor. Zusätzlich wird ein System namens *DABIS* gepflegt, in welchem Werke mit Metadaten versehen und zentral in einer Datenbank abgelegt werden können. Es ist aber oft nicht genau klar, welche der digitalisierten Werke im Filesystem bereits in DABIS erfasst wurden und welche nicht. Weiters wurden Zettelkataloge in der Landesbibliothek Steiermark digitalisiert und in das System DABIS

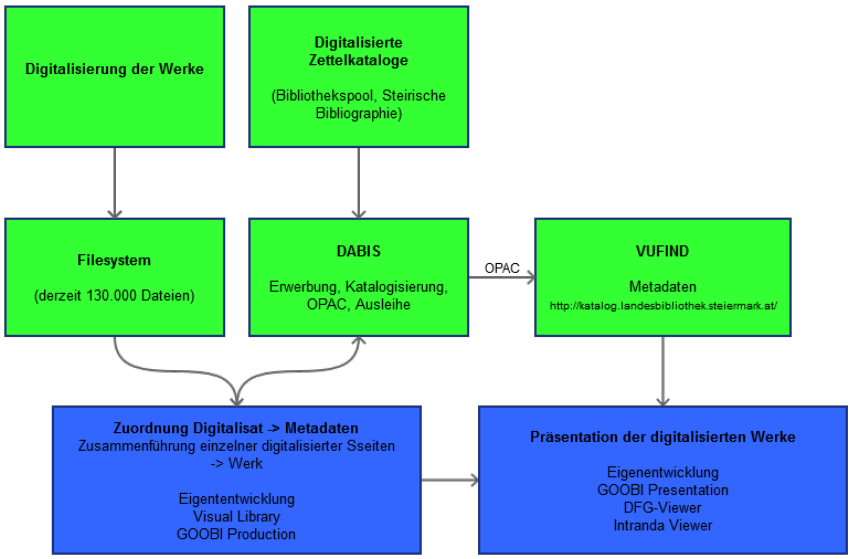

Abbildung 1.1: Aktueller Zustand und Zielsystem

überführt. Ist ein Werk in DABIS erfasst heißt dies nicht zwangsläufig, dass es auch eine Digitalisat davon gibt. DABIS ist eine Anwendung eines Drittherstellers und wurde von der Landesbibliothek zugekauft. Aus DABIS werden laufend die Metadaten der vorhandenen Werke exportiert und in einer Instanz der Open-Source-Software *VuFind* [37] wieder importiert. Diese Instanz ist über das Portal der Landesbibliothek [23] aufrufbar. Hier können Informationen, die aus DABIS exportiert und in VuFind wieder importiert wurden, gesucht und angezeigt werden. VuFind bietet jedoch keinen Book-Viewer im eigentlich Sinne an, sondern nur einen Download von Files welche dann vom Benutzer lokal am Rechner und mit einem passenden Viewer betrachten werden können. Den Benutzern soll jedoch ein interaktiver webbasierter Viewer geboten werden, in dem man, ähnlich wie in einem Buch, navigieren kann.

Es sollen im Prinzip zwei Systeme entstehen. Zum einen ein Administrationsportal, in welchem die Scans vom Filesystem importiert und mit den Export-Daten aus DABIS in Verbindung gebracht werden und zum anderen ein Book-Viewer, mit dem ein Werk im Browser betrachtet werden kann.

1.1.1 Administrationsportal

Folgende Anforderungen seitens der Landesbibliothek wurden an des Administrationsportal gestellt:

- Das Administrationsportal soll durch eine multifunktionale Onlineversion realisiert werden. Die zugrunde liegenden Technologien haben den Vorgaben der IT-Abteilung der Landesregierung Steiermark zu genügen. Das Produktionssystem muss auf jeden Fall webbasiert sein damit keine Client-Software auf den Benutzer-PCs installiert werden muss.
- Es muss eine vielseitige Verwendungsmöglichkeit gegeben sein. Zeitungen, Handschriften, Bandserien, Karten, Monographien usw. sollen unterstützt werden.
- Alle Datenfelder müssen mit den internationalen Standards der *Library of Congress (LOC)* [24] übereinstimmen, um an andere Portale (Uniform Resource Name (URN) Resolver, Europeana [10], etc.) andocken zu können.
- Über eine OAI-PMH-Schnittstelle [29] sollen die vorhandenen Werke in folgenden Formaten zur Verfügung gestellt werden können:
  - Dublin Core (DC) [8]
  - Metadata Encoding & Transmission Standard (METS) [26]
  - Machine-Readable Cataloging XML (MARCXML) [5]
- Aus DABIS sollen generierte Metadaten-Files in das Administrationsportal importiert werden können.

- Persistente Identifier (URN) müssen vorhanden sein und automatisch generiert werden.
- Ein Langzeitarchivierungskonzept muss Datenverlust vorbeugen und soll mit entsprechenden Skripten realisiert werden.
- Beliebig viele Benutzer sollen das System benutzen können.
- Das System soll individuell anpassbar sein.
- Eine User- und Rechteverwaltung soll den Zugriff auf des Administrationsportal verwalten.
- Jeder Seite eines Werkes soll eine Zusatzinformation hinzugefügt werden können. Die Anzeige dieser Zusatzinformation soll im Book-Viewer möglich sein.

1.1.2 Book-Viewer

Folgende Anforderungen seitens der Landesbibliothek wurden an den Book-Viewer gestellt:

- Ein multifunktionaler Viewer soll für die Anzeige von digitalisierten Werken zur Verfügung stehen. Es soll sich um ebenfalls um eine webbasierte Lösung handeln.
- Mithilfe einer Volltextsuche soll in Werken gesucht werden können. Voraussetzung dafür ist das Vorhandsein von OCR-Daten zusätzlich zu den Scans.
- Im Viewer soll, ähnlich wie in einem realen Buch, geblättert werden können. Einzelne Buchseiten sollen nebeneinander wie auch einzeln darstellbar sein. Hinein- und hinauszoomen soll ebenfalls möglich sein.
- Zusatzinformation die im Administrationsportal einer Seite zugeordnet wurde soll im Book-Viewer optional angezeigt werden können.

Im Austria-Forum [39] kommt bereits ein Book-Viewer zum Einsatz der am Institut für Informationssysteme und Computermedien (IICM) der TU Graz entwickelt wurde. Dieser Book-Viewer soll, bis auf geringe Anpassungen, auch für die Landesbibliothek Steiermark eingesetzt werden.

1.2 Technische Anforderungen

Die spezielle Infrastruktur der Landesbibliothek für JSF Anwendungen basiert auf Java Enterprise Edition 6 [7]. Als Applikationsserver wird *JBoss Enterprise Application Platform (EAP)* [20] in der Version 7 verwendet.

Abbildung 1.2 zeigt dabei die Auflistung der einzelnen Schichten und die zugehörigen Technologien.

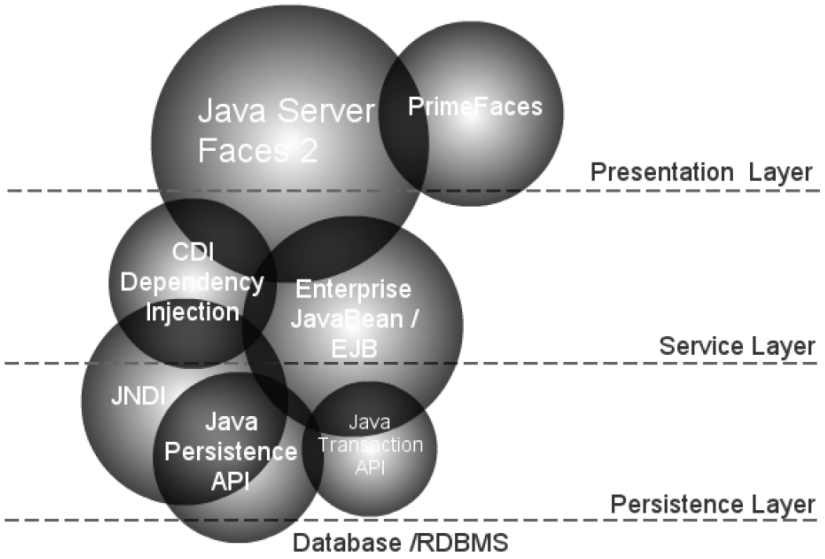

Abbildung 1.2: Schichtenmodell und Technologieübersicht.

Als zentrales Element in der Präsentationsschicht kommt *JSF 2.1 (Mojarra)* [22] zum Einsatz. Als Erweiterung der JSF-Implementierung wird das Komponentenframework *Primefaces* [32] verwendet. Im Service Layer werden die komfortabel zu benutzenden *Enterprise Java Beans (EJB)* [9] verwendet. Der Datenbankzugriff in der Persistenzschicht erfolgt über die *Java Persistance API* [18]. Konkret wird *Hibernate* [14] als *Object-Relational Mapping (ORM)* Framework eingesetzt. Als Datenbankmanagementsystem fungiert außerdem ein Oracle [30] Instanz.

Context and Dependency Injection (CDI) bzw. die Erweiterung *Apache MyFaces Extension CDI (CODI)* [2] werden für den "glue-code" zwischen den einzelnen Schichten verwendet. Als Validierung wird schichtübergreifend *Bean Validation (JSR 303)* mit der Erweiterung *Apache MyFaces Extensions Validator* [3] eingesetzt.

Für die Datenbankanbindung werden die entsprechenden *Java Transaction API (JTA)* [19] behafteten Data-Sources vom Applikationsserver über das *Java Naming and Directory Interface (JNDI)* [17] bereitgestellt.

1.3 Eigenentwicklung vs. Drittsoftware

Nachdem eine Idee für eine Applikation geboren ist, stellt sich meist die Frage "Selber entwickeln oder zukaufen?". Da diese Entscheidung von vielen Faktoren abhängt, gibt es dafür keine pauschal gültige Antwort. Folgende Fragen sollte man sich zumindest stellen:

- Welche Funktionalität soll die Lösung bereitstellen?
- Welche technischen Einschränkungen gibt es? (Infrastruktur, Programmiersprache, ...)
- Wie wird sich das Projekt in naher und mittlerer Zukunft entwickeln (viel/wenig Weiterentwicklung etc.)?

- Wie viel Geld möchte man ausgeben in Hinblick auf Anschaffung und Wartung?

Bereits im Vorfeld des Entwicklungsstarts wurde seitens der IT-Abteilung der Landesregierung Steiermark die Entscheidung getroffen einen eigenen Prototypen zu entwickeln, um einen besseren Vergleich zu Drittsoftware zu haben. In den nachfolgenden Kapitel werden die Beweggründe für diese Entscheidung und die Unterschiede zwischen Eigenentwicklung und Drittsoftware näher erläutert.

1.3.1 Vorteile einer Eigenentwicklung

Die Vorteile der Eigenentwicklung können in den angeführten Punkten nachvollzogen werden:

- **Vollständige Integration in die IT-Infrastruktur**
  Die Eigenentwicklung kann zu 100% an die Anforderungen der internen IT-Infrastruktur angepasst werden. Eine Drittsoftware hätte den Nachteil, dass man eventuell ein eigenes Server- und Softwaresetup benötigt oder das Hosting der Software dem Anbieter der Drittsoftware übergeben müsste.

- **Keine Abhängigkeit**
  Weiters bietet eine Eigenentwicklung auch den Vorteil nicht von einem Dritthersteller abhängig zu sein. Es besteht immer die Gefahr einer Insolvenz und der dadurch herbeigeführten Probleme. Ist man jedoch im Besitz des Quellcodes so kann man selbst über die Weiterentwicklung und Wartung des Projekts entscheiden.

- **Geringe laufende Kosten**
  Erweiterungen und Anpassungen am System können im Vergleich zu einer Drittsoftware schneller und kostengünstiger vorgenommen werden. Besonders diese zwei Faktoren können bei einem ständig wachsenden Projekt ausschlaggebende Faktoren sein.

- **Anforderungsnahe Lösung und bessere Wartbarkeit**
  Die Eigenentwicklung kann sowohl auf die technischen als auch auf die funktionalen Anforderungen maßgeschneidert werden. Überflüssige Funktionalitäten die man nicht benötigt, müssen auch nicht entwickelt werden und verhindern das Aufblähen der Software. Dritthersteller versuchen möglichst viele Kunden mit ihrem Produkt anzusprechen und implementieren viel Funktionalität. Dies verursacht häufig überdimensionierte und fehleranfällige Software.

- **Weniger Aufwand und Kosten für die Mitarbeiterschulung**
  Bei der Eigenentwicklung würde man der Applikation dasselbe User-Interface und "Look & Feel" geben, welches bereits bei vielen anderen Applikation der Landesregierung Steiermark im Einsatz ist. Die Sachbearbeiter sind diese Darstellungsformen gewöhnt und müssen sich nicht erst in die neue Software einarbeiten.

- **Integration in das Authentifizierungssystem**
  Die Applikationen im Pool der steirischen Landesregierung werden über das eigene User- und Rechtesystem STERZ [34] verwaltet. Da in der gewünschten Applikation ein User- und Rechtesystem vorgesehen ist, kann das bereits vorhandene System einfach über die Applikation "gestülpt" werden. Es müssen nur noch die Zugriffsrechte für die Sachbearbeiter vergeben werden.

1.3.2 Nachteile einer Eigenentwicklung

Eine vollständige Eigenentwicklung bringt natürlich nicht ausschließlich Vorteile gegenüber einer zugekauften Lösung. Nachfolgend werden diese Aspekte angeführt:

- **Höhere Kosten während der Entwicklung**
  Natürlich ist für das Unternehmen während der Entwicklung, besonders während der intensiven Entwicklungsphase zu Beginn, mit erhöhten Kosten zu rechnen. Mitarbeiter müssen für das bestimmte Projekt abgestellt oder sogar neue Mitarbeiter eingestellt werden. Dies hängt natürlich auch von Projektgröße und Komplexität ab. Eine Eigenentwicklung muss aber auch über den gesamten Produktlebenszyklus hinweg nicht zwangsläufig billiger sein als eine kommerzielle Software.

- **Abhängigkeit von Mitarbeitern**
  Der Vorteil der Eigenentwicklung, dass man nicht von einem anderen Unternehmen abhängig ist, ist zugleich auch ein Nachteil der Eigenentwicklung. Oft hängen Projekt nur an einigen wenigen Mitarbeitern die viel Know-How besitzen. Ein Verlust solcher Schlüssel-Mitarbeiter kann erhebliche Probleme verursachen und den Projekterfolg gefährden.

- **Zeitbedarf bis zum produktiven Einsatz**
  Abhängig von der Projektgröße muss ein erheblicher Zeitbedarf, bis die Lösung produktiv geht, eingeplant werden. Die Phasen Implementierung und Test im Softwareentwicklungsprozess sollten im Hinblick auf gute Qualität zeitlich großzügig bemessen werden. Ein zu schneller Start der produktiven Phase kann den Erfolg maßgeblich beeinflussen.

- **Möglicher Qualitätsvorsprung der Drittsoftware**
  Software von Drittunternehmen werden oft schon jahrelang erfolgreich eingesetzt und viele Bugs wurden bereits gelöst. Möglicherweise ist also die Qualität der Software besonders zu Beginn der produktiven Phase beim Drittprodukt besser.

2 Vergleich von Lösungen am Markt

Es gibt bereits sich am Markt befindliche Lösungen die teilweise den gewünschten Funktionsumfang der Landesbibliothek abdecken. Sowohl Open-Source Lösungen wie auch kommerzielle Software müssen evaluiert werden. Als Administrationsportal im Hintergrund wünscht man sich einen Vergleich der Produkte *Visual Library Manager* [36] und *GOOBI Production* [13]. Als Book-Viewer sieht die Landesbibliothek *GOOBI Presentation* [12], *DFG-Viewer* [6], *Visual Library Server* [36] und den *Intranda-Viewer* [16] als mögliche Kandidaten.

2.1 Evaluierung für das Administrationsportal

Nachfolgend werden die Alternativen zu einer Eigenentwicklung des Administrationsportal bezogen auf die Produkte *Visual Library* und *GOOBI Production* näher beleuchtet.

2.1.1 GOOBI Production

GOOBI ist ein Open-Source Softwarepaket für Digitalisierungsprojekte. "Goobi. Digitalisieren im Verein e. V." ist ein Verein der 2012 gegründet wurde und das Projekts in vielen Belangen vertritt (Releasemanagement, Öffentlichkeitsarbeit, etc.). GOOBI wird in vielen Bibliotheken weltweit eingesetzt. Es besteht im wesentlichen aus zwei Bestandteilen: *GOOBI Production* und *GOOBI Presentation*. Wobei, wie es der Name schon sagt, GOOBI Production das Workflowmanagement (Abbildung 2.1) für den Digitalisierungsprozess darstellt und GOOBI Presentation zur Darstellung der Digitalisate dient [13].

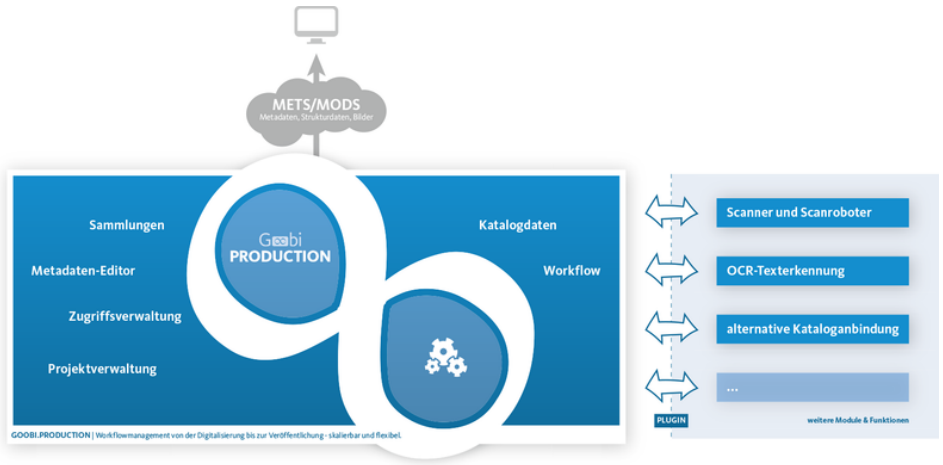

Abbildung 2.1: GOOBI Production [13]

Das gesamte GOOBI Softwarepaket steht kostenlos und lizenzkostenfrei zur Verfügung und ist unter GNU General Public License (GNU GPL) lizenziert. GOOBI Production ist eine Webapplikation die in Java entwickelt wurde und bietet offene Schnittstellen (METS/Metadata Object Description Schema (MODS)). GOOBI Production und GOOBI Presentation sind völlig unabhängig voneinander und können daher individuell kombiniert werden. Es wären zum Beispiel folgende Kombination vorstellbar [13]:

- GOOBI Production mit Intranda Viewer
- GOOBI Production mit Presentation
- GOOBI Production mit DFG-Viewer
- GOOBI Production mit Eigentwicklung

Weiters ist der GOOBI Workflow bereits auf viele verschiedene Medientypen ausgelegt:

- Bücher
- Briefe
- Zeichnungen
- Zeitungen
- Zeitschriften
- Handschriften
- Nachlässe
- usw.

Einige Unternehmen, unter anderem *Zeutschel GmbH* [40] oder *Intranda GmbH* [16], benützen GOOBI als Basis um ihre eigenen Produkte daraus weiterzuentwickeln bzw. um individuelle Anpassungen nach Kundenwunsch zu fertigen.

Vorteile von GOOBI Production

- Source-Code frei zugänglich (Open-Source)
- Keine Lizenzkosten (GNU GPL)
- Kann mit anderen Book-Viewern kombiniert werden (GOOBI Presentation muss nicht zwingend verwendet werden)
- Wird bereits in vielen großen Bibliotheken erfolgreich eingesetzt
- Vollständig an die eigenen funktionalen Bedürfnisse anpassbar
- Laufende Weiterentwicklung des Softwarepakets durch die GOOBI Community

- Ein stabiles Softwarepaket, welches man an die Anforderungen der Landesbibliothek anpassen kann, ist bereits vorhanden
- Offene Implementierung für externe Module
- Offene Schnittstellen (METS/MODS)

Nachteile von GOOBI Production

- Lange Einarbeitungszeit in den bestehenden Source-Code
- Evaluierung der sicherheitsrelevanten Faktoren bei Eingliederung in das IT-System der Landesregierung
- Entscheidet man sich das System nicht selbst zu hosten, könnten kritische Daten den Verantwortungsbereich der Landesbibliothek verlassen
- Bei vollständiger Eingliederung in das Framework für JSF Anwendungen der Landesregierung ist ein erheblicher Aufwand notwendig, welcher bereits eine Eigenentwicklung rechtfertigen könnte
- Ein User- und Rechteverwaltungssystem müsste erst eingebaut werden
- Lagert man die Entwicklung eines an die eigenen Anforderungen angepassten GOOBI an ein externes Unternehmen aus, ist man wieder mit Entwicklungs- und Lizenzkosten konfrontiert

2.1.2 Visual Library Manager (VLM)

Das Softwarepaket *Visual Library* ist aus einem Konsortium der Firmen *semantics Kommunikationsmanagement GmbH* [16] und *Walter Nagel GmbH & Co KG* [38] entstanden. Es ist ein proprietäres Softwareprodukt, somit steht kein Source-Code zur Verfügung. Jegliche Erweiterungen, Änderungen, Bugfixes etc. müssen also direkt vom Softwarehersteller durchgeführt werden. Visual Library besteht, wie auch GOOBI, aus mehreren Bestandteilen:

- Visual Library Manager

- Visual Library Server
- Visual Library Scout

Der **Visual Library Manager (VSM)** ist dabei das zentrale Element welcher die digitalisierten Werke verwaltet und ist für die Zuordnung von Metadaten etc. verwendet wird (Administrationsportal). Es wird als Multi-User System realisiert und es werden zusätzlich zu den Standardfunktionalitäten viele weitere Komponenten angeboten. Folgenden Funktionalitäten werden von den verschiedensten Komponenten unterstützt [36]:

- METS Schnittstelle
- Tools zur Bildbearbeitung
- Automatische Optical Character Recognition (OCR) Generierung über eine ABBY Fine-Reader Engine
- Erstellung von bestimmten Dateiformaten zum Download (PDF, TIFF, PNG, usw.)
- Schnittstellen zu diversen Geräten welche zum digitalisieren verwendet werden können (Scanner)

Der VSM bietet auch eine Schnittstelle zu *Visual Library .NET* [33], welche Zugang zu vielen Bibliothekskatalogen, Buchhandlungen, Fachdatenbanken etc. hat. Dies ermöglicht einen automatischen Import von Metadaten und verbessert die Effizienz der Digitalisierungsprozesse. Über den VSM können Werke freigegeben und über den **Visual Library Server (VLS)** publiziert werden. Der VLS wird im Kapitel 2.2.1 näher betrachtet.

VSM ist keine Web-Anwendung sondern eine Standalone-Applikation. Als mögliche Datenbanken stehen MS-SQLServer, MS-Access, MySQL und Firebird zur Verfügung.

Der dritte Teil des Softwarepakets Visual Library ist das Tool **Visual Library Scout**, welches die Möglichkeiten gibt Inhalte sowohl offline als auch online zu betrachten. Benutzer können sowohl Daten recherchieren als auch Daten

bearbeiten oder anlegen. Visual Library Scout ist dazu gedacht den eigenen Kunden eine Plattform zu bieten und für die aktuellen Anforderungen der Landesbibliothek nicht erforderlich.

Vorteile von Visual Library Manager

- Weite Verbreitung der Software
- Großer Funktionsumfang der viele der Anforderungen bereits abdeckt
- Erweiterungen können je nach Bedarf zugekauft werden
- Schneller produktiver Start möglich
- Viele Schnittstellen
- Theoretisch auch andere Book-Viewer mit geeigneter Schnittstelle verwendbar

Nachteile von Visual Library Manager

- Das Administrationstool (Virtual Library Manager) ist keine Webanwendung (benötigt lokale Installation)
- Proprietäre Software
- Starke Abhängigkeit von den Drittfirmen
- Hohe Anschaffungskosten und laufende Lizenzkosten
- Kann nicht direkt in die Infrastruktur der Landesregierung Steiermark eingegliedert werden
- Erhöhter Einschulungsaufwand für die Sachbearbeiter

2.2 Evaluierung für den Book-Viewer

Nachdem die Möglichkeiten für das Administrationstool bereits beleuchtet wurden, stellt sich nun die Frage nach einem geeigneten Book-Viewer.

GOOBI Presentation, *DFG-Viewer*, *Intranda-Viewer* und der *Visual Library Server* sind potentielle Alternativen zu einer Eigenentwicklung.

2.2.1 Visual Library Server (VLS)

Der VLS (Beispiel siehe Abbildung 2.2) repräsentiert den Book-Viewer im Softwarepaket von Visual Library. Die Software ist als Webanwendung realisiert, wobei Design und Layout vollständig an die eigenen Anforderungen angepasst werden können.

Abbildung 2.2: Visual Library Server (`http://sammlungen.ub.uni-frankfurt.de/cm/periodical/pageview/4884556`)

Der VLS läuft auf einem Apache-Webserver. Wie auch schon der VSM ist der VLS in einer Basisversion als auch in einer mit verschiedensten Komponenten erweiterten Version verfügbar. Die Basisfunktion bietet unter anderem [36]:

- Such- und Sortierfunktionen
- Anzeige von Metadaten

- Klassischen Navigationsfunktionen im Werk (Zoomen, Blättern, etc.)
- Exportfunktionalitäten

Optional stehen viele weitere kostenpflichtig Pakete, die dem Benutzer viele neue Möglichkeiten geben, zur Verfügung [36]:

- OCR-Volltextanzeige inkl. Highlighting von Ergebnissen
- Verschiedenste Schnittstellen für den Export
- Schnellere Anzeige von Grafiken durch Einsatz von *Zoomify* [41]
- Integration eines Online-Shops, um auch kostenpflichtige Werke anbieten zu können
- Ansicht von Thumbnails
- Statistiken über Webseitenaufrufe
- etc.

Vorteile von Visual Library Server

- Weite Verbreitung des Book-Viewers
- Großer Funktionsumfang, der viele der Anforderungen bereits abdeckt
- Erweiterungen können je nach Bedarf zugekauft werden
- Verwendet aktuelle Technologien (HTML5, AJAX)

Nachteile von Visual Library Server

- Proprietäre Software
- Starke Abhängigkeit von den Drittfirmen
- Hohe Anschaffungskosten und laufende Lizenzkosten
- Der Book-Viewer ist hinsichtlich Bedienbarkeit wenig anwenderfreundlich

2.2.2 GOOBI Presentation

GOOBI Presentation (Beispiel siehe Abbildung 2.3) ist keine komplett eigenständige Webapplikation, sondern eine Erweiterung des Content-Managment-System TYPO3 [35]. TYPO3 basiert auf PHP und kann auf unterschiedlichste Datenbanken zugreifen (u.a. MySQL oder Oracle) [12].

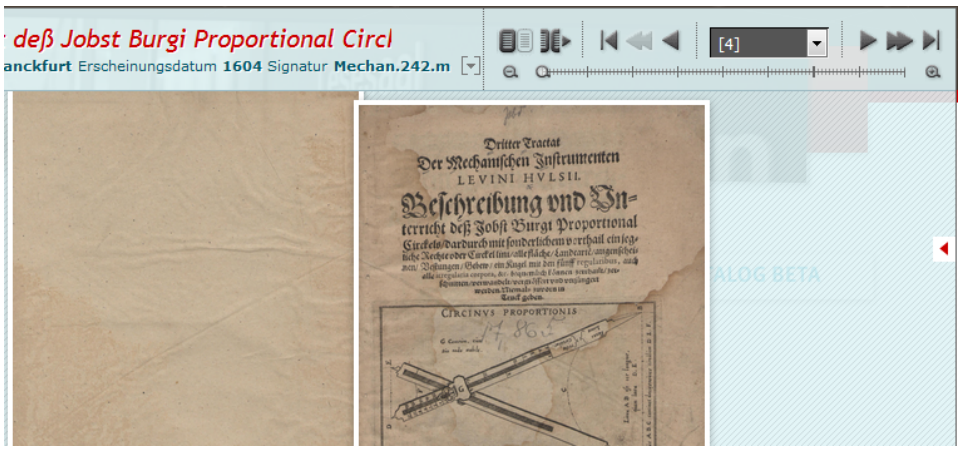

Abbildung 2.3: GOOBI Presentation (`http://digital.slub-dresden.de/werkansicht/dlf/16109/4/1/`)

GOOBI Presentation ist nicht auf das Format METS/MODS beschränkt sonder kann auch viele andere auf METS basierende Formate verwenden (z.B.: DC oder Text Encoding Initiative (TEI)).

Dadurch das GOOBI Presentation Open-Source ist, bietet es natürlich viele Erweiterungsmöglichkeiten. Wie bereits erwähnt ist GOOBI Presentation eine Erweiterung für TYPO3, was konkret bedeutet, dass eine TYPO3 Installation vorhanden sein muss um diesen Book-Viewer zu verwenden. Kritiker sehen TYPO3 als sehr komplexes System an, welches eine lange Einarbeitungszeit für Administratoren mit sich bringt. TYPO3 benötigt einen Apache, IIS oder NGINX Webserver und weicht somit von der Standard-Infrastruktur der Landesregierung (JBoss, Oracle) deutlich ab.

Eine Open Archives Initiative - Protocol for Metadata Harvesting (OAI-PMH) Schnittstelle bietet die Möglichkeit für eine internationale Suche der Metadaten. Dazu ist jedoch eine Eintragung in öffentliche OAI-Register erforderlich.

Die Unternehmen Zeutschel und Intranda bieten jeweils eigene, auf GOOBI Presentation basierende, Versionen an und kommen somit als möglicher Partner in Frage. Beide Unternehmen bieten die Möglichkeit den Book-Viewer an die eigenen Anforderungen anzupassen und bieten sowohl Support wie auch Mitarbeiterschulungen an.

Vorteile von GOOBI Presentation

- Großer Funktionsumfang
- Kann leicht erweitert und adaptiert werden (Open-Source)
- Keine Beschaffungs- und Lizenzkosten
- Bietet mit Intranda und Zeutschel zwei starke Partner
- Weite Verbreitung
- OAI-PMH Schnittstelle
- Viele unterstützte Formate (METS-basierend)
- Nicht auf den Einsatz von GOOBI Production beschränkt

Nachteile von GOOBI Presentation

- Hoher Einschulungsaufwand für TYPO3 wenn man selbst hostet
- Spezielle IT-Infrastruktur notwendig, da große Abweichung zur Standardinfrastruktur der Landesregierung-IT
- Keine Garantie auf Weiterentwicklung seitens des Herstellers
- Kein Support seitens GOOBI e.V.

2.2.3 Intranda Viewer

Der *Intranda Viewer* ist ein proprietäres Produkt der Firma Intranda. Der Book-Viewer (Beispiel siehe Abbildung 2.4) basiert auf Java Server Faces und JavaScript. Der Intranda-Viewer kann zum Beispiel gemeinsam mit GOOBI Production oder einer von Intranda modifzierten GOOBI Version verwendet werden.

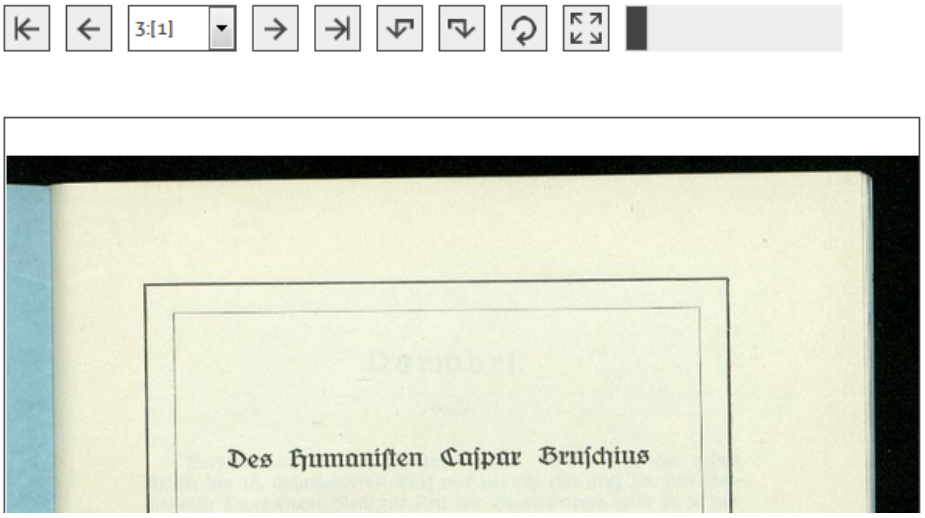

Abbildung 2.4: Intranda Viewer (`http://digi.landesbibliothek.at/viewer/image/AC00990706/3/`)

Als Infrastruktur setzt man auf eine Linux-Distribution mit einer Apache Tomcat Installation. Als Datenbankmanagementsystem kann zum Beispiel MySQL oder Oracle verwendet werden. Zur Metadaten- und Volltextsuche setzt der Book-Viewer auf Apach Solr [4] (basiert auf Apache Lucene). Weiters hat der Viewer bereits einen großen Funktionsumfang. Zum Beispiel [16]:

- Metadatenimport aus METS/MODS
- OAI-PMH Schnittstelle

- On-the-fly Erstellung von PDF-Dateien
- On-thy-fly Erstellung der Thumbnails
- Zugriffsrestriktionen (User, Gruppen, IP-Bereiche)
- User-Login
- OCR Integration mit Volltextsuche
- RSS-Feeds
- Lizenzmanagement
- Übersicht über alle vorhandenen Werke

Als zusätzliches Modul bietet der Intranda Viewer auch ein *Crowdsourcing Modul*, welches die interaktive Einbindung von Benutzern ermöglicht, an. Dabei können Benutzer zum Beispiel Kommentare abgeben oder Texte aus einem Werk, die mit OCR extrahiert wurden, direkt bearbeiten [16].

Vorteile des Intranda Viewer

- Großer Funktionsumfang der viele der Anforderungen bereits abdeckt
- OAI-PMH Schnittstelle
- Weite Verbreitung
- Kann gemeinsam mit GOOBI Production verwendet werden
- Design kann an die Anforderungen angepasst werden

Nachteile des Intranda Viewer

- Hoher Einschulungsaufwand für TYPO3
- Spezielle IT-Infrastruktur notwendig, da große Abweichung zur Standardinfrastruktur der Landesregierung-IT
- Proprietäre Software - keine eigene Weiterentwicklung möglich
- Ankaufs- und Lizenzkosten
- Abhängigkeit von der Firma Intranda

2.2.4 DFG-Viewer

Der *Deutsche Forschungsgemeinschaft (DFG)-Viewer* basiert auf TYPO3 und GOOBI. Er wurde im Auftrag der DFG hauptsächlich von der Sächsische Landesbibliothek Staats- und Universitätsbibliothek Dresden (SLUB) entwickelt und ist frei verfügbar (Open Source-Lizenz GPL3). Eine Weiterentwicklung und Anpassung an die eigenen Anforderungen ist somit leicht möglich.

Man kann den Viewer (Beispiel siehe Abbildung 2.5) entweder als eigene Instanz auf einem Webserver installieren oder die von der SLUB Dresden gehostete Version [6] über die METS Schnittstelle verwenden. Die Übergabe im METS-Format erfolgt als URL-Parameter. Der DFG-Viewer unterstützt zusätzlich zu METS/MODS noch das Datenformat METS/TEI [6].

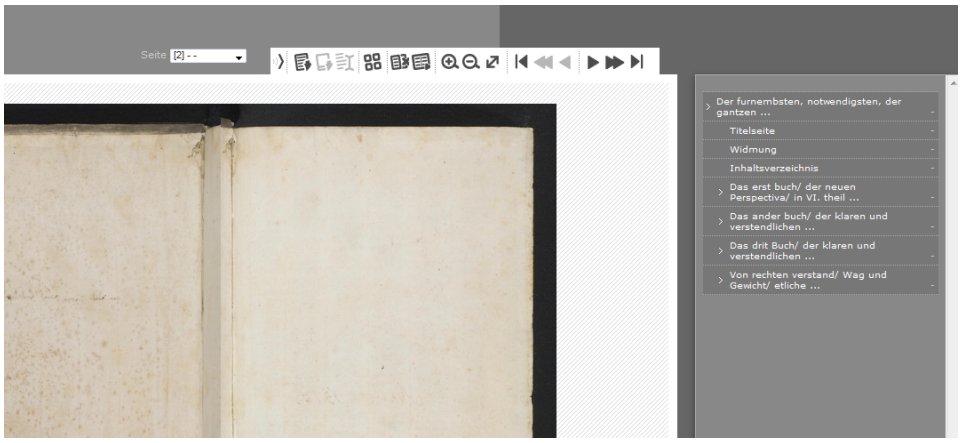

Abbildung 2.5: DFG-Viewer (`http://dfg-viewer.de/show/?tx_dlf[page]=2&tx_dlf[id]=http%3A%2F%2Fdigital.slub-dresden.de%2Foai%2F%3DGetRecord%26metadataPrefix%3Dmets%26identifier%3Doai%3Ade%3Aslub-dresden%3Adb%3Aid-263566811&tx_dlf[double]=1&cHash=93efdfbf434089840edcefcb7578a579`)

Der DFG-Viewer ist ein reiner Book-Viewer und bietet keine Übersicht über

vorhanden Werke. Der DFG-Viewer kann in seiner Standardfunktionalität folgende Funktionsvielfalt bieten [6]:

- Werke anzeigen (einseitig und doppelseitig)
- Inhaltsverzeichnis anzeigen
- Metadaten anzeigen
- Zoomen und scrollen
- Downloadlinks darstellen (Zu Einzelseiten oder für das gesamte Werk)
- In Volltexten suchen - sofern die Daten dazu im Analyzed Layout and Text Object (ALTO)-Format vorliegen

Viele Bibliotheken bieten den DFG-Viewer als zweiten oder sogar dritten Book-Viewer an. In Verwendung als einzigen Viewer in der Version auf `www.dfg-viewer.de` kann nur abgeraten werden, da dieser aus Performancegründen suboptimale Ergebnisse liefert.

Vorteile des DFG-Viewer

- Großer Funktionsumfang
- Kann auch über METS-Übergabe auf `www.dfg-viewer.de` benutzt werden
- Kann leicht erweitert und adaptiert werden (Open-Source)
- Keine Beschaffungs- und Lizenzkosten
- Weite Verbreitung
- OAI-PMH Schnittstelle
- Unterstützt METS/MODS und METS/TEI

Nachteile des DFG-Viewer

- Hoher Einschulungsaufwand für TYPO3
- Verarbeitung auf `www.dfg-viewer.de` dauert lange

- Bei Selbsthosting: Spezielle IT-Infrastruktur notwendig, da große Abweichung zur Standardinfrastruktur der Landesregierung-IT
- Keine Garantie auf Weiterentwicklung seitens des Herstellers

3 Beschreibung der Eigenentwicklung

Wie bereits erwähnt soll eine Eigenentwicklung als Pendant zu den bereits am Mark existierenden Lösungen entwickelt werden. Welche Vor- und Nachteile eine solche Eigenentwicklung mit sich bringt kann in den Kapiteln 1.3.1 und 1.3.2 nachgelesen werden. Es soll sowohl ein webbasiertes Administrationsportal für das Hochladen und Verwalten von Werken, als auch ein Book-Viewer für die grafische Aufbereitung entwickelt werden.

3.1 Administrationsportal

Das Administrationsportal soll den technischen Anforderungen der Landesbibliothek (Kapitel 1.2) entsprechen. Es soll im Prinzip die komplette Verwaltung von Werken ermöglichen.

- Hochladen von gescannten Werken
- Metadaten für ein Werk erstellen und bearbeiten (Abbildung 3.1)
- Metadatenimport aus *DABIS*
- Diverse Konvertierungsprozesse um Scans in mehreren Auflösungen zur Verfügung zu stellen - Dies garantiert ein schnelleres Laden der einzelnen Zoomstufen im Book-Viewer

- Beschreibung von einzelnen Seiten durch Zusatzinformationen (Abbildung 3.2)
- Metadatenindizierung mittels Apache Lucene
- Erstellen von Kategorien und Zuordnung von Werken

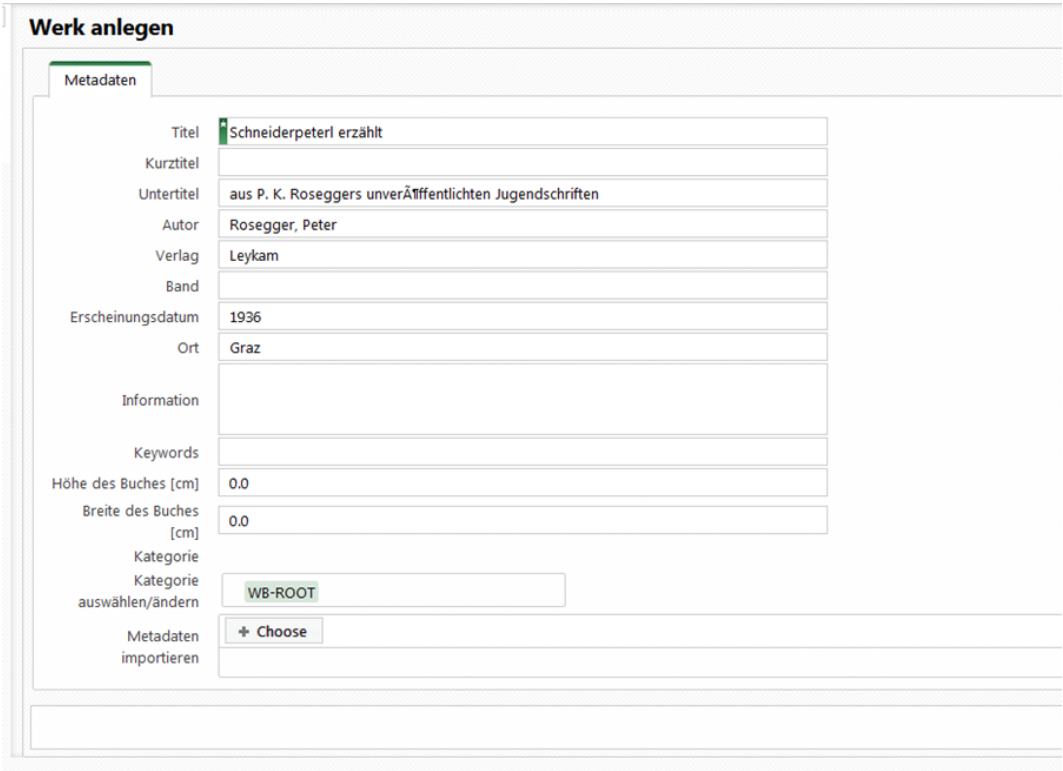

Abbildung 3.1: Bearbeitung von Metadaten im Admintool.

Das Administrationsportal läuft vollständig auf der Serverinfrastruktur der Landesregierung Steiermark und es kann schnell und zuverlässig an die Usermanagementebene *STERZ* [34] angebunden werden.

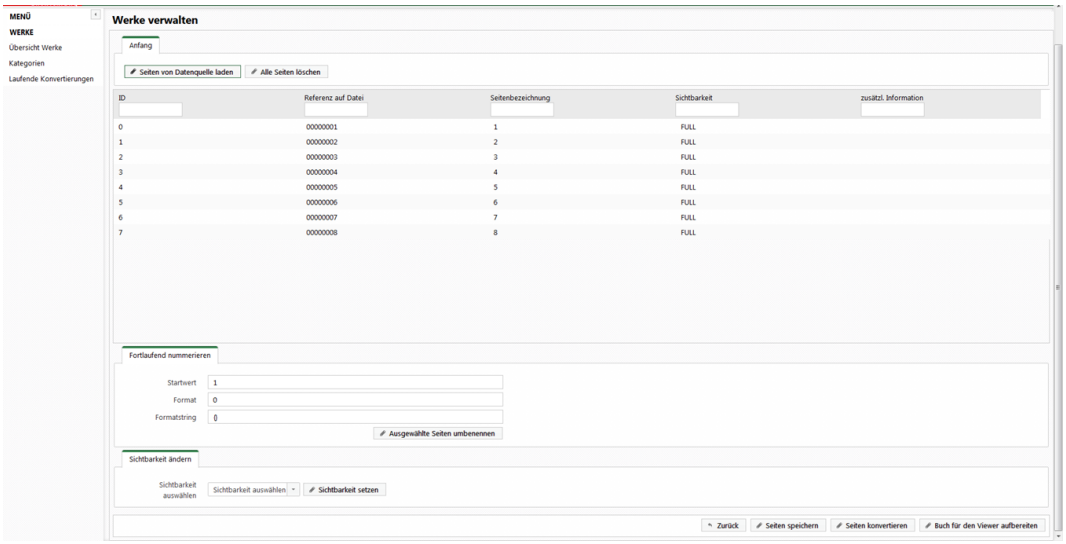

Abbildung 3.2: Bearbeitung von Seiten im Admintool.

3.2 Book-Viewer

Am IICM der TU Graz [15] wurde im Rahmen des Austria-Forums [39] bereits ein Book-Viewer entwickelt und erfolgreich eingesetzt. Dieser Book-Viewer soll auch für das Digitalisierungsprojekt der Landesbibliothek Steiermark zum Einsatz kommen. Marginale Änderungen sowohl im Design als auch bei der Funktionalität sollen in dieser, rein auf JavaScript basierenden, Webapplikation realisiert werden.

Der Book-Viewer verfügt bereits über einen ausgereiften Funktionsumfang. Dieser umfasst:

- Übersicht aller Werke in der Datenbasis inklusive Zuordnung zu den Kategorien (Abbildung 3.4)
- Werke finden durch Suche in den Metadaten
- Anzeigen und navigieren in einem Werk (Abbildung 3.3)
- Zoomen und Scrollen in einzelne Seiten

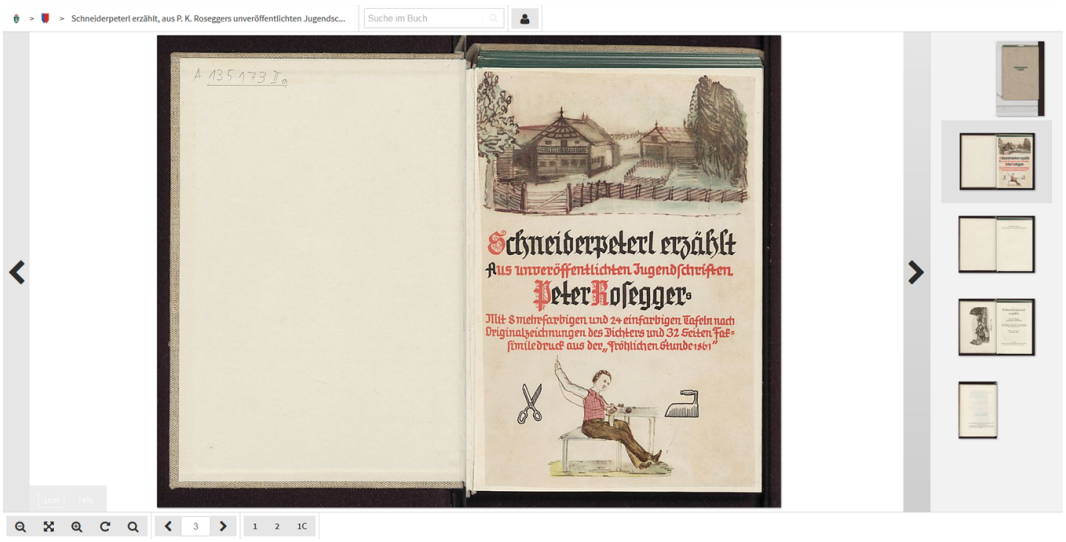

Abbildung 3.3: Anzeige von Seiten im Book-Viewer.

- OCR Volltextsuche in einem Werk und Highlighting der Suchergebnisse
- Seiten einzeln, nebeneinander oder als Übersicht darstellen
- Einfaches navigieren zu bestimmten Seiten über eine Übersichtsliste

Zusätzlich zu der bereits vorhandenen Funktionalität soll es noch möglich sein zu einer Seite bestimmte Informationen zu speichern und diese dann im Book-Viewer anzuzeigen. Der Hintergrundgedanke davon ist, dass viele Werke nur aus Zeichnungen bestehen und ein zusätzliches Informationsfeld dem Benutzer im Book-Viewer nähere Informationen geben soll.

3.2.1 Vorteile dieses Book-Viewers

- Ein sehr benutzerfreundlicher und einfach zu bedienender Book-Viewer
- Reagiert sehr schnell und gezielt

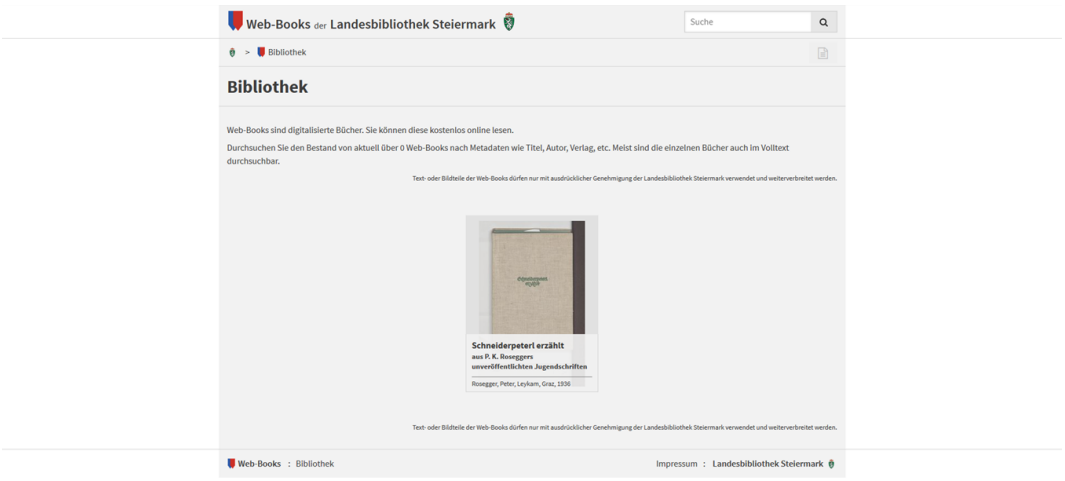

Abbildung 3.4: Übersicht aller Werke im Book-Viewer.

- Großer Funktionsumfang
- Übersicht über alle in der Datenbasis gespeicherten Werke möglich
- OCR-Volltextsuche möglich
- Eigenentwicklung, somit ist der Source-Code für Weiterentwicklungen vorhanden

3.2.2 Nachteile dieses Book-Viewers

- Keine OAI-PMH Schnittstelle
- Zur Zeit nicht mit anderen Administrationsportalen wie GOOBI koppelbar

4 Eingesetzte Technologien

Ausgehend von den Anforderungen der IT-Abteilung der Landesregierung Steiermark wurde auf Java Enterprise Technologien gesetzt. In den kommenden Kapiteln folgen kurze Beschreibungen und ein Überblick der verwendeten Technologien.

4.1 Java Enterprise Edition (Java EE)

Java EE ist eine weit verbreitete Spezifikation um Anwendungen, insbesondere Web-Anwendungen, auszuführen. Java EE ist hauptsächlich in der Programmiersprache Java entwickelt und bietet eine API für

- objektrelationale Abbildung,
- verteilte, mehrschichtige Architektur und
- Web Services.

Alle Komponenten der Java EE benötigen einen Java EE Application Server (AS) als Laufzeitumgebung. Der AS verwaltet:

- Transaktionen
- Sicherheit
- Skalierung
- Management der deployten Komponenten
- Regelt Zugriff auf OS Ressourcen wie Netzwerk und Filesystem

Eigentlich sind multi-tiered Applikationen aufwändig zu entwickeln, da viel komplexer Code für Transaktionsmanagement, Multithreading, etc. benötigt wird. Java EE erleichtert dies, da die Business Logic über wiederverwendbare Komponenten wie in Abbildung 4.1 realisiert wird.

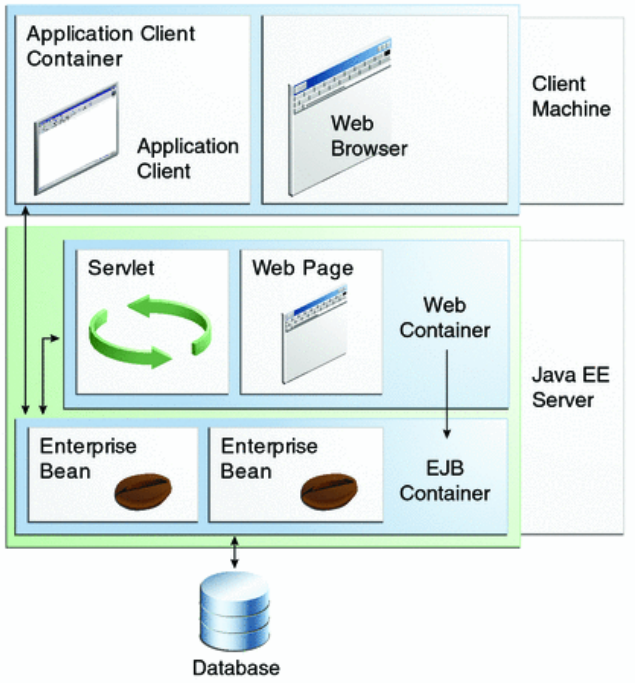

Abbildung 4.1: Java EE Container ([31])

Im Java EE Server läuft der **Web Container** und der **EJB Container**. Während im Web Container die Ausführung von Servlets und Webpages abläuft, verwaltet der EJB Container die Enterprise Beans für Anwendung. Der **Application Client Container** läuft am Server und verwaltet die Client Komponenten [31].

Java EE erweitert die Java Standard Edition (SE) und bietet APIs für viele verschiedene Technologien an.

4.2 Hibernate

Hibernate ist ein Object-Relational Mapping (ORM) Framework für Java. Es dient dazu Java Objekte, sogenannte POJOs (Plain Old Java Ojbect), in der Datenbank zu speichern und ist eine Implementierung der Java Persistence API (JPA). Es bringt viele Vorteile für die Entwicklung mit sich und ermöglicht bei Bedarf einen einfachen Austausch des Datenbankmanagementsystems. Abbildung 4.2 zeigt wie Hibernate zwischen der Datenbank und dem Client agiert.

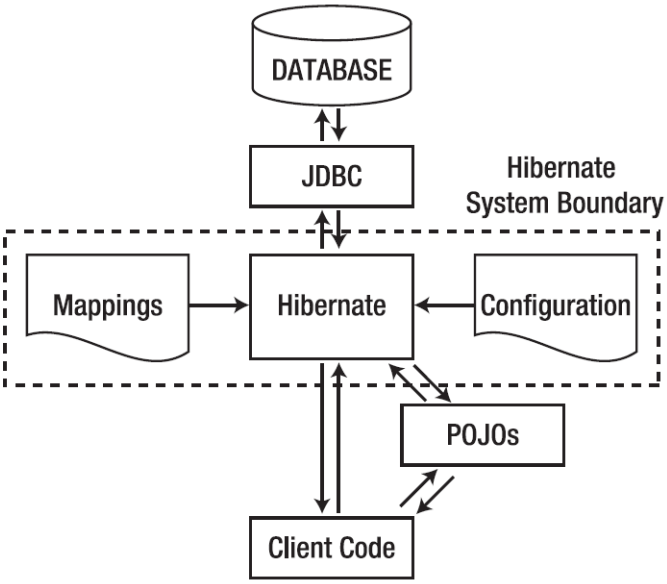

Abbildung 4.2: Hibernate ([25, S. 2]).

Ein kurzes Beispiel soll zeigen wie einfach die Persistierung von Objekten in der Datenbank mithilfe eines ORM Framework ist.

```
1 public Book createBook(String identifier, Scope scope)
2 {
```

```
3   log.info(String.format("////createBook(%s, %s)", identifier,
      scope));
4   if (getBook(identifier) != null)
5   {
6     return null;
7   }
8
9   synchronized (SYNC)
10  {
11    EntityManager em = getEm();
12    try
13    {
14      begin(em);
15      Book book = new Book(identifier, scope);
16
17      em.persist(book);
18      commit(em);
19
20      log.info(String.format("Create Book result=", book));
21      return book;
22    } catch (Exception e)
23    {
24      log.error("FAILED", e);
25      logException("error creating book: " + identifier, e);
26        rollback(em);
27    } finally
28    {
29      returnEm(em);
30    }
31  }
32
33  return null;
34 }
```

Listing 4.1: Persisiterung eines Objekts mit Hibernate

4.3 JavaServer Faces (JSF)

JavaServer Faces sind ein Framework für Webapplikationen. Es ist komponentenbasiert und soll es ermöglichen komplexe, grafische Benutzeroberflächen für Web 2.0 Applikationen zu erstellen. Die JavaServer Faces definieren, wie zum Beispiel Swing für lokale User-Interfaces, ein Komponentenmodell für Benutzerschnittstellenelemente. Komponentenbasiert bedeutet, dass man sowohl eigene Komponenten entwickeln, wie auch Komponenten von Drittherstellern verwenden kann. Will man zum Beispiel eine HTML-Tabelle erstellen, so werden nicht die Reihen und Spalten definiert sondern eine Tabellen-Komponenten der Seite hinzugefügt. JavaServer Faces basieren auf der Servlet Technologie. Um JavaServer Faces verwenden zu können benötigt man einen Servlet-Container der zum Beispiel in Anwendungsservern wie JBoss läuft [11] [28].

JavaServer Faces sind im Grunde genommen nur eine Spezifikation und keine Implementierung. Referenzimplementierungen dieser Spezifikation sind zum Beispiel

- Mojarra [27] von Oracle und
- MyFaces [1] von Apache.

Um eine klare Trennung von Ansicht (*View*) und Logik (*Controller*) zu ermöglichen, setzen auch die JSF auf das **Model View Controller (MVC)** Pattern wie in Abbildung 4.3 dargestellt.

In der *View* können also diverse Komponenten verwendet werden, um die vielen Vorteile von JSF zu nutzen. Zusätzlich zu den standardmäßig ausgelieferten Komponenten von JSF gibt es ja auch noch die Möglichkeit Komponenten von Drittherstellern zu verwenden. Hier sind natürlich die jeweiligen Lizenzbestimmungen zu beachten. Selbstverständlich kann auch pures HTML mit dem gesamten Funktionsumfang verwendet werden. Mit den

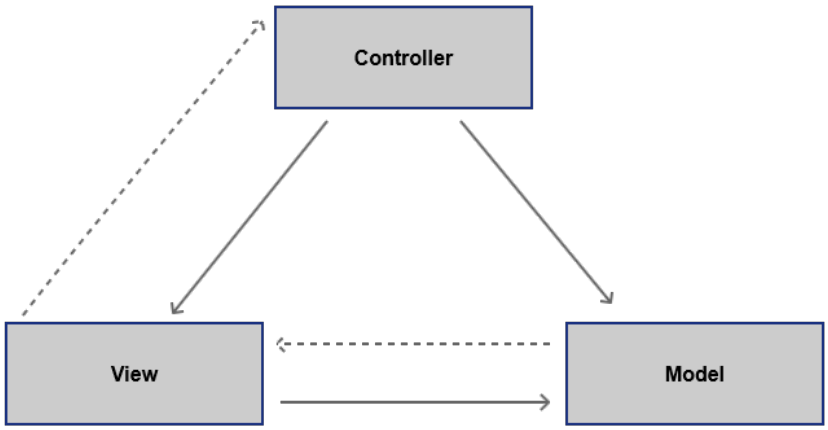

Abbildung 4.3: MVC

vielen Möglichkeiten an Komponenten bieten JSF eine üppige Möglichkeit zur Gestaltung von Oberflächen [28].

Managed Beans bieten die Möglichkeit den Controller in JSF zu realisieren. Sie werden dabei mit bestimmten Annotationen versehen, beinhalten die Programmlogik und können gezielt auf Interaktionen am User-Interface reagieren. Dabei werden durch das Framework die zugehörigen Methoden in den Managed Beans aufgerufen und ausgeführt. Die Verwaltung der Managed Beans obliegt ebenfalls dem Framework und somit braucht sich der Entwickler nicht explizit darum kümmern. Die Gültigkeitsdauer von Managed Beans wird auch über Annotationen festgelegt [28].

Für das *Model* gibt es keine genaue Vorgabe seitens der JSF Spezifikation. Im Prinzip bleibt es dem Entwickler überlassen ob er ein Model einführen will oder nicht. Meist hält das Model applikationsspezifische Funktionalitäten und Daten. Die Managed Beans können aber auch direkt als Model dienen [28].

4.3.1 JSF Lifecycle

Die Zustandslosigkeit von HTTP(S), auf dem alle Webapplikation basieren, ist eine Eigenschaft die nicht immer erwünscht ist. Durch die Zustandslosigkeit von HTTP-Requests und HTTP-Responses wäre es zum Beispiel nicht möglich eine Webanwendung zu realisieren bei der ein Benutzer entweder den Status “eingeloggt“ oder “ausgeloggt“ hat. JavaServer Faces basieren zwar auf Servlets, diese aber wiederum auf dem Request-Response-Model von HTTP(S) und besitzen daher diese Zustandslosigkeit. Es gibt über den JSF Lifecycle aber Möglichkeiten zustandsbehaftete Operationen zu realisieren [28].

Im „Request Processing Lifecycle“ (Abbildung 4.4) wird beschrieben wie JSF beim Verarbeiten von Anfragen (Request) und beim Generieren der Antwort (Response) vorgeht. JSF bieten darin eine automatische Verwaltung von Zuständen.

Für die Servlet Anwendung muss im Deployment-Deskriptor das *Faces-Servlet* (graue Box in Abbildung 4.4) definiert werden. Über dieses Servlet gehen allen Anfragen ein und es werden schließlich auch alle Antworten über dieses Servlet gesendet. Auch die Ressourcenbereitstellung erfolgt darüber. So werden zum Beispiel CSS-Dateien, Grafik-Dateien wie auch JavaScript-Dateien für das Faces-Servlet bereitgestellt. Diese werden bei einer Anfrage sofort herausgefiltert und bereitgestellt und müssen somit nicht den ganzen Lifecycle durchlaufen. Dadurch wird bei einem Request wie

```
1 http://hello-world.org/firstproject/images/picture.png
```

direkt auf die Ressourcen zugegriffen. Andere Anfragen müssen immer den gesamten Lifecycle durchlaufen [28].

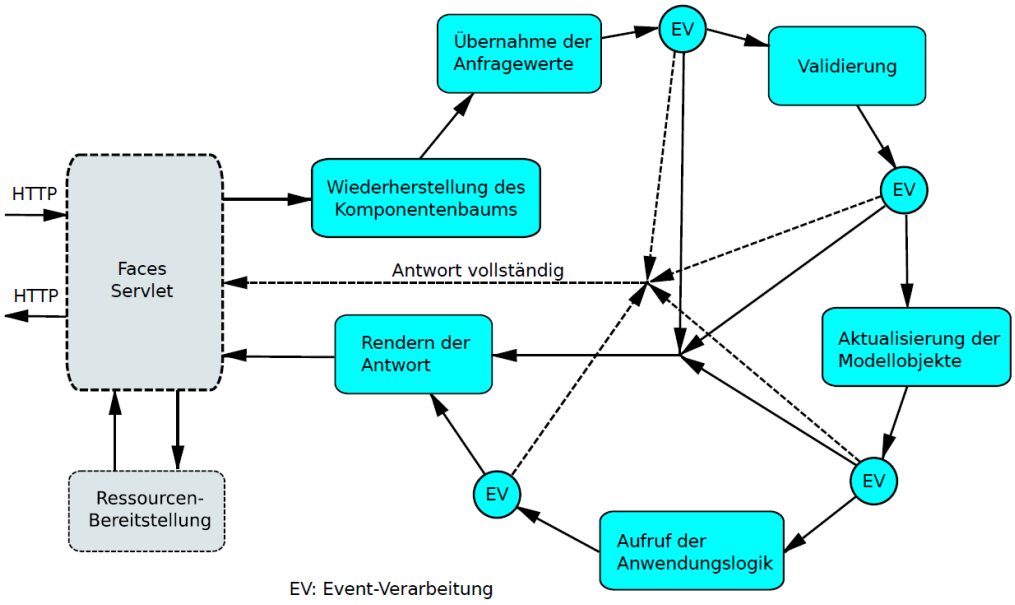

Abbildung 4.4: JSF Lifecycle [28]

Wiederherstellung des Komponentenbaums

JSF-Komponenten sind mit bestimmten Zuständen behaftet. Zum Beispiel speichert ein Drop-Down-Menü das aktuelle selektierte Element. Die View ist dabei die Wurzel des Baums welcher aus allen Komponenten einer Seite besteht. Die Komponenten werden pro Seite und Client-Session aufgebaut, da man zwischen zwei Anfragen nicht alle Komponenten benötigt. Deshalb muss zu Beginn eines Requests der Komponentenbaum wiederhergestellt werden. Gespeichert wird der Komponentenbaum im *FacesContext*, welcher alle Informationen zur Abarbeitung eines Requests enthält [28].

Übernahme der Anfragewerte

Bei manchen UI-Komponenten ist es möglich das der Benutzer aktiv einen Wert oder eine Auswahl der Komponente zuweisen kann oder muss. So zum Beispiel bei einem Input-Feld oder bei einem Drop-Down-Menü. Mittels POST-Parameter und in einem HTTP-Request kodiert werden die Daten dann an das JSF-Framework weitergereicht. Dieses hat die Aufgabe die Daten zu dekodieren und an die zuständige UI-Komponente weiterzuleiten. Ist die Phase zu Ende werden noch alle Events an ihre Listener weitergereicht. Jetzt kann das JSF-Framework entweder die Anfrage komplett beenden, zur Phase "Rendern der Antwort" springen oder mit der "Validierung" fortfahren [28].

Validierung

Um ungültige Eingaben des Benutzers zu verhindern bietet das Framework die Möglichkeit der Validierung. Es ist zum Beispiel sinnvoll im Eingabefeld für das Geburtsjahr nur numerische und vierstellige Ziffern zu erlauben. Alle registrierten Validierer einer Komponente überprüfen also die Eingaben des Benutzers und bei erfolgreicher Validierung wird der Wert der Komponente zugewiesen [28].

Aktualisierung der Modellobjekte

Die Bearbeitung belief sich bis dahin rein auf die UI-Komponenten. Jetzt werden die validen Eingaben des Benutzers an die zugehörigen Managed Beans weitergegeben. Dies passiert über die Setter-Methoden der Bean [28].

Aufruf der Anwendungslogik

Bis jetzt wurde alles vom JSF-Framework automatisch erledigt. Die Eingabewerte wurden validiert und die Properties der Managed Beans wurden mit Daten gefüllt. Jetzt geht es darum den eigentlichen Applikationscode auszuführen. Dies passiert üblicherweise über die Listener *action* und *actionListener* [28].

Rendern der Antwort

Im letzten Schritt sind das Rendern der Antwort und das Abspeichern des Komponentenbaums die zentralen Aufgaben des Frameworks. Durch die Spezifikation der JSF wurde die Zielsprache offen gelassen. Es wären also zum Beispiel folgende Sprachen möglich [28]:

- XHTML
- JSP
- SVG

Am häufigsten werden in JSF der Version 2 XHTML-Antworten generiert. Die Komponentenwerte die bei der Anfrage zuerst dekodiert wurden, müssen jetzt kodiert werden und der Komponentenbaum muss persistiert werden [28].

4.3.2 Managed Beans

Die Managed Beans werden als simple Java-Beans (Plain Old Java Object (POJO)) definiert und haben im konkreten folgende Aufgaben:

- Daten von den UI-Komponenten für die eigenen Properties übernehmen

- Implementieren der Event-Listener
- Unterstützungsaufgaben für UI-Komponenten

Die Managed Beans sind durch die Annotation *@ManagedBean* gekennzeichnet. Die Managed Beans werden durch das JSF-Framework automatisch instanziiert wenn sie benötigt werden.

Wie bereits in einem vorhergehenden Kapitel erwähnt, wird die Gültigkeitsdauer von Beans ebenfalls über Annotationen festgelegt. Gültige Annotation sind:

- **Request (@RequestScoped)**
  Nur gültig solange der aktuelle Request verarbeitet wird
- **View (@ViewScoped)**
  Nur gültig solange die aktuelle View gültig ist
- **Session (@SessionScoped)**
  Nur gültig solange eine Session gültig ist
- **Application (@ApplicationScoped)**
  Es gibt nur eine Instanz dieser Managed Bean für die ganze Applikation
- **None (@NoneScoped)**
  Gültigkeit ist gleich wie die Gültigkeit der Bean in der sie injiziert wurde

Alle Managed Beans sind auch für die Verwendung in der View über die Expression Language definiert. Darauf wird aber später im Kapitel 4.3.3 genauer eingegangen.

Es gibt noch viele weitere Annotationen. Zwei Annotationen die nicht unerwähnt bleiben sollten sind *@PostConstruct* und *@PreDestroy*. Methoden die damit annotiert sind werden beim Instantiieren bzw. beim Zerstören einer Bean ausgeführt [28].

Das Codebeispiel 4.2 zeigt die Verwendung von Annotationen in einer Managed Bean.

```
@ManagedBean
@ViewAccessScoped
public class MyBean implements Serializable
{
  private int val;

  @PostConstruct
  public void init()
  {
    store = prefs.storerw();
  }

  ...

}
```

Listing 4.2: Beispiel Managed Bean

4.3.3 Expression Language

Die *Expression Language (EL)* bietet einfache Ausdrücke um auf Daten von JavaBeans dynamisch zuzugreifen und hat die Eigenschaft, dass sie erst zur Laufzeit ausgewertet wird. Es gibt EL die sofort und EL die verzögert ausgewertet wird. Man bezeichnet diese EL *Immediate Expression* bzw. *Deferred Expression*. Die sogenannten Immediate Expressions beginnen mit einem *Dollar*-Zeichen und werden sofort, also beim Zeitpunkt des Renderings der JSF-Seite, ausgeführt. Deferred Expressions beginnen hingegen mit einem *Raute*-Zeichen und werden erst zur Laufzeit ausgeführt. Betrachtet man den Zeitpunkt der Auswertung im JSF-Lifecycle so werden in der zweiten (Übernahme der Anfragewerte) und sechsten Phase (Rendern der Antwort) die Ausdrücke ausgewertet.

Zur eigentlichen Syntax der EL: Die Ausdrücke der EL beginnen mit dem Startsymbol gefolgt von geschwungenen Klammern in denen dann der eigentliche Ausdruck steht.

Das folgende Beispiel zeigt EL Ausdrücke in einer JSF-Datei:

```
<h:panelGrid>
  <h:outputText value="Expression Language Beispiele" />
  <h:outputText value="#{Bean.string}" />
  <h:outputText value="#{Bean.array[2]} " />
  <h:outputText value="#{Bean.map['key']}" />
</h:panelGrid>
```

Listing 4.3: Beispiel Expression Language

Anwendungsgebiete von EL sind:

- Zugriff auf Attribute von Beans über Punktnotation (siehe Beispiel 4.3). Dabei werden in der Bean-Klasse die zugehörigen Setter- und Getter-Methoden getriggert
- Werteausdruck
- Methodenausdruck
- Einfache arithmetische oder logische Ausdrücke

Syntaxelemente

Die EL bietet viele Syntaxelemente wie Vergleichsoperatoren, logische Operatoren etc. Einige dieser Elemente werden in der nachfolgenden Tabelle 4.1 aus [28, S. 48] aufgelistet.

Diese Syntaxelemente können in den Facelets so benützt werden wie es von Java bekannt ist. Nachfolgend ein Beispiel für Syntaxelemente.

```
1 <h:outputText value="#{Bean.value <= 1 ? myBean.value : ""}"/>
2 <h:outputText value="#{Bean.sum[1] - myBean.sum[2]}" />
3 <h:outputText value="#{Bean.var" />
```

Listing 4.4: Beispiel Syntaxelemente

Vordefinierte Variablen

Einige Variablen (siehe Tabelle 4.2 aus aus [28, S. 48]) werden von JSF implizit vordefiniert. Sie dienen dazu um auf Eigenschaften der zugrundeliegenden Servlet- und JSF-Implementierung zuzugreifen.

```
1 5 <h:outputText value="CL: #{header['Content-Length']}" />
```

Listing 4.5: Beispiel für vordefinierte Variablen

4.3.4 Konfiguration von JSF Implementierungen

Seit Version 2.0 der JavaServer Faces wird bei der Konfiguration der Implementierung großer Wert auf die einfach zu verwendenden Annotationen gelegt. In frühere JSF-Versionen musste viel Konfigurationsaufwand in eigenen Dateien erfolgen.

Betrachtet man sich die weiteren Konfigurationsdateien genauer, bemerkt man das einige Konfigurationen mit dem darunter liegenden Servlet-System und manche direkt mit den JSF-System zusammenhängen.

Servlet Konfiguration

Der sogenannte *Deployment Deskriptor* ist auch als *web.xml* Datei im *WEB-INF* Ordner des Webprojekts bekannt. Der Deployment Deskriptor enthält alle

Servlets der Anwendung. Auch weitere Ressourcen können darin deklariert werden. Dazu spezifiziert JSF genau ein Servlet, auch *Faces-Servlet* genannt, welches die eigentliche Webanwendung realisiert.

Der Codeausschnitt 4.6 zeigt wie das Faces-Servlet als XML-Schema korrekt deklariert wird. Es zeigt auch das Servlet-Mapping für *xhtml*-Dateien.

```
<?xml version="1.0" encoding="ASCII"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://java.sun.com/xml/ns/javaee" xmlns:web="http://java
    .sun.com/xml/ns/javaee/web-app_2_5.xsd"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://
    java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
  version="3.0">
  <display-name>lbstdig</display-name>
  ...

  <servlet>
    <servlet-name>Faces Servlet</servlet-name>
    <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
  </servlet>
  ...
  <servlet-mapping>
    <servlet-name>Faces Servlet</servlet-name>
    <url-pattern>*.xhtml</url-pattern>
  </servlet-mapping>
</web-app>
```

Listing 4.6: Servlet Definition in WEB-INF/web.xml

In der Datei web.xml können auch sogenannten *Kontext-Parameter* eingetragen werden. Sie dienen dazu der Webanwendung bestimmte Initialisierungsparameter zur Verfügung zu stellen. Diese Kontext-Parameter werden über das Element *<context-param>* definiert.

Das Servlet-Mapping im Codausschnitt 4.6 besagt, dass alle Anfragen von *xhtml*-Dateien den JSF-Lifecycle durchlaufen.

JSF Konfiguration

In der Servlet-Konfiguration wird auch definiert wo das File zur JSF Konfiguration (*faces-config.xml*) liegt. Wie auch schon die Servlet-Konfiguration wird auch die JSF-Konfiguartion im XML-Standard angegeben. In diesem File können viele Konfigurationen vorgenommen werden. Zum Beispiel:

- Beans deklarieren
- Eigene Validatoren registrieren
- Eigene Konverter registrieren
- Eigene Renderer registrieren
- Navigationsregeln definieren
- Eigene Komponenten registrieren

Einige Elemente, wie zum Beispiel Managed Beans, können sowohl in der XML-Datei als auch als Annotation angegeben werden.

4.3.5 AJAX and JSF

Asynchronous JavaScript and XML (AJAX) ist eine Technologie die es einem Client ermöglicht Anfragen auszuführen und die Ergebnisse im Browser darzustellen ohne die Website neu zu laden. AJAX ist ein elementarer Bestandteil des Web 2.0 und Basis vieler interaktiver Anwendungen. Vom Konzept hier ist AJAX nicht komplex. Wie ein AJAX-Request von einem Browser an den Server aussieht illustriert Abbildung 4.5.

In JSF 2.0 wird der Lifecycle in zwei Teile geteilt [11]:

- **execute** (Abbildung 4.6)
- **render** (Abbildung 4.7)

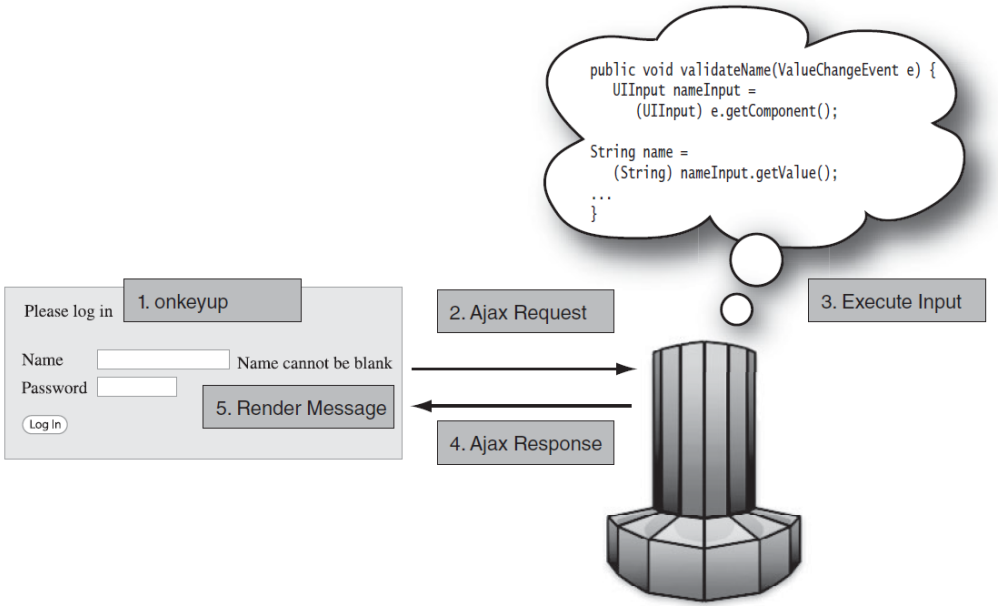

Abbildung 4.5: AJAX Request ([11, S. 386]).

Der *execute* Teil 4.6 ist für die Ausführung des Inputs am Server zuständig. Falls die Komponente ein *Input* ist wird am Server zuerst konvertiert und validiert. Dann wird, falls die Komponente mit einer Bean verknüpft ist, der Wert an die Bean übergeben und anschließend werden, falls die Komponente eine *action* ist, die Actions und Action Listeners ausgeführt. In Abbildung 4.5 wird diese durch Schritt 3 repräsentiert.

Der *render* Teil in Abbildung 4.7 ist, wie es der Name schon impliziert, für das Rendern der Komponenten beim Client verantwortlich.

4.3.6 Konvertierer

Es wurde bereits kurz erwähnt, dass man Properties von Managed Beans an UI-Komponenten binden kann, lesend wie auch schreibend. HTML/HTTP

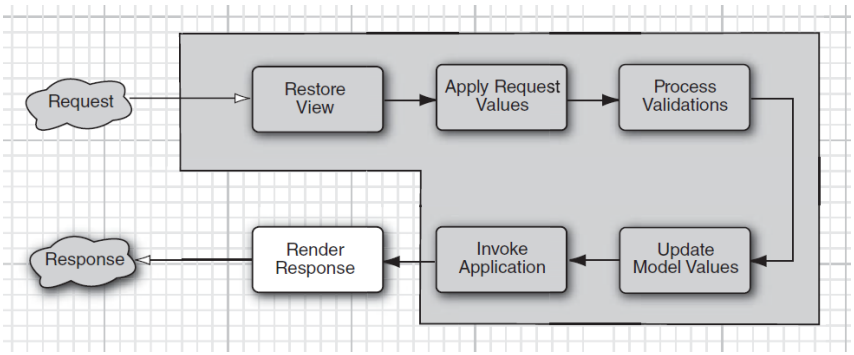

Abbildung 4.6: AJAX Execute ([11, S. 387]).

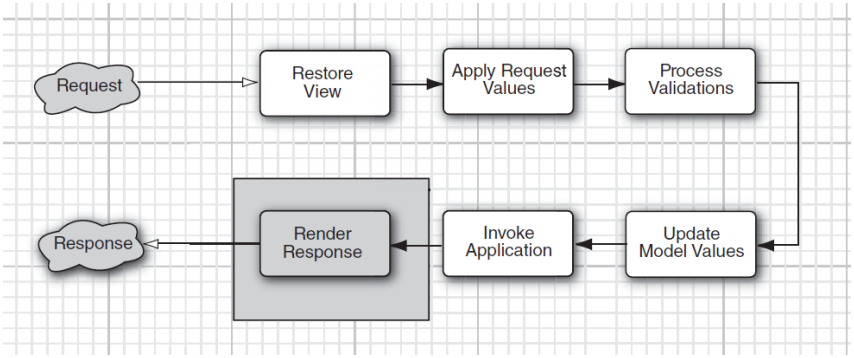

Abbildung 4.7: AJAX Render ([11, S. 387]).

arbeitet String-basiert, aber die Properties von Managed Beans erwarten sich häufig andere Datentypen. Dafür gibt es bereits eine große Anzahl von Standardkonvertierern, die dem Entwickler zur Verfügung stehen. Es ist jedoch auch möglich eigene Konvertierer zu implementieren und diese zu verwenden. Wichtig ist, dass alle selbst implementierten Konvertierer vom Interface *javax.faces.convert.Converter* ableiten und die Methoden

- *getAsObject* und
- *getAsString*

implementieren [28].

Vordefinierte Konvertierer

Das JSF Framework bietet zum Beispiel für Zahlen und Kalenderdaten bereits einige vorgefertigte Konvertierer. Speziell für die Datentypen *BigDecimal, BigInteger, Character, Boolean, Byte, Integer, Short, java.util.Date, Long, Float, Double* und *Enum* [28] hat das JSF Framework bereits solche Standardkonvertierer implementiert. Die aufgelisteten Konvertierer sind, bis auf *java.util.Date*, nicht konfigurierbar. Der zugehörige Tag für die Konfiguration von *java.util.Date* lautet *<f:convertDateTime>*.

Die Standardkonvertierer befinden sich im Package *javax.faces.convert*.

Eigene Konvertierer

Will man einen eigenen Konvertierer schreiben, ist es notwendig das Interface *javax.faces.convert.Converter* zu implementieren. Eingesetzt kann der Konvertierer entweder global für eine Klasse (Attribut *forClass*) oder für ein bestimmtes Binding werden [28]:

- **Global für eine Klasse**
  Die Annotation *FacesConverter* definiert Standardkonvertierer für eine Managed Bean (Codebeispiel 4.7).

```
1 @FacesConverter(forClass = SelfConverter.class)
2 public class MeineBean {
3   ...
4 }
```

Listing 4.7: Globaler Konvertierer [28]

- **Für ein Binding**
  Eine Methode *getMyConverter* ist dafür zuständig eine Instanz des Konvertierers zurückzugeben. Das Codebeispiel 4.8 zeigt einen Konvertierer mit einem bestimmten Binding.

```
<h:inputText value="#{meineBean.i}" converter="#{meineBean.
    myConverter}" />
```

Listing 4.8: Konvertierer für ein Binding [28]

4.3.7 Validierung

Eine Validierung von Daten setzt immer eine Kompatibilität der Datentypen voraus. So ist oft notwendig die Daten vor der eigentlichen Validierung zu konvertieren. Wie bereits erwähnt ist die automatische Validierung ein besonderes Feature der JSF und erleichtert den Entwicklern den Implementierungsprozess. Wie schon bei den Konvertierern, gibt es auch bei den Validierern einige vordefinierte Standardvalidierer. Alle Validierer, sowohl vordefinierte wie auch selbst implementierte, müssen vom Interface *javax.faces.validator.Validator* ableiten [28].

Ist eine Benutzereingabe valide, dann ist die Arbeit der Validierer eher unauffällig. Ist die Benutzereingabe jedoch nicht wie definiert so schlägt einer der Validierer an und es muss für eine aussagekräftige Fehlermeldung für den User gesorgt werden [11].

Vordefinierte Validierer

Die vordefinierten Validierer der JSF werden über die Attribute im Facelet gesteuert. Vordefinierte Validierer sind:

- LengthValidator
- LongRangeValidator
- DoubleRangeValidator
- RegexValidator
- RequiredValidator

Eine Steuerung der Validierer erfolgt entweder über einen eigenen Tag wie zum Beispiel *<f:validateLongRange>* oder ist attributgesteuert. Eine *required* Attribut führt dazu das automatisch der zugehörige *RequiredValidator* ausgeführt wird. Das Beispiel 4.9 zeigt beide Arten für die Steuerung von Validatoren [11].

```
<h:inputText value="#{myBean.longValue}" required="true">
  <f:validateLongRange minimum="1" maximum="350" />
</h:inputText>
```

Listing 4.9: Vordefinierte Validierer [28]

Eigene Validierer

Wie auch bei den Konvertierern gibt es bei den Validierern die Möglichkeit eigene Validierer zu implementieren. Die Verwendung ist analog wie die Verwendung bei den Konvertierern. Man kann für ein bestimmtes Binding oder für die gesamte Bean einen Validierer definieren.

```
<h:inputText value="#{meineBean.i}" validator="#{meineBean.
    meinValidator}" />
```

Listing 4.10: Eigene Validierer für eine Klasse [28]

```
@FacesValidator("meinValidierer")
public class MeinValidierer implements Validator {
  ...
}
```

Listing 4.11: Eigene Validierer für ein bestimmtes Binding [28]

4.3.8 UI Komponenten

JSF Komponenten sind UI Elemente die definiert und immer wieder verwendet werden können (keine Redundanzen). Sieht man sich so ein JSF-File

an, werden die Komponenten am Anfang der Datei definiert.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:h="http://java.sun.com/jsf/html"
xmlns:f="http://java.sun.com/jsf/core"
xmlns:ui="http://java.sun.com/jsf/facelets"
xmlns:g="http://sterz.stlrg.gv.at/jsf/gasworks"
xmlns:layout="http://sterz.stlrg.gv.at/jsf/layout"
xmlns:p="http://primefaces.org/ui">
```

Listing 4.12: Komponenten

Es gibt eine Reihe von Standardkomponenten die in JSF verfügbar sind (siehe Abbildung 4.8). Zusätzlich zu den Standardkomponenten werden in 4.12 weitere Komponenten eingebunden. Die vergebenen Präfixe wie zum Beispiel *xmlns:layout* können dabei frei gewählt werden.

Dabei sind

- *xmlns:g="http://sterz.stlrg.gv.at/jsf/gasworks"* und
- *xmlns:layout="http://sterz.stlrg.gv.at/jsf/layout"*

Komponenten der Landesregierung selbst. Die letzte Komponente im Beispiel, *Primefaces* [32], ist eine von vielen zur Verfügung stehenden Third-Party-Komponenten und hat einen hohen Verbreitungsgrad.

4.3.9 Navigation

In modernen Webanwendungen, aber vor allem in modernen Unternehmensanwendungen, werden ganz neue Anforderungen an die Navigation auf einer Website gelegt. Früher wurde die Navigation ganz einfach über

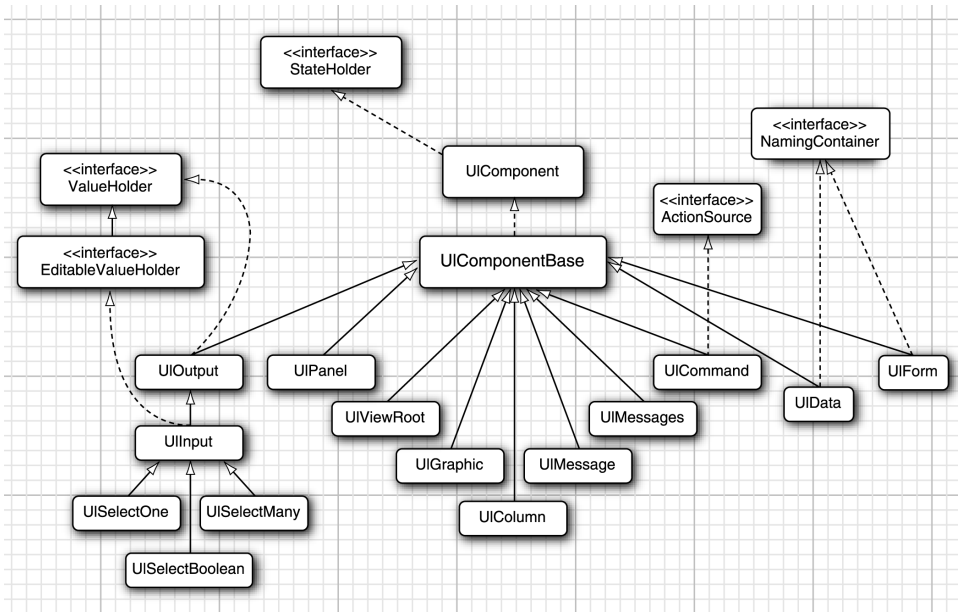

Abbildung 4.8: JSF Komponentenhierarchie ([11, S. 422]).

Navigationslinks (*<a>-Tag*) in einer Navigationsleiste gelöst. Mittlerweile soll die Navigation, genauso wie der gesamte Aufbau der Website, viel dynamischer gestaltet werden. JSF bieten dafür eine eigene Navigationskomponente, den *Navigation-Handler*.

Implizite Regeln für Navigation

Implizite Regeln erinnern am meisten an einen normalen Link. Gibt man eine View-ID direkt als Rückgabewert in der *action* Methode oder im JSF-Tag an, so spricht man von einer impliziten Regel. Das Beispiel 4.13 zeigt die Möglichkeiten der impliziten Navigation.

```
1 <h:commandButton action="/home.xhtml" />
2
3 <h:commandButton action="#{meineBean.getHomePage}" />
4
```

```
5 public String getNextPage() {
6   return "contact.xhtml";
7 }
```

Listing 4.13: Implizite Navigation in JSF [28]

Explizite Regeln zur Navigation

Über Konfigurationsdateien in JSF können explizite Regeln definiert werden. Diese Konfigurationsdatei ist XML-basiert und besteht aus den Elementen wie in Codeauszug 4.14.

```
1 <navigation-rule>
2   <from-view-id>
3   <navigation-case>*
4   <from-action>?
5   <from-outcome>?
6   <if>?
7   <to-view-id>
8   <redirect>?
```

Listing 4.14: Explizite Navigation in JSF [28]

Umschlossen werden die Regeln mit dem Tag *<navigation-rule>*. Der *<from-view-id>* gibt an für welche Seiten die Regel gültig ist. Es besteht auch die Möglichkeit Wildcard-Symbole zu verwenden (*). Anschließend können beliebig viele *<navigation-case>* Elemente folgen. Das Framework sieht sich dabei den Output des *action*-Attributes aus der UI-Komponente an und sieht nach, ob es ein passendes *<from-outcome>* Element gibt (Beispiel 4.15).

```
1 <navigation-rule>
2   <from-view-id>/page/*</from-view-id>
3   <navigation-case>
4     <from-outcome>login</from-outcome>
5     <to-view-id>/page/login.xhtml</to-view-id>
6   </navigation-case>
```

```
</navigation-rule>
```

Listing 4.15: Explizite Navigation in JSF [28]

Fügt man jeder *action*-Methode das Codesegment 4.16 hinzu, so führt dies dazu, dass man immer auf die Login-Seiten weitergeleitet wird wenn der Benutzer nicht eingeloggt ist.

```
if (!user.isLoggedIn())
  return "login";
```

Listing 4.16: Explizite Navigation anhand eins Login-Beispiels [28]

Tabelle 4.1: Syntaxelemente EL

Operator	**Alternative**	**Beschreibung**
.		Zugriff auf ein Property, eine Methode oder einen Map-Eintrag
[]		Zugriff auf ein Array- oder Listen-Element oder einen Map-Eintrag
()		Klammerung für Teilausdrücke
?:		Bedingter Ausdruck
+		Addition
-		Subtraktion oder negative Zahl
*		Multiplikation
/	div	Division
%	mod	Modulo
==	eq	gleich
!=	ne	ungleich
<	lt	kleiner
>	gt	größer
<=	le	kleiner-gleich
>=	ge	größer-gleich
&&	and	logisches UND
||	or	logisches ODER
!	not	logische Negation
empty		Test auf null, einen leeren String, oder Test auf Array, Map oder Collection ohne Elemente

Tabelle 4.2: Vordefinierte Variablen

Variablenname	**Beschreibung**
header	Eine *Map* von Request-Header-Werten. Schlüssel ist der Header-Name, Rückgabewert ist ein String.
headerValues	Eine *Map* von Request-Header-Werten. Schlüssel ist der Header-Name, Rückgabewert ist ein Array von Strings.
cookie	Eine Map von Cookies (Klasse *Cookie* im *Package javax.servlet.http*). Schlüssel ist der Cookie-Name.
initParam	Eine *Map* von Initialisierungsparametern der Anwendung (Application-Scope). Diese werden im Deployment-Deskriptor definiert.
param	Eine *Map* von Anfrageparametern. Schlüssel ist der Parametername. Rückgabewert ist ein String.
paramValues	Eine *Map* von Anfrageparametern. Schlüssel ist der Parametername. Rückgabewert ist ein Array von Strings.
facesContext	Die *FacesContext*-Instanz der aktuellen Anfrage.
component	Die im Augenblick bearbeitete Komponente.
cc	Die im Augenblick bearbeitete zusammengesetzte Komponente.
resource	Eine *Map* von Ressourcen.
flash	Eine *Map* von temporären Objekten für die nächste View.
view	Das View-Objekt (View-Scope).
viewScope	Eine *Map* von Variablen mit View-Scope.
request	Das Request-Objekt (Request-Scope).
requestScope	Eine *Map* von Variablen mit Request-Scope.
session	Das Session-Objekt (Session-Scope).
sessionScope	Eine *Map* von Variablen mit Session-Scope.
application	Das Application-Objekt (Application-Scope).
applicationScope	Eine *Map* von Variablen mit Application-Scope.

5 Implementationsdetails

Wie bereits erwähnt, basiert das gesamte Projekt auf Java Enterprise Edition. Nachfolgend wird sowohl die Architektur der Gesamtlösung als auch das entwickelte Datenbankschema genauer betrachtet.

5.1 Architektur

Im Grunde besteht die Gesamtlösung aus drei Komponenten (Abbildung 5.1):

- Dem *Book-Viewer* zur Anzeige der Werke
- Dem *Web-Book Server* der für die gesamte Verarbeitung der Daten zuständig ist
- Dem *Web-Book Administrationsportal* für das Anlegen und Bearbeiten von Werken

Für die Darstellung der Werke im Browser dient der Book-Viewer. Wie genau dieser funktioniert und aussieht, wurde bereits ausgiebig im Kapitel 3.2 beleuchtet. Der Book-Viewer holt sich die Anzeigedaten über ein Webservice von Web-Books Server. Dieser greift dann auf die Ressourcen wie Datenbank und auf die gespeicherten Files am Filesystem zu. Der Server implementiert somit den Großteil der Logik und kommuniziert über ein RESTEasy Webservice [21] mit den beiden anderen Komponenten. Das

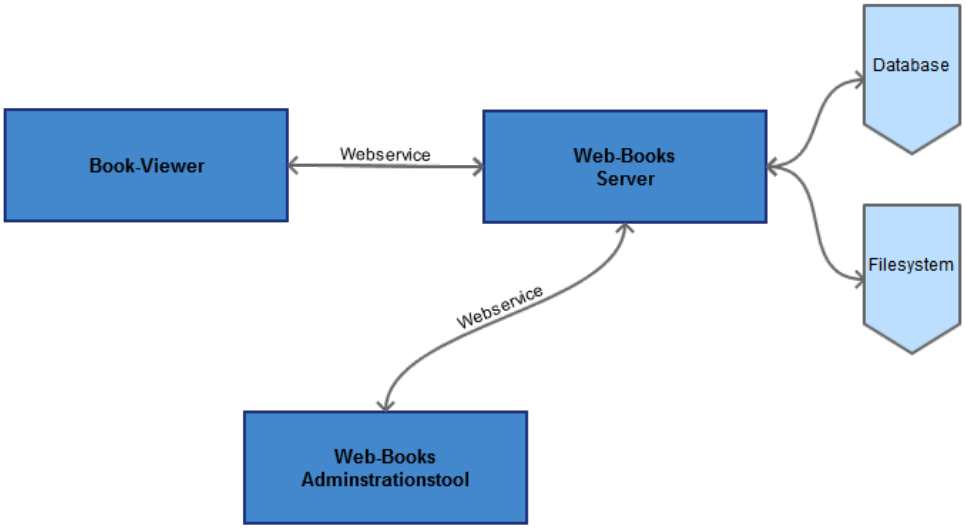

Abbildung 5.1: Architektur.

Web-Books Administrationsportal dient den Sachbearbeitern der Landesbibliothek Steiermark zum Anlegen und Bearbeiten von Werken. Eine genaue Beschreibung sowie eine Gegenüberstellung von Vor- und Nachteilen ist im Kapitel 3.1 zu finden.

5.2 Datenbankschema

Die wichtigsten Tabellen der Oracle Datenbank werden in den nachfolgenden Kapiteln beschrieben. Die gescannten Files werden nicht in der Datenbank selbst sondern direkt im Filesystem gespeichert und stehen über die Tabelle *WB_PAGE* und den darin gespeicherten Dateinamen in Relation.

5.2.1 WB_BOOK

Die Tabelle *WB_BOOK* (Abbildung 5.2) bildet ein Werk in der Datenbank ab. *ContentIndexed*, *MetadataIndexed* sind boolsche Werte die angeben ob der Inhalt bzw. die Metadaten mit Apache Lucene indiziert wurden. Weitere Spalten geben unter anderem an, ob das Werk öffentlich zugänglich sein soll oder ob es im Book-Viewer für den Benutzer downloadbar ist. Jedes Werk ist außerdem mit einem *Scope* und einer *Source* versehen (Genaue Beschreibung in den Kapiteln 5.2.4 und 5.2.3). Weitere Informationen zu einem Werk werden in der Tabelle *WB_METADATA* (Kapitel 5.2.5) gespeichert.

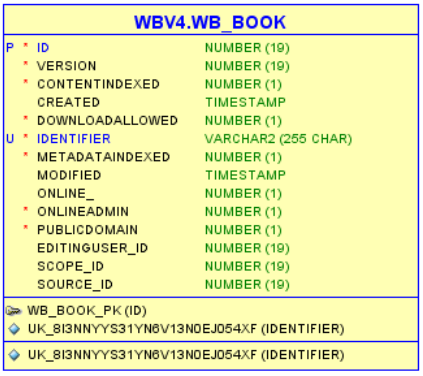

Abbildung 5.2: Tabelle WB_BOOK.

5.2.2 WB_USER_BOOK_OPTIONS

Die Tabelle *WB_USER_BOOK_OPTIONS* (Abbildung 5.3) erweitert die Tabelle *WB_BOOK* und speichert die Dateisystem-Pfade für das Werk. Die wichtigsten Pfade sind:

- Pfad zu den Original-Daten
- Pfad zum Coverbild
- Pfad zu den OCR-Daten

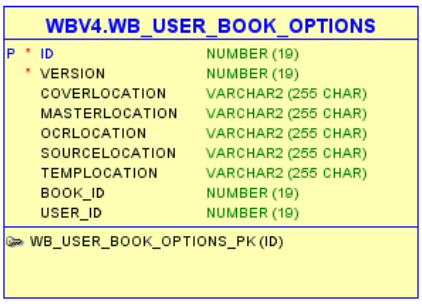

Abbildung 5.3: Tabelle WB_USER_BOOK_OPTIONS.

5.2.3 WB_SOURCE

Werke können mit einer Quelle versehen werden. Die Tabelle *WB_SOURCE* (Abbildung 5.4) beinhaltet alle möglichen Quellen. Man kann zum Beispiel definieren ob ein Werk der *Landesbibliothek Steiermark* oder *Universitätsbibliothek der TU Graz* zugehörig ist.

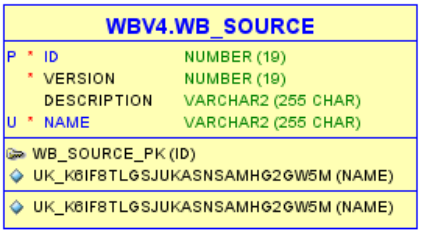

Abbildung 5.4: Tabelle WB_SOURCE.

5.2.4 WB_SCOPE

Um ein Werk einen bestimmten Gültigkeitsbereich zuzuweisen kann die Tabelle *WB_SCOPE* (Abbildung 5.5) herangezogen werden. Dadurch besteht die Möglichkeit im Book-Viewer nur die Werke die einem definierten Scope zugeordnet sind, dem Benutzer in der Online Bibliothek anzuzeigen.

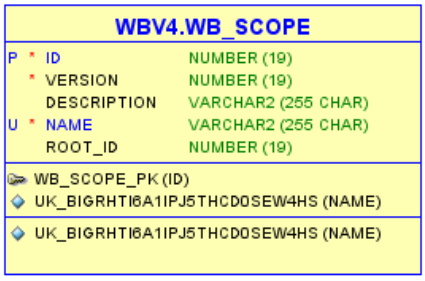

Abbildung 5.5: Tabelle WB_SCOPE.

Die Zuordnung eins Werkes zu einem Scope ist im bereitgestellten Prototyp des Administrationstools nicht enthalten.

5.2.5 WB_METADATA

Die eigentlichen Eigenschaften die ein Buch beschreiben werden in der Tabelle *WB_METADATA* (Abbildung 5.6) gehalten. Die Metadaten stehen, sowie viele andere Tabellen, über das Feld *BOOK_ID* in Relation mit der Tabelle *WB_BOOK*. *WB_METADATA* enthält folgende Felder:

- Titel
- Kurztitel
- Unteritel
- Autor (Referenz zu Tabelle *WB_AUTHOR* (Kapitel 5.2.6))
- Verlag (Referenz zu Tabelle *WB_PUBLISHER* (Kapitel 5.2.7))
- Höhe und Breite des Werkes
- Band
- Erscheinungsort des Werkes
- Weitere Informationen
- Keywords

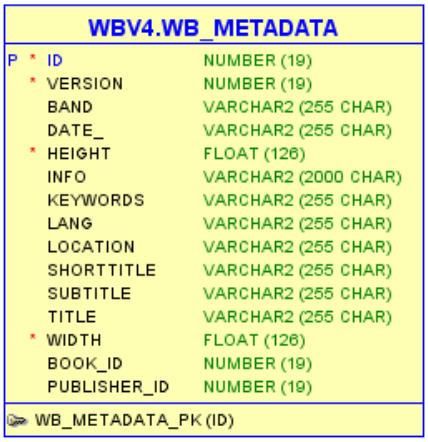

Abbildung 5.6: Tabelle WB_METADATA.

Verwendet man die XML-Import Funktion um Daten aus dem System *DABIS* zu importieren wird auch in diese Tabelle geschrieben.

5.2.6 WB_AUTHOR

Beim Speichern des Autors hätte es auch die Möglichkeit gegeben ihn direkt den Metadaten zuzuordnen. Um jedoch zu verhindern das der gleiche Autor mehrfach in der Datenbank gespeichert wird (Redundanz), gibt es die Zuordnung über die Tabelle *WB_AUTHOR* (Abbildung 5.7). Weiters soll es über eine Zwischentabelle möglich sein einem Werk mehrere Autoren zuzuordnen.

5.2.7 WB_PUBLISHER

Für die Tabelle *WB_PUBLISHER* (Abbildung 5.8) gilt das selbe Prinzip wie für die Tabelle *WB_AUTHOR*. Man will Redundanzen vermeiden.

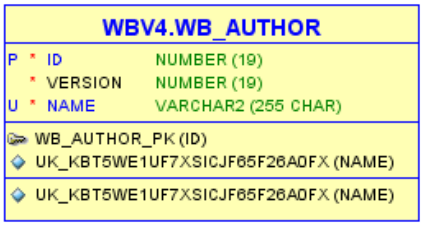

Abbildung 5.7: Tabelle WB_AUTHOR.

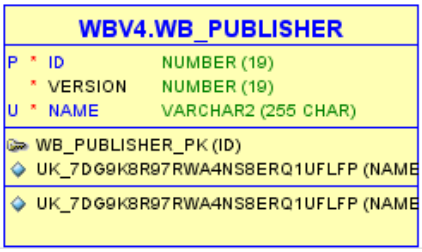

Abbildung 5.8: Tabelle WB_PUBLISHER.

5.2.8 WB_METADATA_TO_AUTHOR

Um einem Werk einen oder mehrere Autoren zuzuordnen gibt es die Tabelle *WB_METADATA_TO_AUTHOR* (Abbildung 5.8).

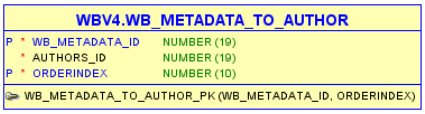

Abbildung 5.9: Tabelle WB_METADATA_TO_AUTHOR.

5.2.9 WB_CATEGORY

Ein Werk kann einer oder mehreren Kategorien zugeordnet werden. In der Tabelle *WB_CATEGORY* (Abbildung 5.10) können diese Kategorien

definiert werden. Eine Zuordnung eines Werkes erfolgt über die Tabelle *WB_CATEGORY_MEMBER*.

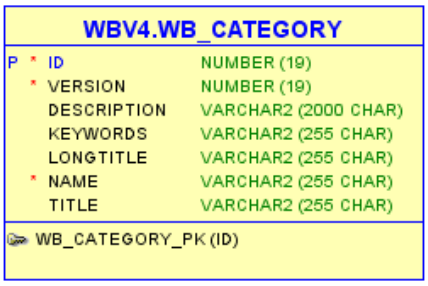

Abbildung 5.10: Tabelle WB_CATEGORY.

5.2.10 WB_CATEGORY_MEMBER

Die Tabelle *WB_CATEGORY_MEMBER* (Abbildung 5.11) hält alle Zuordnungen von Werken zu Kategorien.

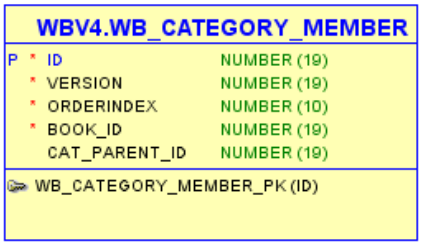

Abbildung 5.11: Tabelle WB_CATEGORY_MEMBER.

5.2.11 WB_CATEGORY_CHILDREN

Kategorien können hierarchisch angeordnet werden. Dies bedeutet das eine bestimme Kategorie beliebig viele andere Kategorien beinhalten kann. Dies

wird mithilfe der Tabelle *WB_CATEGORY_CHILDREN* (Abbildung 5.12) in der Datenbank abgebildet. Über das Feld *CAT_PARENT_ID* ist diese hierarchische Zuordnung möglich.

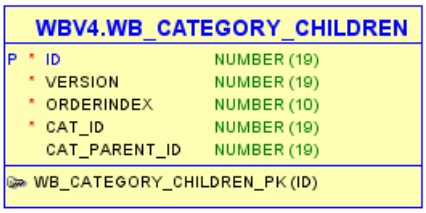

Abbildung 5.12: Tabelle WB_CHILDREN_MEMBER.

5.2.12 WB_PAGE

Eine einzelne Seite in einem Werk kann mit bestimmten Eigenschaften behaftet sein. Diese Eigenschaften werden in der Tabelle *WB_PAGE* (Abbildung 5.13) gespeichert.

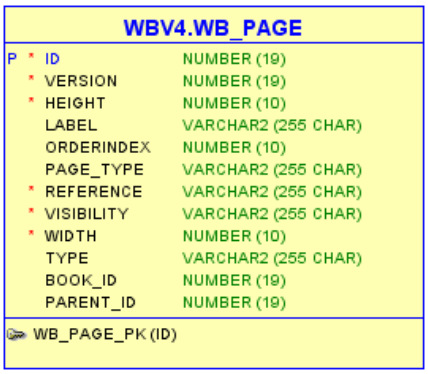

Abbildung 5.13: Tabelle WB_PAGE.

Höhe und Breite werden durch die Felder *HEIGHT* und *WIDTH* spezifiziert und in Pixel angegeben. Eine Seite kann auch ein *LABEL* haben. Dieses *LABEL* kommt einem Seitentitel gleich und kann zum Beispiel *"Einband vorne"*,

"Inhaltsverzeichnis" oder *"Seite 45"* lauten. Diese Seitentitel werden auch im Book-Viewer dem Benutzer angezeigt. In der Tabelle wird auch die Referenz auf den Deitnamen im Filesystem gespeichert (Spalte *REFERENCE*). Auch die Sichtbarkeit einer Seite kann spezifiziert werden. Dies ermöglicht es, einzelne Seiten dem Benutzer im Book-Viewer nicht anzuzeigen. Durch die Spalte *ORDERINDEX* wird die Sortierung der Seiten im Book-Viewer angegeben.

5.2.13 WB_PAGE_META

Nicht nur ein gesamtes Werk sondern auch eine einzelne Seite kann mit Metadaten behaftet sein die in der Tabelle *WB_PAGE_META* (Abbildung 5.14) gespeichert werden.

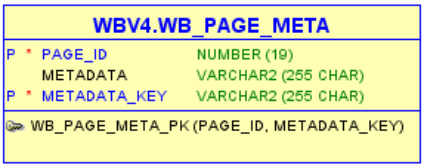

Abbildung 5.14: Tabelle WB_PAGE_META.

Durch die Angabe von *key* und *value* Werten können beliebig viele Metadaten für eine Seite spezifiziert werden. Hintergrund dieser Modellierung ist die Tatsache, dass viele Werke aus handgezeichneten Bildern bestehen die durch Zusatztext dem Benutzer im Book-Viewer besser nähergebracht werden sollen.

5.2.14 WB_LAYER

Die Tabelle *WB_LAYER* (Abbildung 5.15) ordnet einem Werk die verfügbaren Layer zu. Ein Werk kann sowohl als PDF (*print*) oder in einer gescannten Ver-

sion als Grafik (*facsimile)* zur Verfügung stehen. Zusätzlich zu der gescannten Version kann auch noch ein OCR Layer verfügbar sein.

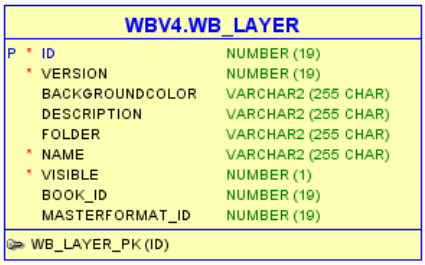

Abbildung 5.15: Tabelle WB_LAYER.

5.2.15 WB_FORMAT

WB_FORMAT (Abbildung 5.16) definiert alle möglichen Dateiformate für die Quellmedien. Diese Liste ist übersichtlich und beinhaltet zur Zeit neben verbreiteten Bildformaten (*JPG, PNG, TIF*) auch das Format *PDF*.

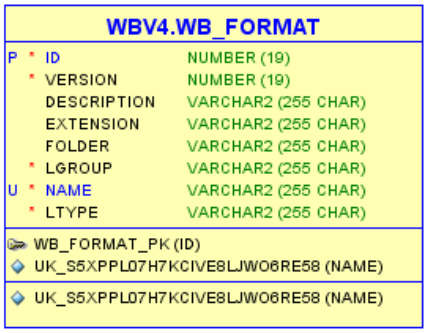

Abbildung 5.16: Tabelle WB_FORMAT.

5.2.16 WB_LAYER_TO_FORMAT

Die Tabelle *WB_LAYER_TO_FORMAT* (Abbildung 5.17) spezifiziert die Zuordnung von einem Layer zu einem Format. Als Beispiel und für das einfachere Verständnis:

- Layer *facsimile* des Werkes *Faust* enthält das Format *JPG*
- Layer *print* des Werkes *Erlkönig* enthält das Format *JPG*

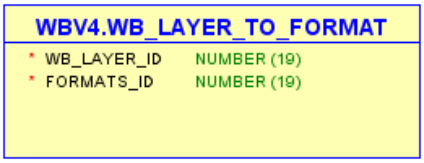

Abbildung 5.17: Tabelle WB_LAYER_TO_FORMAT.

5.2.17 WB_PROGRESS

Die Konvertierungs- und Uploadprozesse, die beim Generieren von Daten für den Book-Viewer notwendig sind, werden von Threads im Hintergrund erledigt. Der aktuelle Prozessstatus wird in die Tabelle *WB_PROGRESS* (Abbildung 5.18) gespeichert. Im Administrationsportal können alle laufenden Konvertierungs- und Uploadprozesse angezeigt werden.

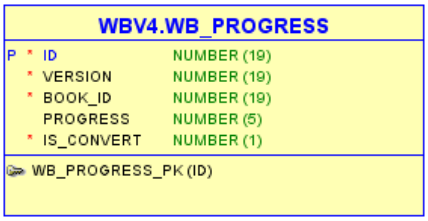

Abbildung 5.18: Tabelle WB_PROGRESS.

Endet ein Konvertierungs- oder Uploadthread wird der zugehörige Eintrag in der Tabelle wieder gelöscht. Im Fall das kein Thread mehr läuft, ist diese Tabelle leer.

6 Ausblick

Die Landesbibliothek Steiermark hat mit dem Prototypen für das webbasierte Administrationsportal in Kombination mit dem Book-Viewer nun ein gut funktionierendes Paket, um ihre digitale Bibliothek im Web zu präsentieren. Selbstverständlich gibt is bei dem Prototypen noch einiges an Verbesserungs- und Erweiterungspotential, jedoch bietet dieser Prototyp einen soliden Grundstock und eine gute Basis um weitere Entwicklungen voranzutreiben. Aktuell läuft der Prototyp auf einem Testsystem der IT-Abteilung der steirischen Landesregierung und wurde mit einigen Testdaten befüllt. Die wichtigste Erweiterungsmöglichkeit ist vor allem die Erhöhung der Kompatibilität mit internationalen gültigen Standards.

- OAI-PMH Schnittstelle für erhöhte internationale Sichtbarkeit von Werken
- METS Schnittstelle für die Anbindung weiterer Book-Viewer

Die Entscheidung liegt nun bei der Landesbibliothek Steiermark, die festlegt, ob auf die neue Webapplikation die im Zuge dieser Arbeit entwickelt wurde, oder auf eine bereits vorhandene (sowohl Open-Source als auch proprietär) aufgebaut wird. Hierbei sollte diese Arbeit bei der Entscheidungsfindung durch den Vergleich der Systeme behilflich sein.

Fällt die Entscheidung auf den neuen Prototypen wäre man außerdem in der Lage für viele weitere Bibliotheken das Hosting von digitalen Werken zu übernehmen und wäre damit einem breiten Anwendungsspektrum

konfrontiert. Verknüpft man die digitale Bibliothek auch noch mit einem Shop-System bei dem Kunden einzelne Werke durch Kauf erwerben können, blüht diesem System eine aussichtsreiche Zukunft im Umfeld der Landesbibliothek Steiermark.

7 Resümee

Ein durchwegs spannendes Projekt welches aufzeigt wieviel Potential für Bibliotheken vorhanden ist. Besonders Werke in schlechtem Zustand die sonst möglicherweise in Vergessenheit geraten und in den Archiven der Bibliothek verstauben würden, können mithilfe einer Präsentation im Internet wieder in den Vordergrund gerückt werden. Dieses Projekt hat auch gelehrt, dass viele Bibliothek Ressourcen in Form von Geld in die Hand nehmen müssen, um den aktuellen Kundenwünschen und -anforderungen gerecht zu werden. Der hart umkämpfte Markt und das enorme Potential wird wohl viele Bibliotheken dazu bringen ihren Bestand digital anzubieten.

Dieses Potential wird auch dazu führen, dass weitere kommerzielle Software auf den Markt kommen wird, um potentielle Kundschaft anzuwerben. Besonders interessant ist auch, dass auch die Open-Source Szene mit dem Projekt GOOBI Interesse an diesem Markt hat. Besonders kleinere Bibliotheken die weder eigene Server noch Personal für die Wartung eines komplexen Systems wie GOOBI haben, werden wohl eher auf einen Softwareanbieter mit Supportmöglichkeiten zurückgreifen.

Appendix

Literatur

[1] *Apache MyFaces*. besucht am 16.11.2015. URL: https://myfaces.apache.org/ (siehe S. 51).

[2] *Apache MyFaces Extension CDI (CODI)*. besucht am 14.05.2015. URL: http://myfaces.apache.org/extensions/cdi/ (siehe S. 20).

[3] *Apache MyFaces Extensions Validator*. besucht am 11.05.2015. URL: http://myfaces.apache.org/extensions/validator/ (siehe S. 20).

[4] *Apache Solr*. besucht am 01.04.2015. URL: http://lucene.apache.org/solr/ (siehe S. 35).

[5] *Definition des MARCXML Standard*. besucht am 10.11.2014. URL: https://www.loc.gov/standards/marcxml/ (siehe S. 17).

[6] *DFG-Viewer: Ueber das Projekt*. besucht am 21.08.2015. URL: http://dfg-viewer.de/ueber-das-projekt/ (siehe S. 25, 37, 38).

[7] *Dokumentation zu Java Enterprise Edition 6*. besucht am 07.12.2014. URL: http://www.oracle.com/technetwork/java/javaee/tech/javaee6technologies-1955512.html (siehe S. 19).

[8] *Dublin Core*. besucht am 23.01.2015. URL: http://dublincore.org/ (siehe S. 17).

[9] *Enterprise Java Beans*. besucht am 04.12.2015. URL: http://www.oracle.com/technetwork/java/javaee/ejb/ (siehe S. 20).

[10] *Europeana*. besucht am 01.12.2015. URL: http://www.europeana.eu/ (siehe S. 17).

[11] David Geary und Cay S. Horstmann. *Core JavaServer Faces*. Prentice Hall, 2004. ISBN: 0131463055 (siehe S. 51, 62–64, 66, 67, 69).

[12] *GOOBI Presentation*. besucht am 13.11.2015. URL: http://www.goobi.org/software/goobipresentation/ (siehe S. 25, 33).

[13] *GOOBI Production*. besucht am 13.11.2015. URL: http://www.goobi.org/software/goobiproduction/ (siehe S. 25, 26).

[14] *Hibernate Object-Relational Mapping*. besucht am 24.10.2014. URL: http://hibernate.org/orm/ (siehe S. 20).

[15] *Institut fuer Informationssysteme und Computermedien (TU Graz)*. besucht am 04.12.2015. IICM. URL: http://www.iicm.tugraz.at/ (siehe S. 43).

[16] *Intranda Viewer*. besucht ma 10.11.2015. URL: www.intranda.com/digiverso/intranda-viewer (siehe S. 25, 27, 28, 35, 36).

[17] *Java Naming and Directory Interface*. besucht am 19.09.2015. URL: http://www.oracle.com/technetwork/java/jndi/ (siehe S. 20).

[18] *Java Persistence API*. besucht am 19.08.2015. URL: http://www.oracle.com/technetwork/java/javaee/tech/persistence-jsp-140049.html (siehe S. 20).

[19] *Java Transaction API*. besucht am 08.08.2015. URL: http://www.oracle.com/technetwork/java/javaee/jta (siehe S. 20).

[20] *JBoss Enterprise Application Platform*. besucht am 17.03.2015. URL: www.jboss.org/products/eap/ (siehe S. 19).

[21] *JBoss RESTEasy*. besucht am 07.07.2015. URL: http://resteasy.jboss.org/ (siehe S. 75).

[22] *JSF 2.1*. besucht am 30.06.2015. URL: https://javaserverfaces.java.net/2.1/ (siehe S. 20).

[23] *Katalog der Landesbibliothek Steiermark*. besucht am 02.11.2015. URL: http://katalog.landesbibliothek.steiermark.at/ (siehe S. 16).

[24] *Library of Congress (LOC)*. besucht am 01.03.2015. URL: http://www.loc.gov/ (siehe S. 17).

[25] Jeff Linwood und Dave Minter. *Beginning Hibernate (Expert's Voice in Java Technology)*. Apress, 2010. ISBN: 1430228504 (siehe S. 49).

[26] *METS Standard*. besucht am 20.03.2015. URL: http://www.loc.gov/standards/mets/ (siehe S. 17).

[27] *Mojarra JSF*. besucht am 14.10.2015. URL: https://javaserverfaces.java.net/ (siehe S. 51).

[28] Bernd Mueller. *JavaServer Faces*. Hanser Fachbuchverlag, 2010. ISBN: 3446419926 (siehe S. 51–57, 59, 60, 64–67, 70, 71).

[29] *OAI-PMH Schnittstelle*. besucht am 21.11.2015. URL: https://www.openarchives.org/pmh/ (siehe S. 17).

[30] *Oracle DBMS*. besucht am 02.12.2014. URL: http://www.oracle.com (siehe S. 20).

[31] *Oracle Technical Documentation Java Enterprise Edition (Java EE)*. besucht am 12.11.2015. URL: https://docs.oracle.com/javaee/6/ (siehe S. 48).

[32] *Primefaces*. besucht am 12.08.2015. URL: http://www.primefaces.org/ (siehe S. 20, 68).

[33] *Semantics Visual Library .NET*. besucht am 20.11.2015. URL: http://www.visuallibrary.net/ (siehe S. 29).

[34] *STERZ-Portal der steirischen Landesverwaltung*. besucht am 04.03.2015. URL: https://sterz.stmk.gv.at/ (siehe S. 22, 42).

[35] *TYPO 3 Content Management System*. besucht am 17.11.2015. URL: https://typo3.org/ (siehe S. 33).

[36] *Visual Library*. besucht am 12.11.2015. URL: http://www.semantics.de/produkte/visual_library/ (siehe S. 25, 29, 31, 32).

[37] *vufind*. besucht am 25.11.2015. URL: http://vufind-org.github.io/vufind/ (siehe S. 16).

[38] *Walter Nagel*. besucht am 17.11.2015. URL: www.walternagel.de (siehe S. 28).

[39] *Web-Books des Austria-Forum*. besucht am 12.10.2015. URL: http://austria-forum.org/web-books/ (siehe S. 19, 43).

[40] *Zeutschel*. besucht am 15.11.2015. URL: https://www.zeutschel.de/ (siehe S. 27).

[41] *Zoomify*. besucht am 23.11.2015. URL: http://www.zoomify.com/ (siehe S. 32).