Diplomarbeit

Entwurf und Implementierung der webbasierten Anwendungssoftware CoRS-O-Mat

Elvir Hasanović
Matrikelnummer: 3678790
elvir@hasanovic.de

Betreuung durch Prof. Dr. Lutz Prechelt

Berlin, 29.02.2016

Zusammenfassung

Die Anwendungssoftware CoRS-O-Mat wurde entwickelt, um das geisteswissenschaftliche Forschungsprojekt CoRS zu digitalisieren und um einen Relaunch des Projekts initiieren zu können. Zudem soll CoRS-O-Mat auch anderen geisteswissenschaftlichen Forschungsprojekten mit ähnlichen Anforderungen zur Verfügung stehen. Mit geringen Änderungen ist es möglich, CoRS-O-Mat an die Anforderungen anderer Projekte anzupassen. Das wurde erreicht durch eine komponentenbasierte Architektur, in der die Komponenten über die REST-Schnittstelle lose miteinander gekoppelt sind. Zudem sind die Komponenten isoliert, d. h. keine Komponente kennt die internen Mechanismen einer anderen Komponente. Zum Datenaustausch verwenden die Komponenten die Messages-Objekte der Komponente Messages. Ein weiteres Architekturmerkmal ist die datenbankunabhängige Implementierung der Komponente Storage mit der automatischen Generierung der Entitys durch JPA.

CoRS-O-Mat wurde unter Berücksichtigung der aktuellen Java EE Standards und unter Nutzung der verfügbaren Referenzimplementierungen der offiziellen Spezifikationen entwickelt. Dadurch ist sichergestellt, dass die Anwendung nur gegen offizielle Standards programmiert und keine proprietären Interfaces implementiert wurden, damit sie mit allen anderen standardkonformen Java EE Architekturkomponenten interoperabel ist. Durch die Orientierung an modernen Entwurfsmustern, wie z. B. Responsive Web Design, Progressive Enhancement, ECB und MVC und der browserbasierten Architektur erreicht CoRS-O-Mat ein Maximum an Plattform- und Geräteunabhängigkeit.

Eidesstattliche Erklärung

Ich versichere hiermit an Eides Statt, dass diese Arbeit von niemand anderem als meiner Person verfasst worden ist. Alle verwendeten Hilfsmittel wie Berichte, Bücher, Internetseiten oder ähnliches sind im Literaturverzeichnis angegeben. Zitate aus fremden Arbeiten sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungskommission vorgelegt und auch nicht veröffentlicht.

Berlin, 29.02.2016

Elvir Hasanović

Inhaltsverzeichnis

1. Einleitung 1
1.1. Aufgabenstellung 1
1.2. Gliederung dieser Arbeit 2
2. Sprachwissenschaftlicher Hintergrund 3
2.1. Projekt CoRS: Ziele, Methoden, Arbeitsprozesse 3
2.2. Bewertung der Arbeitsprozesse 7
2.3. Anforderungen an eine Softwarelösung 7
2.4. Existierende Anwendungssoftwares 8
3. Informationstechnischer Hintergrund 11
3.1. Freie Software 11
3.2. Offene Standards 12
3.3. Java Enterprise Edition (Java EE) 12
3.3.1. Java EE Laufzeitumgebung 13
3.3.2. Java EE Container 14
3.3.3. Java Database Connectivity (JDBC) 14
3.3.4. Java Persistence API (JPA) 14
3.3.5. Enterprise Java Beans (EJB) 15
3.3.6. Context Dependency Injection (CDI) 15
3.3.7. Java Server Faces (JSF) 16
3.3.8. Unified Expression Language (EL) 18
3.3.9. Java Transaction API (JTA) 18
3.4. Apache Maven 18
3.5. Unified Modeling Language (UML) 20
3.6. Scrum 20
4. CoRS-O-Mat Softwareentwurf 22
4.1. Anforderungen 22
4.2. Wiederverwendung durch andere Projekte 23
4.3. Softwarearchitektur 24
4.3.1. Softwarekomponenten 26
4.3.2. Weitere Architekturmerkmale 28
4.4. Datenbankentwurf 28
4.4.1. Datenbankabstraktion 28
4.4.2. Datenbankmodell 29
4.4.3. Datenbankschlüssel 30
4.5. Sicherheit 33
4.5.1. Zugriffskontrolle 34
4.6. Das Zusammenspiel der verwendeten Frameworks 38

4.6.1. JSF und jQuery mobile 38
4.6.2. ManagedBeans, CDI Beans und EJBs 41
4.6.3. RESTful JSF mit der POST-redirect-GET-Methode 41
5. CoRS-O-Mat Implementierung 44
5.1. Software und Entwicklungstools 44
5.1.1. Integrierte Entwicklungsumgebung (IDE) und Projektsetup 44
5.1.2. Kontinuierliche Lieferung 45
5.1.3. Projektmanagement 46
5.1.4. Projektdurchführung mit Scrum 46
5.1.5. Datenbank 48
5.2. Implementierung 48
5.2.1. Vermeidung von Coderedundanz 48
5.2.2. UUID Generator 49
5.2.3. JPA-Entitys 49
5.2.4. Entity PersistenceBean 52
5.2.5. Entity PersistenceControl 52
5.2.6. RESTful Entity TransferBean 55
5.2.7. Entity TransferControl 58
5.2.8. Objektrelationale Konverter 60
5.2.9. Datenimport 61
5.2.10. Sätze zerlegen 62
6. Diskussion: Moderne Software und Zugänglichkeit 63
7. Fazit 65
7.1. Zusammenfassung 65
7.2. Ausblick 66
7.2.1. Generative Grammatik 66
7.2.2. Codegenerierung aus UML 66
7.2.3. Dokumentation 66
A. CoRS-O-Mat Domänenmodell 68
Literaturverzeichnis 69

Abbildungsverzeichnis

4.1. Die Komponenten der CoRS-O-Mat Softwarearchitektur 25
4.2. Auszug aus dem Domänenmodell 30
4.3. Authentifizierung mit JSON Web Token (JWT) 36
5.1. Scrum User Stories im Redmine Wiki 46
5.2. Scrum Product-Backlog in Redmine 47
5.3. Scrum-Sprint in Redmine 47
5.4. Auszug aus der CoRS Datentabelle 62
A.1. CoRS-O-Mat Domänenmodell als UML Diagramm 68

Tabellenverzeichnis

2.1. Wortstellungsmuster SVO und VSO 6
2.2. Funktionale Kategorie T 6
4.1. Aufbau einer UUID nach RFC 4122 32

Kapitel 1. Einleitung

In dieser Diplomarbeit wird der Entwurf und die Implementierung der webbasierten Anwendungssoftware CoRS-O-Mat beschrieben, die im Rahmen des sprachwissenschaftlichen Forschungsprojekts "Components of Romance Syntax (CoRS)"[1] zum Zweck der Erfassung, Annotation und Auswertung der durch das Projekt gesammelten linguistischen Forschungsdaten verwendet werden soll.

1.1. Aufgabenstellung

Die webbasierte Anwendungssoftware CoRS-O-Mat soll die Wiederaufnahme des Projekts CoRS ermöglichen und die Projektmitarbeiter besser als bisher bei der linguistischen Forschungsarbeit unterstützen. Hierzu soll insbesondere die bestehende Datenverwaltung ersetzt werden. Dieses Vorhaben kann dem Forschungsgegenstand der Digitalen Geisteswissenschaften (Digital Humanities)[2] zugeordnet werden, der zur Zeit in großem Umfang von der Bundesrepublik Deutschland und der Europäischen Union gefördert wird[3].

Ziel der Implementierung von CoRS-O-Mat ist es, anderen Fachbereichen und Forschungsprojekten mit ähnlichen Anforderungen zu ermöglichen, das System einzusetzen. Um die Wiederverwendbarkeit zu gewährleisten, wird die Anwendungssoftware nach offenen Standards und unter Nutzung von freier Software implementiert, sowie unter einen freien Softwarelizenz veröffentlicht[4]. Auf Implementierungsebene wird Wiederverwendbarkeit und Erweiterbarkeit von CoRS-O-Mat durch eine modular gekapselte Komponentenarchitektur und einer Abstraktion der Datenschicht umgesetzt. Weiterhin soll sichergestellt werden, dass der Aufwand für die Anpassung an die spezifischen Anforderungen anderer Forschungsprojekte möglichst gering ausfällt. Das wird erreicht durch die Orientierung an modernen Software-Entwurfsmustern sowie der Anfertigung einer verständlichen Dokumentation. Die Komponenten werden derart entworfen und implementiert, so dass Teile der Software automatisch generiert werden können und sich dadurch der manuelle Anpassungsaufwand auf die Gestalt und Struktur der Benutzeroberfläche und das Datenmodell beschränkt. Erreicht wird das durch Entwurf und Verwendung von abstrakten und typisierten Klassen.

---

[1] Components of Romance Syntax (Bausteine romanischer Syntax) - Übereinzelsprachlich gültige Formalisierung grundlegender syntaktischer Eigenschaften romanischer Sprachen im Rahmen des minimalistischen Programms. Projektleitung durch Prof. Dr. Eva-Maria Remberger, Universität Wien und Prof. Dr. Guido Mensching, Universität Göttingen

[2] Was sind die digitalen Geisteswissenschaften [http://dig-hum.de/sites/dig-hum.de/files/cceh_broschuereweb.pdf]

[3] Förderung der digitalen Geisteswissenschaften [http://www.dig-hum.de/forschung/foerderung/]

[4] Die Software CoRS-O-Mat und deren Dokumentationen sollen unter die freie Lizenz European Union Public Licence (EUPL) [https://joinup.ec.europa.eu/software/page/eupl] gestellt und in einem öffentlichen Repository bei SourceForge [http://sourceforge.net/projects/corsomat/] publiziert werden.

Als Softwareentwicklungsstrategie wird Inkrementelle Entwicklung in Kombination mit der Entwicklungsmethode Scrum verwendet. Zur Qualitätssicherung werden die Methoden Akzeptanztest getriebene Entwicklung und Kontinuierliche Lieferung eingesetzt.

Ein erster Entwurf sah eine Desktop-Applikation auf Basis von Apache OpenOffice Base vor. Es wurde schnell klar, dass das Konzept einer zentralen Datenspeicherung für ein kollaboratives Forschungsprojekt nicht geeignet ist, da sich verteiltes Arbeiten auf lokalen Daten schwer realisieren lässt. Dieser Entwurf wurde zu Gunsten einer browserbasierten Webapplikation verworfen. Dadurch wird zudem die Plattformunabhängigkeit vergrößert werden und die Anwendungssoftware kann so auch auf Tablets und Smartphones aller Hersteller benutzt werden. Der Entwurf sieht eine Client-Server-Infrastruktur vor, die also auf nahezu allen internetfähigen Geräten benutzt werden kann und welche durch klar definierte, standardisierte Schnittstellen mit geringem Aufwand erweitert werden kann.

1.2. Gliederung dieser Arbeit

Kapitel 2 stellt die Ziele, Methoden und Arbeitsprozesse des Projekts CoRS vor. Anhand der Bewertung der Arbeitsprozesse in Abschnitt 2.1 werden im darauffolgenden Abschnitt die domänenspezifischen Anforderungen abgeleitet. Im Abschnitt 2.4 werden existierende Anwendungen bezüglich dieser Anforderungen evaluiert. In Kapitel 3 werden Fachbegriffe, Technologien und Methoden erläutert, die zum technischen Verständnis des Softwareentwurfs notwendig sind. Das darauf folgende Kapitel 4 beschreibt den geplanten Softwareentwurf. In Abschnitt 4.1 werden die Anforderungen anhand von User Stories nach Scrum dargestellt. Abschnitt 4.2 liefert Argumente für eine komponentenbasierte Softwarearchitektur, die in Abschnitt 4.3 erläutert wird. Es folgen der Datenbankentwurf und das Sicherheitskonzept in Abschnitt 4.4 und Abschnitt 4.5. In Abschnitt 4.6 werden die umgesetzten Maßnahmen beschrieben, die ein konfliktfreies Zusammenspiel der verwendeten Frameworks ermöglichen. Kapitel 5 beschreibt das Projektsetup und die tatsächliche Implementierung der Software. Überlegungen zur Zugänglichkeit werden in Kapitel 6 diskutiert. In Kapitel 7 wird eine Zusammenfassung der vorliegenden Arbeit formuliert und ein Ausblick auf zukünftige Entwicklungen der Software eröffnet.

Kapitel 2. Sprachwissenschaftlicher Hintergrund

Dieses Kapitel erläutert den sprachwissenschaftlichen Hintergrund, in dem die Software CoRS-O-Mat angesiedelt ist. Daraus wird ein Domänenmodell abgeleitet, aus dem die nötigen Softwarestrukturen erzeugt werden können, wie Datenbankschema, Dokumenten- und Entity-Klassen und Ansichtskomponenten, auf die in den folgenden Kapiteln näher eingegangen wird.

2.1. Projekt CoRS: Ziele, Methoden, Arbeitsprozesse

Im Folgenden werden die Ziele, Methoden und Arbeitsprozesse des Projekts CoRS beschrieben. Es sollen nur die Aspekte betrachtet werden, die für den Entwurf der Anwendungssoftware CoRS-O-Mat relevant sind. Die benötigten sprachwissenschaftlichen Hintergründe sollen dabei erläutert werden. Dieser Abschnitt ist als Frage-Antwort-Dialog zwischen Elvir Hasanović, dem Verfasser dieser Arbeit, und der CoRS-Projektmitarbeiterin Anja Weingart realisiert, die im weiteren Softwareentwurf die Scrum-Rolle des Product Owners[1] einnehmen wird.

Die offizielle Projektbeschreibung laut Webseite der DFG[2] lautet: "Das Projekt hat eine umfassende, formal homogene und moderne syntaktische Analyse der größeren romanischen Sprachen zum Ziel."

Elvir H.: "Welche romanischen Sprachen wurden untersucht?"

Anja W.: "Wir haben die Standardvarietäten des Spanischen, Italienischen, Rumänischen, Französischen und Europäischen Portugiesisch untersucht. Eingeplant waren auch Lateinamerikanische Varietäten des Spanischen und das Brasilianische Portugiesisch sowie das Katalanische, Sardische, Okzitanische, Rätoromanische und Frankoprovenzalische, aber aufgrund der Projektlaufzeit und der Anzahl an Mitarbeitern wurden diese wieder herausgenommen."

Elvir H.: "Was bedeutet in diesem Zusammenhang eine "umfassende, formal homogene und moderne syntaktische Analyse"?"

Anja W.: "Erstmal ist modern hier im Gegensatz zu traditionell, strukturalistisch gemeint. Zu den modernen, formalen Syntaxtheorien gehören LFG (Lexical Functional Grammar), HPSG (Head-Driven Phrase Structure Grammar) und generative Grammatik. Wir haben eine Analyse im Rahmen der generativen Grammatik (Minimalistisches Programm[3]) angestrebt, weil hierzu nur Untersuchungen zu Einzelphänomenen oder Einzelsprachen existieren. Umfassend bezieht sich auf die Sprachen und folgende syntaktische Phänomene: Wortstellung in Deklarativsätzen, Interaktion zwischen Wortstellung und Informationsstruktur, die linke Satzperipherie, die Posi-

[1] Vgl. Abschnitt 3.6, „Scrum“

[2] CoRS-Projektbeschreibung auf der Webseite der DFG GEPRIS [http://gepris.dfg.de/gepris/projekt/42308454]

[3] Vgl. [CHOMSKY.2000], [CHOMSKY.2001], [CHOMSKY.2004], [CHOMSKY.2008]

tion und Reihenfolge von Klitika, Diathese und Reflexivkonstruktionen sowie nicht-finite Konstruktionen."

Elvir H.: "Wir nehmt ihr die syntaktischen Phänomene auf?"

Anja W.: "Für jedes Phänomen, wie z. B. ein Wortstellungsmuster (Verb-Subjekt-Objekt) haben wir Beispielsätze in einer Word-Datentabelle gesammelt. Die Sätze stammen entweder aus der Literatur, aus Korpora oder aus Sprecherbefragungen, die wir selber durchgeführt haben. Der Aufbau der Tabelle wurde im Laufe der Zeit immer komplexer. Anfangs haben wir in jeder Zeile jeweils ein Wortstellungsmuster mit Beispielen in den verschiedenen Sprachen aufgenommen. Es hat sich herausgestellt, dass der Verbtyp und die Informationsstruktur für die Analyse relevant sind und hinzugenommen werden mussten. Nach Erfassung nur der Hälfte der Phänomene enthielt die Datensammlung bereits 4000 Einträge und die Tabelle wurde unübersichtlich."

Elvir H.: "Wer hat an dieser Tabelle gearbeitet und wie habt ihr das gemeinsame Arbeiten organisiert?"

Anja W.: "Wir haben zu zweit an den Daten gearbeitet und haben die Sprachen aufgeteilt. Ich habe Spanisch und Europäisches Portugiesisch bearbeitet, mein Kollege Rumänisch und Italienisch und wir beide haben am Französischen gearbeitet. Die Tabelle wurde im Zwei-Wochenrhythmus per E-Mail von Mitarbeiter zu Mitarbeiter weitergegeben. Das bedeutete doppelte Arbeit, da in der Zeit, in der die Tabelle nicht verfügbar war, neue Daten auf einem Extrazettel aufgenommen wurden und später übertragen werden mussten. Problematisch war auch, dass wir die Daten nicht einheitlich aufgenommen haben, weil wir uns nicht der Schwierigkeiten bewusst waren, die eine große, uneinheitlich aufgenommene Datensammlung mit sich bringen würde."

Elvir H.: "Was waren das für Schwierigkeiten?"

Anja W.: "Die Ergebnisse der Textsuche waren unvollständig und die Tabelle war für Dritte nicht mehr lesbar. Letztendlich war eine sinnvolle Datenauswertung auch nicht mehr möglich und für eine umfassende, vergleichende Analyse wären komplexe Abfragen nötig gewesen."

Elvir H.: "Warum fiel die Entscheidung zu Gunsten einer Word-Tabelle aus?"

Anja W.: "Der Projektplan sah keine Softwareentwicklung oder -beschaffung vor. Eine Eigenentwicklung hätte sicherlich zu lange gedauert und die eigentliche Forschungsarbeit zu lange aufgehalten. Kein Projektbeteiligter hatte über ausreichende Kenntnisse verfügt, um ein solches System aufzusetzen oder existierende Software anzupassen. Microsoft Word war aus technischer Sicht der kleinste gemeinsame Nenner."

Elvir H.: "Was müsste eine Software können, um die Projektmitarbeiter besser als bisher zu unterstützen?"

Anja W.: "Ich möchte meine Sätze aufnehmen und annotieren können. Schön wäre, wenn die Software dafür sorgt, dass Annotationsmerkmale vordefiniert sind, um Uneinheitlichkeiten zu vermeiden. Ich möchte, dass die Projektmitarbeiter gleichzeitig am Datenbestand arbeiten können. Praktisch wäre auch, dass andere Forscher Zugang zu den Daten bekommen könnten, um z. B. Akzeptabilitätsurteile abgeben zu können. Ich möchte komplexe Abfragen über die erfassten und annotierten Daten formulieren können. Ich möchte, dass die Software flexibel ist, so dass man jederzeit Änderungen am Datenmodell vornehmen kann. Im Idealfall hilft mir die Software, Hypothesen zu testen."

Elvir H.: "Wie sieht denn so ein vollständig annotierter Satz aus?"

Anja W.: "Zum Beispiel repräsentieren die Sätze "Comió Juan una manzana." und "#Comeu o João uma maçã." die Wortstellung VSO. Im Portugiesischen ist der Satz mit einer out-of-the-blue Lesart nicht akzeptabel, aber mit Fokus auf dem Subjekt schon. Im Spanischen ist der Satz mit beiden Lesarten in Ordnung. Es müssen also Akzeptabilitätsurteile und Funktion und Informationsstruktur von Konstituenten erfasst werden können. Bei der Akzeptabilität der SV/ VS Wortstellungen spielt auch der Verbtyp eine Rolle. Z. B. erlaubt nur das Portugiesische SV mit unakkusativischen Verben und einer out-of-the-blue Lesart. Es müssen also auch Wortarten annotierbar sein. Problematisch stelle ich mir die Annotation von Wortformen wie z. B. "comprar-lo-ia" (dt.: "Ich würde es kaufen") vor. Das Objektpronomen ("lo") liegt zwischen Wortstamm ("comprar") und Flexionsendung ("ia"). Es müssen auch Kongruenzmerkmale, Kasus und semantische Merkmale wie Definitheit annotierbar sein."

Elvir H.: "Bei der Datenerfassung sind also Annotation auf mehreren Ebenen erforderlich?"

Anja W.: "Ja, auf Wort-, Phrasen- und Satzebene. Und ganz wichtig ist, dass wir auch ungrammatische und nicht-akzeptierte Sätze wie "#Comeu o João uma maçã." kennzeichnen können."

Elvir H.: "Was ist so eine Hypothese?"

Anja W.: "Ziel des Projekts ist es, zu erklären, warum eine Wortstellung in einer Sprache erlaubt ist, aber nicht in einer anderen. Innerhalb des Minimalistischen Programms wird Variation ins Lexikon verlagert. Es wird angenommen, dass syntaktische Strukturerzeugung für alle Sprachen gleich funktioniert und sich die Unterschiede aus den Lexikoneinträgen von Wörtern und funktionalen Elementen ergeben. Hypothesen werden über die Zusammensetzung von funktionalen Elementen formuliert. Zum Beispiel die Sätze in Tabelle 1 repräsentieren die Wortstellungsmuster [Subjekt-Verb-Ojekt] und [Verb-Subjekt-Objekt] mit einer out-of-the-blue Lesart[4]. Die Wortstellung SVO wird in allen fünf Sprachen mit dieser Lesart akzeptiert, VSO jedoch nur im

[4]Zeichenerklärung: ✓ = akzeptiert , * = nicht-akzeptiert, unabhängig von der Lesart, # nur mit dieser Lesart nicht akzeptiert

Spanischen und Rumänischen. Im Italienischen und Portugiesischen ist die VSO-Wortstellung nur möglich, wenn das Subjekt und/oder das Objekt fokussiert sind. Im Französischen ist diese Wortstellung unabhängig von der Informationsstruktur nie akzeptiert.

		SVO		**VSO**
a.	It.	Gianni mangia una mela.	a´.	#Mangió Gianni una/la mela.
b.	Rum.	Ion mănâncă un măr.	b´.	Mănâncă Ion un măr / mărul.
c.	Sp.	Juan come una manzana.	c´.	Comió Juan una/la manzana.
d.	EP	O João come uma maçã.	d´.	#Comeu o João uma /a maçã.
e.	Fr.	Jean mange une pomme.	e´.	*(Il) Mangea Jean une / la pomme.
		'J. isst einen Apfel'		*'Aß einen/den Apfel J.'*

Tabelle 2.1. Wortstellungsmuster SVO und VSO

Aus welchen Merkmalen müssen die funktionalen Elemente bestehen, damit

i. die Wortstellung SVO für alle Sprachen generiert wird,

ii. für das Französische die Wortstellungen VSO ausgeschlossen wird,

iii. einbezogen wird, dass im Italienischen und Portugiesischen VSO nur mit dieser Lesart nicht akzeptabel ist

iv. aber im Rumänischen und Spanischen mit dieser Lesart erlaubt ist?

Die parametrisierte funktionale Kategorie T wird mit der Ausprägung T1 nur für das Französische angenommen und in der Ausprägung T2 für die anderen Sprachen.

Fr.	T1 : [finit] [uφ]-[EPP]
It., Sp., Rum., EP	T2 : [finit] [uφ] [uAbout]-[EPP]

Tabelle 2.2. Funktionale Kategorie T

Die Kategorie T1 erzeugt immer eine Struktur mit präverbalem Element, unabhängig von der Informationsstruktur. (Das strukturerzeugende Merkmal EPP ist an rein grammatische Merk-

male [uφ] gekoppelt.) Bei der Kategorie T2 ist das EPP an ein informationsstrukturelles Merkmal gekoppelt. Die Verfügbarkeit eines Elements, das das [uAbout] Merkmal sättigen kann, ist wiederum für das Spanische/Rumänische und das Italienische/Portugiesische unterschiedlich[5].

2.2. Bewertung der Arbeitsprozesse

Zur Datenverwaltung wurde eine Datei im Dateiformat Microsoft Word angelegt. Das gewählte Dateiformat erlaubt kein gleichzeitiges Bearbeiten der Daten. Das stellt ein Problem dar, da das Projekt CoRS ein kollaboratives Forschungsprojekt ist, an dem mehrere Mitarbeiter an verschiedenen, internationalen Standorten verteilt arbeiten. Die ca. 4000 gesammelten Sprachdatensätze sind nicht effizient auswertbar und in ihrer Struktur nicht einheitlich aufgenommen worden. Aus diesen Gründen konnten nur Einzelphänomene auf der Basis eines Bruchteils der erhobenen Daten untersucht werden. Umfassendere Analysen scheiterten am gewählten Datenformat, das keine gezielten Abfragen zulässt. In den nächsten beiden Abschnitten werden die Anforderungen an eine Anwendungssoftware zur Überwindung genannter methodischer Defizite formuliert und existierende Anwendungssoftwares im Hinblick auf die Einsetzbarkeit für das CoRS-Projekt untersucht.

2.3. Anforderungen an eine Softwarelösung

CoRS-O-Mat soll verwendet werden, um die Sprachdaten einheitlich zu annotieren und zu strukturieren und zusammen mit den dazugehörigen bibliographischen Angaben zu verwalten. Zudem sollen komplexe und teilautomatisierte Auswertungen ermöglicht werden. Um die Anwendungssoftware zu realisieren, müssen als Erstes die fachlichen Anforderungen formuliert werden. Diese werden in den folgenden Phasen des Softwareentwurfs weiter verarbeitet und letztendlich in der Implementierungsphase umgesetzt.

Es ergeben sich die folgenden fachlichen Anforderungen an die Komponenten der Anwendungssoftware:

- Komplexe Abfragen in Bezug auf Strukturmuster und morpho-syntaktische, semantische und informationsstrukturelle Merkmale, sowie Akzeptabilitätsurteile sollen möglich sein. Hierfür müssen die Beispielsätze auf Wort-, Phrasen- und auf Konstituentenebene mit diesen Merkmalen und Positionsangaben annotiert werden können.

- Grammatikableitungen an Teilen oder am gesamten Datenbestand sollen überprüft werden können. Hierfür sollen Ansätze aus dem maschinellen Lernen implementiert werden.

- Erforderlich ist eine sichere, webbasierte Lösung, die auch Besuchern einen Zugang zu den Forschungsdaten ermöglicht.

[5] Vgl. [GIURGEA.REMBERGER.2012] und [MENSCHING.WEINGART.2016]

Der Entwurf und die Implementierung der Anwendungssoftware haben als langfristiges Ziel, dass das System einfach zu warten und zu erweitern sein wird. Jedoch ist das Erreichen dieses Ziels verbunden mit der Tatsache, dass mit dieser eher komplexen Lösung im Vergleich zu einem einfacheren Ansatz ein erheblich größerer Umsetzungsaufwand einhergeht. Im folgenden Abschnitt findet eine Betrachtung statt, warum für das Projekt CoRS nicht auf eine bereits existierende Lösung zurückgegriffen werden kann.

2.4. Existierende Anwendungssoftwares

Das Projekt CoRS kann nicht auf eine bestehende Softwarelösung zurückgreifen, da die auf dem Markt existierende Anwendungssoftwares für sprachwissenschaftliche Forschungsprojekte meist bereits eine der Grundanforderungen des Projekts nicht abdecken können. Die Untersuchungen von CoRS basieren auf einem dynamischen Korpus, d. h. der Datenbestand kann jederzeit verändert werden und das Datenmodell ist auch dynamisch und kann sich ändern. Die im Rahmen dieser Arbeit untersuchten Anwendungen nahmen alle einen geschlossenen Korpus als Voraussetzung an. Diese sehen nicht vor, dass auch ungrammatische Sätze aufgenommen werden können. Dabei bieten manche Anwendungen Teilfunktionen an, die durchaus im Rahmen von CoRS verwendbar wären und zum Teil auch gut umgesetzt wurden, wie z. B. die Annotierung von Sprachdaten auf mehreren frei definierbaren Ebenen. Da aber für diese Funktionen kein API existiert, können diese nicht losgelöst von der Hauptanwendung verwendet werden. Es sollen im Folgenden beispielhaft zwei Anwendungen genannt werden, die für das Projekt CoRS auf den ersten Blick in Frage kommen könnten. Zuerst wird die seit 2006 in der Entwicklung befindliche und vom Bundesministerium für Bildung und Forschung (BMBF[6]) geförderte freie Software TextGrid[7] vorgestellt, danach die von der Deutschen Forschungsgesellschaft (DFG[8]) geförderte Werkzeugsammlung corpus-tools.org[9].

TextGrid wurde entwickelt, um eine virtuelle Forschungsumgebung für Geistes- und Kulturwissenschaftler zu schaffen. Sie besteht aus den zwei Bestandteilen TextGrid Repository, einem Datenarchiv, und TextGrid Laboratory, der Clientsoftware. Mit TextGrid ist es möglich, Daten zu erfassen und Volltextsuchen über textbasierten Daten auszuführen und diese Daten über ein ausgereiftes Benutzer- und Rechtemanagement anderen Teilnehmern zur Verfügung zu stellen. Textgrid Laboratory enthält Funktionalitäten, die für das Projekt CoRS irrelevant sind, wie z. B. ein Text-Link-Bild-Editor, der es erlaubt, eingescannte Bildvorlagen mit Texten und Annotationen zu verknüpfen oder einer Wörterbuchsuche, die ein eingegebenes Schlüsselwort in verschiedenen Wörterbüchern zur Begriffsdefinition sucht. Die Clientsoftware ist ein auf der Eclipse IDE

[6]BMBF Webseite [https://www.bmbf.de]

[7]TextGrid - Virtuelle Forschungsumgebung für die Geisteswissenschaften [https://textgrid.de]

[8]Webseite der Deutschen Forschungsgesellschaft [http://www.dfg.de]

[9]corpus-tools.org [http://corpus-tools.org/home/]

basierender Editor, der für alle großen Plattformen erhältlich ist (Linux, MacOS und Windows) und auf dem jeweiligen Rechner installiert werden muss. Die Bedienung der Clientsoftware ist etwas umständlich und Bedarf einer größeren Einarbeitung, eine zugänglichere Anwendung wäre wünschenswert. Für kleinere Projekte ist diese Lösung zu schwergewichtig, eine leichtgewichtigere und flexiblere Lösung ist nicht vorgesehen. Die Nutzung der Software auf Rechnern, auf denen keine Software installiert werden kann oder darf, ist nicht möglich. Die erfassten Daten können mit TextGrid zwar geteilt und publiziert werden, jedoch ist das Erfassen und Ändern der Daten nur mit der Clientsoftware TextGrid Laboratory möglich. Bei CoRS sollen die Daten jedoch von mehreren Benutzern gleichzeitig erfasst, bearbeitet und gelöscht werden können.

Die Werkzeugsammlung corpus-tools.org besteht aus den vier Werkzeugen ANNIS, Atomic, Pepper und Salt. Atomic, ebenso eine auf Eclipse basierende Anwendung, dient der Erfassung und Annotierung eines Textbestands. Es ist vom Funktionsumfang für das Projekt CoRS besser geeignet als TextGrid, da es Annotationen an einem Textkorpus auf mehreren Ebenen ermöglicht, jedoch muss der Korpus abgeschlossen sein, wenn er zur Annotation importiert werden soll. Ein nachträgliches Einfügen, Löschen oder Ändern von Sätzen ist nicht vorgesehen. Diese Einschränkung ist nur dann sinnvoll, wenn der Korpus auch einen abgeschlossenen Text darstellt, wie es z. B. bei einem Buch der Fall ist. Das Projekt CoRS basiert aber auf ausgesuchten Datensätzen, die im Laufe der Zeit durchaus verändert werden sollen. Atomic ist umständlich in der Bedienung und nicht selbsterklärend. Bei einem initialen Test mit dem Betriebssystem Windows 7 fiel auf, dass die Anwendung instabil läuft und zu unerklärlichen Abstürzen mit kryptischen Fehlermeldungen neigt. Es war leider fast unmöglich, auf einem aktuellen Rechner mit ausreichenden Ressourcen einen Testkorpus mit rund 50 Sätzen und ca. 10000 zerlegten Tokens zu laden und samt Annotationen anzuzeigen, das Programm hat überwiegend nach minutenlanger Wartezeit seinen Dienst eingestellt und wurde letztendlich vom Betriebssystem geschlossen. Das Arbeiten am aktuellen Datenbestand von CoRS mit rund 4000 erfassten Datensätzen wäre unter diesen Umständen nicht möglich. ANNIS ist eine Anwendung für die Formulierung von Abfragen, die den Datenbestand nach definierbaren Kriterien durchsucht. Die Abfragen werden in einer eigenen Abfragesprache (AQL) formuliert, die an der Definition von Regular Expressions angelehnt ist und diese durch neue Schlüsselwörter und Operatoren erweitert. Das Formulieren einer solchen Abfrage bedingt eine Einarbeitung in AQL, ohne dieses spezielle Wissen ist es dem Benutzer nicht möglich, komplexere Abfragen zu formulieren. ANNIS ist eine browserbasierte Anwendung, die auf einem Server instaliert werden muss. Alternativ können auch öffentliche Installationen genutzt werden, wie z. B. die der Humboldt Universität zu Berlin[10]. Die Abfragesprache AQL ist mächtig und erlaubt komplexe Abfragen, die Visualisierung der Ergebnisse ist sehr gut und informativ. Allerdings lassen sich die Abfragen nur auf einem Korpus durchführen, eine übereinzelsprachliche Analyse der Daten scheint nicht vorgesehen zu sein. Die Anwendung

[10]Webseite der ANNIS-Installation der HU Berlin [https://korpling.german.hu-berlin.de/annis3/]

Salt dient zur Definition des Datenmodells und wird von Atomic implizit verwendet, Pepper ist eine Konvertierungsanwendung, die Daten zwischen linguistischen Standardformaten konvertieren kann, wie z. B. EXMARaLDA, Tiger XML, MMAX2 und weitere. Salt und Pepper bieten auch ein REST-API an, um von externen Anwendungen verwendet werden zu können, kommen jedoch für CoRS nicht in Frage. Zum Einen gibt es keinen Bedarf an einer nicht-standardisierten Meta-Sprache für ein Datenmodell, welches Salt liefern könnte und zum Anderen werden Daten weder in eines der linguistischen Standardformate importiert noch exportiert.

Um also eine existierende Anwendung für das Projekt CoRS verwenden zu können, müssen Abstriche gemacht werden, die in allen Fällen nicht hinnehmbar sind. Es lässt sich von den untersuchten Anwendungen kein leichtgewichtiges und performantes Werkzeug finden, was die Datenerfassung und -auswertung nach den für das Projekt CoRS wichtigen Gesichtspunkten ermöglicht.

Kapitel 3. Informationstechnischer Hintergrund

Im folgenden Kapitel werden die Grundlagen der Anwendungssoftware CoRS-O-Mat erläutert.

3.1. Freie Software

Unter freier Software im Sinne der Free Software Foundation (FSF) versteht man Programme, die dem Nutzer folgende vier Freiheiten gewähren[1]:

- "Die Freiheit, das Programm auszuführen wie man möchte, für jeden Zweck (Freiheit 0)."
- "Die Freiheit, die Funktionsweise des Programms zu untersuchen und eigenen Datenverarbeitungbedürfnissen anzupassen (Freiheit 1). Der Zugang zum Quellcode ist dafür Voraussetzung."
- "Die Freiheit, das Programm weiterzuverbreiten und damit seinen Mitmenschen zu helfen (Freiheit 2)."
- "Die Freiheit, das Programm zu verbessern und diese Verbesserungen der Öffentlichkeit freizugeben, damit die gesamte Gemeinschaft davon profitiert (Freiheit 3). Der Zugang zum Quellcode ist dafür Voraussetzung."

Werden eine oder mehrere dieser Freiheiten nicht gewährt, so spricht die FSF von unfreier oder proprietärer Software. Die FSF benutzt den Begriff der freien Software als ein politisches und gesellschaftliches Statement, das die Absicht verfolgt, Software der ganzen Menschheit zu Gute kommen zu lassen. Dem gegenüber steht die Bewegung Open Source Initiative (OSI[2]) mit der Definition der Open Source Software (OSS). Grundsätzlich sehr ähnlich definiert, positioniert die OSI den Begriff der OSS weniger als politisches, sondern eher als marketingstrategisches Instrument. Die größten Unterschiede bei den Initiativen bestehen in der Anwendung ihrer Definitionen. Während die FSF sich ausschließlich auf Software bezieht, werden bei der OSI auch Wissen und Information im Allgemeinen miteinbezogen. Ein Beispiel dafür sind die Creative Commons Lizenzen[3]. Weiterhin kann Software, die von einer OSS-Lizenz abgeleitet wird, unter bestimmten Voraussetzungen auch nicht quelloffen und nicht frei sein, was bei den Lizenzen der FSF nicht möglich ist. OSS ist demnach nicht mit freier Software gleichzusetzen.

CoRS-O-Mat wird als freie Software im Sinne der FSF öffentlich zur Verfügung gestellt.

---

[1] Vgl. [WEB.FSF]

[2] Webseite der Open Source Initiative [https://opensource.org]

[3] Vgl. [WEB.CC]

3.2. Offene Standards

Von Standardisierung spricht man im Allgemeinen, wenn bestimmte Eigenschaften oder Merkmale von Produkten oder Systemen vereinheitlicht und in einer Spezifikation definiert werden. Man unterscheidet zwischen proprietären und offenen Standards. Proprietäre Standards werden von einem oder mehreren Unternehmen entwickelt und deren Aufbau wird nicht veröffentlicht. Man spricht von einem offenen Standard, wenn folgende Bedingungen zutreffen:

- Der Standard kann öffentlich bewertet und genutzt werden und ist für alle zugänglich.
- Der Standard enthält keine Komponenten oder Erweiterungen, die selbst nicht einem offenen Standard entsprechen.
- Der Standard enthält keine juristischen oder technischen Klauseln, welche die Nutzung unter bestimmten Voraussetzungen einschränken.
- Der Standard wird unabhängig von einem einzelnen Unternehmen weiterentwickelt in einem Prozess, bei dem Wettbewerber und Dritte teilnehmen können.
- Der Standard ist für alle Beteiligten verfügbar als vollständige Implementierung oder in verschiedenen vollständigen Implementierungen von verschiedenen Anbietern.

Stehen allen Wettbewerbsteilnehmern Standards gleichermaßen zur Verfügung, so kann von einem Wettbewerbsgewinn und einer damit einhergehenden Nutzensteigerung ausgegangen werden.

CoRS-O-Mat wird unter Nutzung offener Standards implementiert. Dadurch wird gewährleistet, dass die Anwendungssoftware wiederverwendbar und erweiterbar ist.

3.3. Java Enterprise Edition (Java EE)

Java EE ist eine Spezifikation, die eine Java-basierte Softwarearchitektur definiert. Mit Frameworks, die diese Architektur implementieren, ist es möglich, skalierbare verteilte Anwendungssoftware zu erzeugen. Java EE ist aktuell in der Version 7 und wird durch den Java Community Process (JCP), dem verschiedene Unternehmen angehören, weiterentwickelt und in Form eines Dokuments zusammen mit einer Referenzimplementierung veröffentlicht. Ein Ziel der Java EE Spezifikation ist es, dass die Komponenten verschiedener Hersteller interoperabel und austauschbar sind. Zu den Aufgaben des JCP gehören die Weiterentwicklung der Programmiersprache Java und der Java Plattformen (Java SE, Java EE, Java ME), die als Java Specification Requests (JSR) ausgearbeitet werden. Java EE basiert auf Java SE und ist prinzipiell eine aufeinander abgestimmte Sammlung von APIs, die in verschiedenen JSRs definiert wurden.

Im folgenden ein Auszug aus den in CoRS-O-Mat verwendeten Java EE 7 JSRs[4]

- JSR 337: Java Standard Edition 8 (Java SE 8)
- JSR 315: Java Servlet 3.0
- JSR 222: Java Architecture for XML Binding (JAXB) 2.0
- JSR 338: Java Persistence API (JPA) 2.1
- JSR 318: Enterprise JavaBeans (EJB) 3.1
- JSR 365: Contexts and Dependency Injection for Java (CDI) 2.0
- JSR 372: JavaServer Faces (JSF) 2.3
- JSR 341: Expression Language (EL) 3.0
- JSR 907: Java Transaction API (JTA)

CoRS-O-Mat wird aus den folgenden Gründen als webbasierte Java EE Applikation umgesetzt: Java EE ist ein offener Standard und hat sich zum Industriestandard entwickelt. Es wird von namhaften Unternehmen, darunter Oracle, Google, IBM und SAP, im JCP weiterentwickelt. Zu den Spezifikationen existieren sowohl freie Referenzimplementierungen, als auch weitere freie Implementierungen und auch proprietäre Implementierungen mit professionellem Support. Diese Vorteile sind beispielsweise bei PHP, Ruby, NodeJS oder Microsoft .NET nicht vorhanden. Java ist 2015 laut TIOBE Index die populärste und gefragteste Programmiersprache gewesen[5]. Da Java seit mehreren Jahren eine der beliebtesten Programmiersprachen ist, gibt es weltweit viele Java-Entwickler und dadurch ist davon auszugehen, dass es im World Wide Web viele Hilfestellungen zu allen erdenklichen Themen gibt. Zum Implementieren von Java-basierter Anwendungssoftware existieren die freien Entwicklungsumgebungen Netbeans von Oracle und Eclipse von der Eclipse Foundation.

3.3.1. Java EE Laufzeitumgebung

Die Laufzeitumgebung einer Java EE Applikation ist ein Java EE Application Server, der auf der JVM ausgeführt wird. Java EE Komponenten können auf Laufzeitumgebungen unterschiedlicher Hersteller ausgeführt werden, wenn diese hinsichtlich der Konformität zur Spezifikation zertifiziert wurden.

Beispiele für Java EE 7 Application Server

[4]Vgl. Webseite vom JCP: Java EE JSRs [https://www.jcp.org/en/jsr/platform?listBy=3&listByType=platform]
[5]TIOBE Index für 2015 [http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html]

- GlassFish Server Open Source Edition 4.0
- IBM WebSphere Application Server Version 8.5.5.6
- Redhat WildFly Application Server 8.x

CoRS-O-Mat läuft auf einem Glassfish Application Server, da dieser die freie Referenzimplementierung einer Java EE Laufzeitumgebung ist.

3.3.2. Java EE Container

Java EE liegt ein Komponenten-Container-Modell zugrunde. Java EE Applikationen bestehen aus Komponenten, also abgeschlossenen Funktionseinheiten, die innerhalb von sogenannten Containern ausgeführt werden. Java EE Container stellen die spezifizierten Schnittstellen und Dienste, wie z. B. Transaktionsmanagement zur Verfügung, die von den jeweiligen Komponenten genutzt werden. Darüber hinaus wird die Kommunikation und der Lebenszyklus der Komponenten von den Containern verwaltet. Es gibt verschiedene Containertypen, zu den wichtigsten gehören EJB Container und Web Container.

3.3.3. Java Database Connectivity (JDBC)

JDBC ist eine Datenbankschnittstelle von Java, deren Aufgaben u. a. darin bestehen, Datenbankverbindungen aufzubauen und zu verwalten, Anfragen an die Datenbank weiterzuleiten und die Ergebnisse der Anwendung zur Verfügung zu stellen. Durch JDBC ist es möglich, SQL-Anweisungen auf einer Datenbank auszuführen. Die zu benutzende Datenbank muss über einen Treiber, der die JDBC-Spezifikationen implementiert, eingebunden werden. Dieser Treiber wird in der Regel vom Datenbankhersteller zur Verfügung gestellt. Da viele relationale Datenbanken eine eigene SQL-Syntaxdefinition besitzen, kann ein Wechsel der Datenbank ohne weitere Abstraktionsmechanismen einen erheblichen Aufwand produzieren.

CoRS-O-Mat verwendet JDBC, um sich mit relationalen Datenbanken verschiedener Hersteller zu verbinden und auf ihnen operieren zu können.

3.3.4. Java Persistence API (JPA)

JPA ist eine Menge von Schnittstellen, um Java-Objekte in persistente Datenbankeinträge zu überführen und umgekehrt (objekt-relationales Mapping). JPA definiert ein Standardmapping zwischen einer Datenbanktabelle und einem Plain Old Java Object (POJO) mittels annotierten Metadaten innerhalb der Entity-Klassen. Mit Hilfe der Annotationen lassen sich Beschreibungen der Tabellen und Relationen in die Klassen einfügen. JPA stellt den EntityManager zur Verfügung, mit dessen Hilfe Objekte erzeugt, gelesen, aktualisiert, gelöscht und gesucht werden

können. Der EntityManager kann auf beliebige Speichermedien angesetzt werden und ist deshalb nicht zwingend und ausschließlich mit relationalen Datenbanken verwendbar. Als Abfragesprache wird die Java Persistence Query Language (JPQL[6]) verwendet. Dadurch ist es möglich, Abfragen auf den Datenbestand ohne spezifische SQL-Befehle im Programmcode zu formulieren, was den Austausch der zugrundeliegenden Datenbank erheblich vereinfacht.

Die Referenzimplementierung von JPA, EclipseLink, kommt bei CoRS-O-Mat zum Einsatz.

3.3.5. Enterprise Java Beans (EJB)

Enterprise Java Beans sind Komponenten innerhalb einer Java EE Applikation, die die Entwicklung mehrschichtiger skalierbarer Anwendungssoftware vereinfachen. Sie stellen Funktionen zur Verfügung, die das Ausführen transaktionsbasierter Aufgaben ermöglicht. Es gibt folgende Typen von EJBs: Stateless und Stateful Session Beans und Message Driven Beans.

Session Beans bilden Benutzervorgänge ab und lassen sich in zustandslos (stateless) und zustandsbehaftet (stateful) einteilen. Stateless Session Beans haben kein Zustandsgedächtnis, d. h. ihr aktueller Zustand geht nach Beenden einer Transaktion verloren. Für jede Transaktion wird vom EJB Container eine beliebige EJB Instanz aus einem Instanzenpool verwendet, da sie sich nicht unterscheiden. Stateful Session Beans speichern ihren aktuellen Zustand und können deshalb nicht beliebig ausgetauscht werden. Für jede Transaktion muss immer die jeweils bereits verwendete EJB aufgerufen werden.

Message Drive Beans ermöglichen asynchrone Kommunikation mit EJB Komponenten unter Verwendung des Java Message Service (JMS). Diese Art von Bean wird z. B. zur Kommunikation mit Altsystemen verwendet oder zur Ausführung asynchroner Aufgaben, wie das Verschicken einer E-Mail.

In CoRS-O-Mat werden Enterprise Java Beans ausschließlich als zustandslose EJBs verwendet.

3.3.6. Context Dependency Injection (CDI)

Context Dependency Injection ist das Bean Management und Dependency Injection Framework, das mit Java EE 6 eingeführt und zum Standard erhoben wurde. CDI ermöglicht die Konfiguration von Modulen durch Injektion von Abhängigkeiten. CDI Beans sind interoperabel zu EJBs, CDI Beans können in EJBs injiziert werden und umgekehrt und direkt auf JSF-Seiten mittels EL-Ausdrücken eingebunden werden. CDI ist der aktuelle Standard, hat aber nicht alle Leistungsmerkmale von EJB, deshalb kommen bei CoRS-O-Mat EJBs und CDI Beans zum Einsatz, wie in Abschnitt 4.6.2, „ManagedBeans, CDI Beans und EJBs“ beschrieben.

[6]Dokumentation zur JPQL [http://docs.oracle.com/cd/E12839_01/apirefs.1111/e13946/ejb3_langref.html]

3.3.7. Java Server Faces (JSF)

JSF ist als Standard-Benutzeroberfläche für webbasierte Java EE Anwendungssoftware festgelegt. Oracle Mojarra[7] ist die Referenzimplementierung der JSF-Spezifikation und wird aus diesem Grund in CoRS-O-Mat verwendet.

JSF stellt eine Komponentenbibliothek und ein API zur Erzeugung und Verwaltung von Benutzeroberflächenkomponenten zur Verfügung. Als View Handler wird bei JSF standardmäßig Facelets verwendet. Die gesamte Benutzeroberfläche wird in Form einer XML-basierten Baumstruktur definiert und abschließend von einer Renderkomponente in das gewünschte Ausgabeformat, z. B. XHTML5, überführt. Durch Facelets ist es auch möglich, Templates für das Oberflächenlayout zu definieren und diese Teile wiederzuverwenden. Weiterhin stellt JSF Funktionen zur Navigation, Zustandverwaltung der Komponenten, Ereignisverarbeitung und Validierung von Benutzereingaben zur Verfügung.

JSF trennt strikt zwischen Präsentation, Modell und Steuerung und setzt damit das Entwurfsmuster MVC (Model View Controller) um. Die Benutzerkomponenten werden in XHTML-Dateien in einer XML-basierten Auszeichnungssprache definiert und können Daten mit in Java programmierten Managed Beans austauschen. Managed Beans haben einen Gültigkeitsbereich, der durch spezielle Annotationen festgelegt werden kann. Mögliche Werte sind

- RequestScoped
  Gültig, solange die aktuelle Anfrage bearbeitet wird.
- ViewScoped
  Gültig, solange der Lebenszyklus der aktuellen Sicht noch nicht beendet wurde.
- SessionScoped
  Gültig, solange die aktuelle Session gültig ist.
- ApplicationScoped
  Gültig für die Lebensdauer der Applikation
- NoneScoped
  So lange gültig, wie die übergeordnete Bean-Klasse.

Mit jeder Anfrage durchläuft die JSF-Anwendung einen Lebenszyklus, der in sechs Phasen aufgeteilt ist:

- Restore View

[7] Webseite von Oracles JSF Referenzimplementierung Mojarra [https://javaserverfaces.java.net]

Bei einer erstmaligen Anfrage wird eine neue Sicht mit einem leeren Komponentenbaum erzeugt, diese wird bei darauffolgenen Anfragen wiederverwendet.

- Apply Request Values
  Hier werden die Daten aus der Anfrage, üblicherweise Daten eines Formulars, in die entsprechenden Felder des Komponentenbaums übertragen.

- Process Validations
  In dieser Phase werden die definierten Konverter und Validatoren ausgeführt, um die Daten einerseits in die benötigten Datentypen zu konvertieren und andererseits auf Korrektheit zu überprüfen.

- Update Model Values
  Hier wird das Modell mit den neuen Werten aktualisiert.

- Invoke Application
  Die Ereignisse auf der Anwendungsebene werden jetzt verarbeitet, wenn z. B. ein Button gedrückt wurde, wird der entsprechende ActionListener ausgeführt.

- Render Response
  Diese Phase erzeugt die Antwort auf die Anfrage, indem der Komponentenbaum durchlaufen wird und auf jedem Element die encode-Methode aufgerufen wird.

Die Phasen müssen nicht alle durchlaufen werden, es ist möglich, einzelne Phasen zu überspringen, wenn z. B. eine HTML-Seite ohne JSF-Komponenten angezeigt werden soll oder wenn ein Fehler aufgetreten ist, kann auch sofort eine Antwort generiert werden.

Die Definition des JSF Komponentenbaums ist durch die Renderengine prinzipiell entkoppelt vom endgültigen Ausgabeformat, es können neben HTML auch andere Ausgabeformate erzeugt werden.

In CoRS-O-Mat kommt die Referenzimplementierung von JSF, Oracle Mojarra, zur Anwendung. Im Gegensatz zu einem clientseitigen GUI-Framework, wie z. B. AngularJS, basiert die serverseitige GUI JSF darauf, dass alle Aktionen und Anweisungen auf dem Server verarbeitet werden und nicht im Client. Das bedeutet, dass die komplette Businesslogik auf dem Server gekapselt ist und nichts davon an den Browser übermittelt wird und es bedeutet insbesondere, dass keine weiteren Webservices erzeugt werden müssen, um zusätzliche Funktionalität, wie z. B. AJAX-Aufrufe, zu implementieren.

3.3.8. Unified Expression Language (EL)

Die Unified Expression Language ermöglicht das Einbinden von Eigenschaften und Methoden von Managed Beans innerhalb von JSF-Seiten. Hierfür stellt die EL eine eigene Syntax mit Value Expressions und Method Expressions zur Verfügung

CoRS-O-Mat benutzt EL, um dynamische Inhalte in XHTML darzustellen.

3.3.9. Java Transaction API (JTA)

Die Java Transaction API ermöglicht die Verwendung von Transaktionen über mehreren Ressourcen. Methoden können als Transaktion gekennzeichnet werden und die Auslösung von Ereignissen, wie Commit oder Roll-Back, kann variabel in den Programmablauf eingebaut werden.

In CoRS-O-Mat werden transaktionsbasierte EJBs verwendet. EJB Container nutzen JTA zur Transaktionsverwaltung und ermöglichen, dass transaktionsbasierte Interaktionen mit den annotierten EJBs möglich sind.

3.4. Apache Maven[8]

Maven ist ein Java-basiertes Build-Management-Werkzeug der Apache Software Foundation und dient dem Zweck, den Entwickler soweit bei der Softwareentwicklung zu unterstützen, dass möglichst viele Schritte im Entwicklungsprozess automatisiert werden können. Maven folgt dabei der Prämisse Konvention vor Konfiguration, d. h., dass im Allgemeinen sinnvolle Standardwerte existieren, wenn nichts anderes konfiguriert wurde. Das bedeutet, wenn man den vorgegebenen Konventionen folgt, muss sehr wenig zusätzlich konfiguriert werden. Ein großer Vorteil von Maven ist die Integration in die IDEs Eclipse und Netbeans.

Maven folgt dem objektorientierten Programmierparadigma, indem es hierarchische, auf XML basierende, Projektkonfigurationsdateien verwendet, die sogenannten POM (Project Object Model) Dateien. Diese werden hierarchisch, also Objekt für Objekt, abgearbeitet und nicht sequentiell, wie es bei anderen Build-Management-Werkzeugen, wie z. B. Ant, der Fall ist. In einer POM-Datei können eine Vielzahl an Knoten mit Attributen definiert werden, um Konfigurationen vorzunehmen, Abhängigkeiten zu definieren, PlugIns zu laden oder Aktionen auszuführen, wie beispielsweise das Kopieren von Dateien innerhalb einer bestimmten Build-Phase. Definierte Abhängigkeiten zu anderen Maven-Projekten werden in ein lokales Repository abgelegt und bei Bedarf wiederverwendet, um Netzressourcen zu sparen. POM-Dateien können an andere Projekte vererbt werden. Dadurch lassen sich hierarchische Projektstrukturen bauen, die keine Konfigurationsredundanzen enthalten.

[8] Apache Maven Webseite [https://maven.apache.org]

Der Standard-Lebenszyklus eines Softwareprojekts ist in Maven als Durchlauf durch mehrere Phasen definiert. Nicht jedes Softwareprojekt muss dabei alle Phasen durchlaufen, es können einzelne Phasen weggelassen oder übersprungen werden, es können aber auch neue Phasen definiert und hinzugefügt werden. Die Phasen eines Standard-Lebenszyklus sind wie folgt:

- archetype
  Es ist möglich, Projektvorlagen, sogenannte archetypes zu verwenden. In dieser Phase werden die zugehörigen Abhängigkeiten aufgelöst und heruntergeladen.

- validate
  In dieser Phase wird die Projektstruktur auf Vollständigkeit und Gültigkeit überprüft.

- compile
  Hier wird der Quellcode kompiliert.

- test
  In dieser Phase werden die definierten Tests innerhalb des definierten Testframeworks ausgeführt. Mögliche Frameworks sind z. B. Unit Tests mit JUnit oder TestNG. Weitere Frameworks können eingebunden werden.

- package
  Die kompilierte Software wird zusammen mit den Ressourcen in eine Paketdatei geschrieben, das kann eine WAR (Web Archive)-Datei, eine JAR (Java Archive)-Datei oder eine POM (Project Object Model)-Datei sein.

- integration-test
  Das Paket wird auf einen anderen Server geladen und auf korrekte Funktionalität geprüft.

- verify
  Hier wird das Paket auf seine Gültigkeit hin überprüft.

- install
  In dieser Phase wird die Software ins lokale Maven-Repository installiert, wodurch es durch andere Maven-Projekte wiederverwendet werden kann.

- deploy
  Die fertige Software wird ins definierte Web-Repository geladen und steht dann zum Download bereit. Sie kann als fertige Releaseversion oder auch als Vorabversion, der sogenannten SNAPSHOT-Version, veröffentlicht werden.

CoRS-O-Mat verwendet Maven für das Build-Management, d. h. die Komponenten der Anwendungssoftware CoRS-O-Mat sind als Maven-Projekte realisiert.

3.5. Unified Modeling Language (UML)

Um komplexe Zusammenhänge verständlich und nachvollziehbar zu visualisieren, spezifizieren oder zu dokumentieren, ist es nötig, von den konkreten und meist komplexen Beziehungen zu abstrahieren und ein Modell zu entwickeln, welches alle Elemente und deren Beziehungen auf wesentliche Aspekte reduziert darstellt. Eine Sprache, die ein solches Modell beschreibt, ist UML. Um die verschiedenen Aspekte eines Systems vollständig beschreiben zu können, sind in UML in der aktuellen Version 2.5 13 verschiedene Diagrammtypen definiert, welche sich in drei Hauptgruppen unterteilen lassen: Strukturdiagramme, Verhaltensdiagramme und Interaktionsdiagramme[9]. Die Grenzen zwischen den Gruppen sind weich, d. h. es ist durchaus gestattet, Elemente eines Diagrammtyps in einem anderen Typ zu verwenden. Per Definition stellen Strukturdiagramme statische Strukturen dar, Verhaltensdiagramme allgemeines Verhalten und Interaktionsdiagramme verschiedene Aspekte von Interaktionen mit Software.

- Zu den Strukturdiagrammen zählen die Typen Klassen-, Objekt-, Komponenten-, Kompositionsstruktur-, Paket- und Verteilungsdiagramm.

- Zu den Verhaltensdiagrammen zählen die Diagrammtypen Anwendungsfall-, Aktivitäts- und Zustandsdiagramm

- Zu den Interaktionsdiagrammen gehören die Typen Sequenz-, Kommunikations-, Zeitverlaufs- und Interaktionsübersichtsdiagramm.

In der Anwendungssoftware CoRS-O-Mat wird mit Hilfe des Eclipse-PlugIns Papyrus[10] ein UML-Klassendiagramm erstellt, welches das Domänenmodell definiert. Dieses Modell wird als Basis für den zu erzeugenden Code verwendet. Im ersten Schritt wird das Modell in ein Datenbankschema überführt, welches mit Hilfe des Eclipse-PlugIns Liquibase[11] abstrahiert repräsentiert wird. Im zweiten Schritt wird aus diesem die Datenbank erzeugt. Danach werden mit Hilfe von JPA alle Entity-Klassen generiert.

3.6. Scrum

Scrum ist ein inkrementeller, agiler Prozess für die Softwareentwicklung und das Projektmanagement. Für Teilnehmende am Prozess werden folgende Rollen definiert[12]:

- Der Product Owner definiert die fachlichen Anforderungen und priorisiert sie.

---

[9] Vgl. [WEB.UML]

[10] Webseite des Papyrus-PlugIns für Eclipse [https://eclipse.org/papyrus/]

[11] Webseite des Liquibase-PlugIns für Eclipse [http://www.liquibase.org]

[12] Vgl. [WIRDEMANN.2011], S. 36ff.

- Der Scrum Master verwaltet den Prozess und beseitigt Hindernisse, die dem Team im Wege stehen.

- Das Entwicklungsteam entwickelt das Produkt.

Die Anforderungen an das Produkt werden im Product Backlog eingetragen und priorisiert. In bestimmten Zeitabständen wird dem Backlog das Inkrement (ein Arbeitspaket mit hoher Priorität) entnommen und vollständig umgesetzt, d. h. inklusive Entwicklung, Tests und Dokumentation. Während dieser laufenden Iteration, dem sogenannten Sprint darf das Inkrement nicht mehr modifiziert werden, um den Sprint nicht zu gefährden. Die restlichen Teile des Backlogs können nach Belieben verändert werden. Im Sprint wird das Inkrement in kleinere Arbeitspakete, den Tasks unterteilt. Diese Tasks werden einem Bearbeiter zugeordnet und es wird ein Sprint Backlog mit den übrigen Tasks angelegt. Es findet ein tägliches, 15-Minütiges Meeting statt, das sogenannte Daily Scrum, in dem jeder Beteiligte die Anderen darüber informiert, woran er zuletzt gearbeitet hat, was als nächstes rankommt und welche Probleme aufgetreten sind. Nach Fertigstellung des Sprints werden die Ergebnisse dem Product Owner präsentiert, das gesammelte Feedback und eventuelle neue Anforderungen werden in das Sprint Backlog eingebaut. Dann startet der Prozess wieder von vorne, solange bis das Produkt fertiggestellt ist.

Kapitel 4. CoRS-O-Mat Softwareentwurf

Die folgenden Abschnitte beschreiben den Entwurf der Software CoRS-O-Mat und erläutern die Gründe für die jeweils getroffenen Entwurfsentscheidungen.

4.1. Anforderungen

Die Anforderungen an CoRS-O-Mat werden mittels User Stories verfeinert. User Stories sind allgemeinsprachlich formulierte Anforderungen, also kurze Benutzergeschichten, aus denen technische Anforderungen abgeleitet werden können. Sie werden im weiteren Verlauf der Entwicklung immer weiter verfeinert und konkretisiert. Diese User Stories können dann initial in das Backlog überführt werden und bilden die Anfangsmenge der zu bearbeitenden Tasks nach Scrum.

1. Sätze erfassen
   Als registrierter Benutzer möchte ich Sätze (inkl. Literaturangaben) erfassen.

   Akzeptanzkriterien:

   - Der Benutzer kann einen Satz erfassen
   - Der Benutzer kann den Satz einer der hinterlegten romanischen Sprache zuordnen
   - Der Benutzer kann den Satz einem der hinterlegten Satztypen zuordnen
   - Der Benutzer kann dem Satz Sprecherurteile bezüglich der Grammatikalität und der Informationsstruktur zuordnen
   - Der Benutzer kann dem Satz Literaturangaben hinzufügen
   - Der erfasste Satz wird gespeichert

2. Sätze annotieren
   Als registrierter Benutzer oder Datenerfasser möchte ich Sätze annotieren.

   Merkmalstypen:

   1. Konstituentenstrukturmerkmale
   2. morpho-syntaktische Merkmale
   3. Informationsstrukturmerkmale
   4. funktionale Merkmale

Akzeptanzkriterien:

- Der Benutzer kann einen Satz aus dem Datenbestand annotieren
- Der Benutzer kann den Satz automatisch in Wörter zerlegen lassen
- Der Benutzer kann einzelne Wörter zusätzlich manuell zerlegen
- Der Benutzer kann einem Wort oder einer Wortgruppe die entsprechenden Merkmalstypen (Typ 1 - 4) zuordnen

3. Sätze abfragen
   Als öffentlicher Benutzer möchte ich Sätze mit bestimmten Wortstellungsmustern und bestimmten Merkmalskonfigurationen abfragen und die Abfragen für spätere Wiederverwendung speichern.

   Akzeptanzkriterien:

   - Der Benutzer kann nach Sätzen mit bestimmten Wortstellungsmustern und bestimmten Merkmalskonfigurationen suchen
   - Die Suche liefert eine Liste von gefundenen Sätzen
   - Der Benutzer kann seine Anfrage für eine spätere Wiederverwendung speichern

4.2. Wiederverwendung durch andere Projekte

Die definierten Anforderungen können im Prinzip durch eine monolithische Softwareanwendung umgesetzt werden. Das würde jedoch eine Wiederverwendung der Anwendungssoftware in einem anderen Umfeld erheblich erschweren. Um eine möglichst einfache Wiederverwendbarkeit und Anpassbarkeit zu gewährleisten, wird CoRS-O-Mat als komponentenbasierte Java EE 7 Architektur entwickelt. Es existiert bereits eine Anfrage des sprachwissenschaftlichen Forschungsprojekts "Quantification in Old Italian"[1], welches aufgrund ähnlicher Anforderungen die Anwendungssoftware CoRS-O-Mat zur Unterstützung der Forschungsarbeit einsetzen würde.

[1] Projektseite vom DFG GEPRIS [http://gepris.dfg.de/gepris/projekt/256240798]

4.3. Softwarearchitektur

CoRS-O-Mat wird als webbasierte[2] Anwendungssoftware mit einer komponentenbasierten Architektur umgesetzt. Diese sind durch ein REST-API lose miteinander gekoppelt. Insbesondere sollen somit folgende Eigenschaften umgesetzt werden können[3]:

- Austauschbarkeit
  Es sollte möglich sein, einzelne Komponenten auszutauschen, ohne andere Komponenten zu beeinflussen. Gründe für einen nötigen Austausch könnten fehlerhaft funktionierende Komponenten sein oder neue Komponenten mit erweitertem Funktionsumfang.

- Erweiterbarkeit und Reduzierbarkeit
  Es sollen neue Komponenten in das System eingebracht werden können, wenn sie beispielsweise das System um neue Funktionalität erweitern, oder auch entfernt werden können.

- Wiederverwendbarkeit
  Es sollte ohne weiteres möglich sein, Komponenten in einem anderen, kompatiblen System einsetzen und damit wiederverwenden zu können.

- Zusammenbau aus Standardmodulen
  Das System sollte überwiegend aus Standardmodulen zusammengebaut sein, um die konkreten Implementierungsaufwände möglichst gering zu halten und die Interoperabilität mit anderen Standardkomponenten zu maximieren[4].

Eine weitere Eigenschaft einer komponentenbasierten Architektur, die bei OECHSLE.2013 nicht explizit erwähnt wird, ist die Vereinfachung der Softwaretests des Gesamtsystems. Durch den gewählten Aufbau ist es einfacher, die Komponenten separat auf Korrektheit zu testen, denn es ist möglich, die Abhängigkeiten von außen durch sogenanntes Mocking in Java nachzuahmen und damit nur die eigentliche Funktionalität zu testen.

Die folgende Abbildung veranschaulicht die Komponenten von CoRS-O-Mat.

---

[2]Zu den Hauptgründen für diese Entwurfsentscheidung gehören die Plattformunabhängigkeit der Software und die stetig wachsende Verfügbarkeit internetfähiger Mobilgeräte. Eine Desktopapplikation würde nur auf Rechnern lauffähig sein und würde die Geräteklassen der Smartphones und Tablet-Rechner ausschließen. Da diese Geräte heutzutage jedoch leistungsfähige Internetbrowser besitzen, ist durch eine webbasierte Anwendungssoftware eine größere Geräteabdeckung möglich. Es wäre prinzipiell auch vorstellbar, eine Desktopapplikation für JRE-fähige Rechner zu entwickeln und native Apps für Smartphones und Tablets, jedoch würde das bedeuten, mehrere verschiedene Quellcodestränge in verschiedenen Programmiersprachen warten zu müssen, was grundsätzlich nicht erwünscht ist und dem angestrebten Ziel der Plattformunabhängigkeit unter Verwendung eines gemeinsamen Quellcodes widerstreben würde.

[3]Vgl. [OECHSLE.2013], S.123 ff.

[4]Vgl. [HERDEN.ET.AL.2006], S.37-38

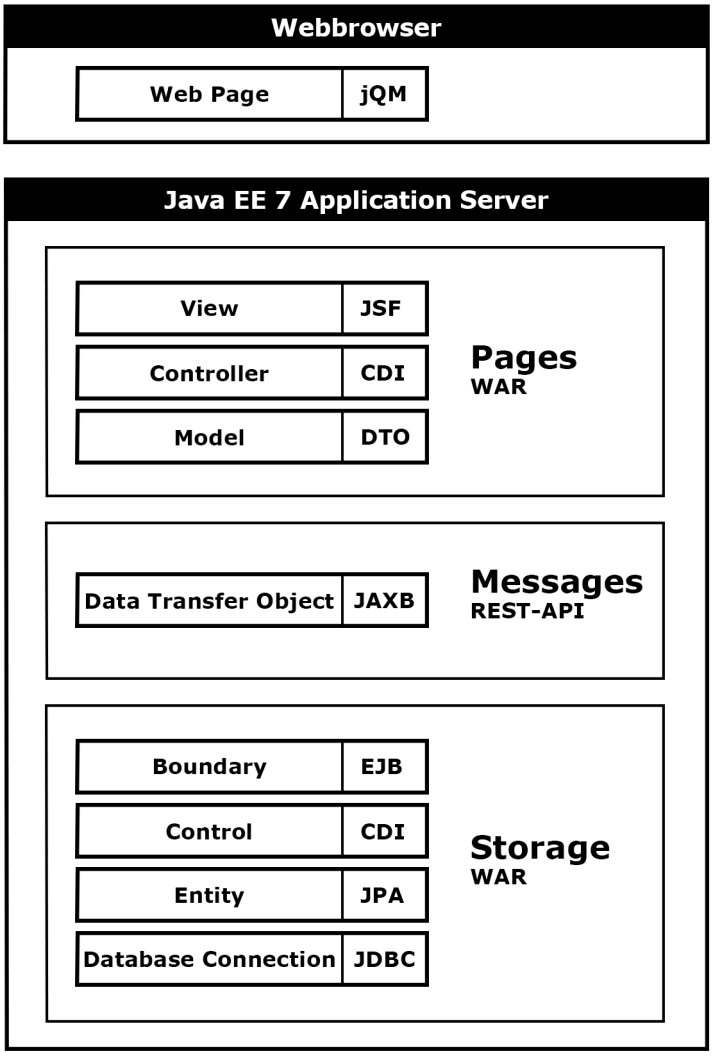

Abbildung 4.1. Die Komponenten der CoRS-O-Mat Softwarearchitektur

CoRS-O-Mat ist eine Client-Server-Architektur. Der Client ist ein Webbrowser, der eine HTML5-konforme Webseite darstellt. Der Server ist ein Java EE 7 Application Server und besteht aus einer Sammlung von Java EE WARs welche über HTTP miteinander kommunizieren. Im Wesentlichen sind das die Komponenten Pages, Messages und Storage.

Die Komponente Messages bildet als REST-API eine Schnittstelle zwischen den Komponenten Pages und Storage. Sie kapselt die Entity-Objekte der JPA-Persistenzschicht, damit diese allen beteiligten Komponenten, außer Storage, nicht bekannt sein müssen. Messages ist als JAR-Datei in Pages und Storage eingebunden.

Die Komponente Pages erzeugt die Webseiten, die im Webbrowser zur Darstellung gebracht werden. Diese Seiten können mit Aktionsbuttons und den entsprechenden Aktionen (z. B. CREATE, UPDATE und DELETE) angereichert werden, in Abhängigkeit von den zugeteilten Benut-

zerrechten. Pages kommuniziert mit dem Browser über HTTP mit HTML/CSS-Dokumenten. Pages kommuniziert mit Storage über REST und verwendet Messages-Objekte.

Die Komponente Storage hat die Verbindung zur Datenbank. Sie ist für das objektrelationale Mapping der Tabellendaten und den Entity-Klassen zuständig. Sie liest die Daten aus der Datenbank und erzeugt daraus Entity-Objekte, übersetzt diese in Messages-Objekte und stellt diese über eindeutige URIs mittels REST der Pages-Komponente zur Verfügung. Die Kommunikation zur Datenbank erfolgt über JDBC und das objektrelationale Mapping wird durch JPA realisiert.

4.3.1. Softwarekomponenten

In den folgenden Abschnitten werden die einzelnen Komponenten Web Page, Messages, Pages und Storage und deren geplante Umsetzung detaillierter beschrieben.

4.3.1.1. Web Page

Die Webseiten setzen die Konzepte Responsive Webdesign und Progressive Enhancement um. Responsive Webdesign ist ein Konzept, welches dafür sorgt, dass sich die darzustellende Webseite dynamisch an verschiedene Bildschirmgrößen, und damit Geräteklassen, anpassen kann. Üblicherweise wird diese Funktionalität mittels CSS Media Queries[5] realisiert. Um das Konzept des Responsive Webdesigns im CoRS-O-Mat zu implementieren, wird die freie Softwarebibliothek jQuery mobile[6] verwendet. JQuery mobile ist kompatibel zu älteren Browsern, wie z. B. Internet Explorer 8[7]. Das Konzept des Progressive Enhancements definiert, dass zuerst die einfachste Grundfunktionalität für alle Browser und Geräte implementiert wird und dann progressiv komplexere und modernere Technologien dazugeschaltet werden, so dass die webbasierte Anwendungssoftware am Ende in allen Browsern funktioniert, ohne dass verschiedene Codeversionen erzeugt werden müssen. Durch die Abwärtskompatibilität von jQuery mobile und den Verzicht auf JSF-AJAX Benutzeroberflächenkomponenten funktioniert CoRS-O-Mat sowohl in älteren Browsern ohne JavaScript, als auch in modernen Browsern der neuesten Generation.

4.3.1.2. Messages

Die Komponente Messages stellt serialisierbare Objekte zur Verfügung, welche per REST (JAX-RS) verschickt werden können. Die Serialisierung wird mit JAX-B (Java Architecture for XML Binding) durch entsprechende Annotationen realisiert. Damit wird das Entwurfsmuster Data Transfer Object (DTO) in Messages umgesetzt. DTO wird verwendet, wenn Daten in Objekte aggregiert werden, um nicht mehrere Datenverbindungen öffnen zu müssen, sondern pro Objekt

[5]Webseite des W3C [https://www.w3.org/TR/css3-mediaqueries/]

[6]Webseite von jQuery mobile [https://jquerymobile.com]

[7]Webseite zur Browserkompatibilität von jQuery mobile [https://jquerymobile.com/browser-support/1.4/]

eben nur eine. Die Daten sind in den Objekten hierarchisch in einer Baumstruktur angeordnet und kommen ohne komplexe Beziehungen zwischen mehreren Objekten aus. Die Messages-Objekte verbergen Implementierungsdetails der Datenbank und der JPA-Entity-Objekte.

4.3.1.3. Pages

Die Komponente Pages untergliedert sich in drei Schichten: View, Controller und Model. Es wird hier das Entwurfsmuster Model View Controller (MVC) umgesetzt. MVC ist ein Softwareentwicklungsmuster zur Strukturierung von Anwendungen mit dem Ziel, dass diese Anwendungen leichter zu erweitern und wiederzuverwenden sind. Model enthält die Daten oder Datenobjekte, View ist zuständig für die Darstellung und das Entgegennehmen von Benutzeraktionen und Controller validiert diese und führt sie aus. In der Implementierung von CoRS-O-Mat erzeugt View die Webseiten, welche im Webbrowser zur Ansicht gebracht werden und verwendet dafür JSF in der aktuellen Referenzimplementierung Mojarra von Oracle. Zur View gehören die XHTML-Dateien und alle Webressourcen wie CSS-Dateien, JavaScript-Dateien und Bilddateien. Model ist indirekt durch die Einbindung der Messages-JAR-Datei als abstrahierte Entitys gegeben. Controller ist eine CDI-Bean, welche die Benutzeraktionen verarbeitet. Diese Bean wird innerhalb der View-XHTML-Dateien per Java Expression Language eingebunden.

4.3.1.4. Storage

Die Komponente Storage hat im Wesentlichen folgende Aufgaben: Sie kommuniziert mit der angebundenen Datenbank über JDBC, sie sorgt für die persistente Speicherung der Daten mittels JPA und sie stellt den anderen Komponenten Methoden zur Verfügung, Messages-Objekte über REST abzurufen oder zu manipulieren. Diese Komponente wurde nach dem Entwurfsmuster Entity Control Boundary (ECB) entwickelt. ECB ist im Prinzip eine Variation von MVC mit dem Unterschied, dass der wechselseitige Zugriff der einzelnen Bereiche eingeschränkt ist. Control befindet sich zwischen Entity und Boundary und der Zugriff kann auch nur in dieser linearen Anordnung stattfinden, d. h. Boundary kann nur auf Control zugreifen und nicht direkt auf Entity und umgekehrt. Entity enthält die Daten, Control führt die Logik aus und Boundary ist als Schnittstelle nach außen zu verstehen. Der Code der Storage-Komponente ist zur besseren Übersicht in zwei Schichten aufgeteilt: Transfer und Persistence. Transfer enthält CDI-Beans als Control und EJBs als Boundary. Control baut Messages-Objekte aus Entitys und zerlegt Message-Objekte bei Bedarf wieder, und Boundary definiert eindeutige REST-URIs mittels JAX-RS-Annotationen an den Methoden `delete`, `get`, `getById`, `post` und `put`. Persistence enthält JPA-Entitys, CDI-Beans als Control und EJBs als Boundary. Control persistiert und liest die Entitys aus der Datenbank und Boundary wird in der Transferschicht eingebunden, um die entsprechenden Entitys zu liefern.

4.3.2. Weitere Architekturmerkmale

Das Gesamtsystem wird als zustandslose Applikation implementiert. Das bedeutet, dass jede Zustandsinformation vom Client bei jeder Anfrage mitgeschickt werden muss. Das Server-Backend wird keine Zustandsinformationen in einer Session speichern, es wird eine Session ausschließlich aus Performanzgründen zum Teilen und einmaligen allokieren von Objekten, Ressourcen und Verbindungen nutzen. Das Frontend wird ebenso zustandslos umgesetzt, d. h., dass die aktuelle Webseite und der aktuelle Zustand zu jeder Zeit in der angezeigten Webadresse und den HTTP-Headerinformationen reflektiert wird und dass die eingebauten Browserfunktionen zur Seitennavigation (Vorwärts- und Zurückbutton) durch den Benutzer sinnvoll verwendet werden können. Der größte Vorteil einer zustandslosen Implementierung ist eine optimale Skalierbarkeit des Systems, da die einzelnen Server keine Zustandsinformationen speichern müssen. So können die eingehenden Anfragen auf eine dynamische Zahl von Servern verteilt werden, wie es z. B. in einer zeitgemäßen Server-Cloud-Infrastruktur der Fall ist.

Durch die Umsetzung als Client-Server-Architektur können Benutzer ohne großen Aufwand einen Zugang zur Software erhalten. Es muss lediglich eine Webseite aufgerufen und optional ein Benutzeraccount angelegt werden, mit dem eine Anmeldung auf der Webseite stattfinden kann. Die Beschaffenheit der Anwendungssoftware sieht vor, dass der Client eher weniger komplexe Aufgaben ausführt, während rechenintensivere Prozesse auf dem Server ausgeführt werden. Das ermöglicht die Nutzung der Webseiten auch auf hardwareschwächeren Geräten. Die Anwendungssoftware wird sich auf vielen aktuell verfügbaren Plattformen und Geräten optimal ausführen lassen können, wie z. B. Smartphones, Tablet-PCs, Laptop-PCs und Desktop-PCs verschiedener Hersteller und sie wird so gestaltet, dass sie in allen aktuell verfügbaren Browsern, wie z. B. Mozilla Firefox, Google Chrome, Apple Safari oder Microsoft Internet Explorer, auf allen webfähigen Betriebssystemen, wie Microsoft Windows, Linux, Apple OSX, Apple iOS oder Google Android mit der gleichen Codebasis lauffähig ist.

4.4. Datenbankentwurf

Das folgende Kapitel beschreibt den Entwurf der zugrundeliegenden Datenbank und des Datenmodells.

4.4.1. Datenbankabstraktion

CoRS-O-Mat soll mit jeder Datenbanktechnologie benutzt werden können, für die es einen vollständig implementierten JDBC-Treiber gibt. Zu den verwendbaren Datenbanken gehören bekannte DBMS wie H2 Database Engine, Apache Derby, MySQL, PostgreSQL und einige weitere.

Als objektrelationaler Mapper kommt die Referenzimplementierung von JPA, EclipseLink Version 2.5.1, zum Einsatz. Durch Annotationen, die durch JPA bereitgestellt werden, lassen sich Entitys als POJOs (Plain Old Java Objects) implementieren, welche als Datenbankabstraktion in der Anwendung verwendet werden können. POJOs lassen sich besonders gut in jede Anwendung integrieren, da sie als einfache Java Objekte keine weiteren Abhängigkeiten besitzen.

Das Erzeugen der benötigten Tabellen mit deren Attributen, Schlüsseln und den weiteren Parametern, sowie das Befüllen der Tabellen mit initialen Werten geschieht unter Verwendung des Open-Source Frameworks Liquibase[8]. Liquibase ermöglicht datenbankunabhängige, XML-basierte Definitionen aller Tabellen mit deren Parametern und Operationen auf den Daten. Liquibase kann mit Maven verwendet werden. Liquibase unterstützt alle DBMS, für die ein JDBC Treiber existiert. Die Definitionsdateien sind reine XML-Textdateien und können in das Versionierungssystem eingebunden werden, was Vergleiche zwischen unterschiedlichen historischen Versionen der Datenbank ermöglicht.

Eine mit Liquibase erzeugte Tabellendefinition sieht beispielsweise folgendermaßen aus:

```
1 <createTable tableName="LANGUAGES" >
2     <column name="ID" type="BIGINT" autoIncrement="true" >
3         <constraints nullable="false" primaryKey="true" />
4     </column>
5     <column name="NAME" type="VARCHAR(256)" >
6         <constraints nullable="false" />
7     </column>
8 </createTable>
```

Diese Anweisungen werden von Liquibase in Statements, z. B. SQL-Statements, umgewandelt, die von der spezifischen Datenbank verstanden und verarbeitet werden können. Diese Statements werden via JDBC auf der Datenbank ausgeführt. Auf diese Weise kann das vorliegende Datenbankschema in jeder beliebigen, kompatiblen Datenbank in Entitäten umgesetzt werden.

4.4.2. Datenbankmodell

Das Datenbankmodell wird in Zusammenarbeit mit den CoRS-Projektmitarbeitern modelliert. Als Modellierungssprache wird UML2 unter Nutzung des freien Eclipse-PlugIns Papyrus verwendet.

Die folgende Abbildung zeigt einen Auszug aus dem Domänenmodell, zu sehen ist hier die Entität `CLAUSES`, welche in der Domäne des Projekts CoRS die zentrale Entität darstellt. Das komplette Diagramm ist im Anhang dieser Arbeit zu finden: Abbildung A.1, „CoRS-O-Mat Domänenmodell als UML Diagramm“.

[8] Webseite von Liquibase [http://www.liquibase.org]

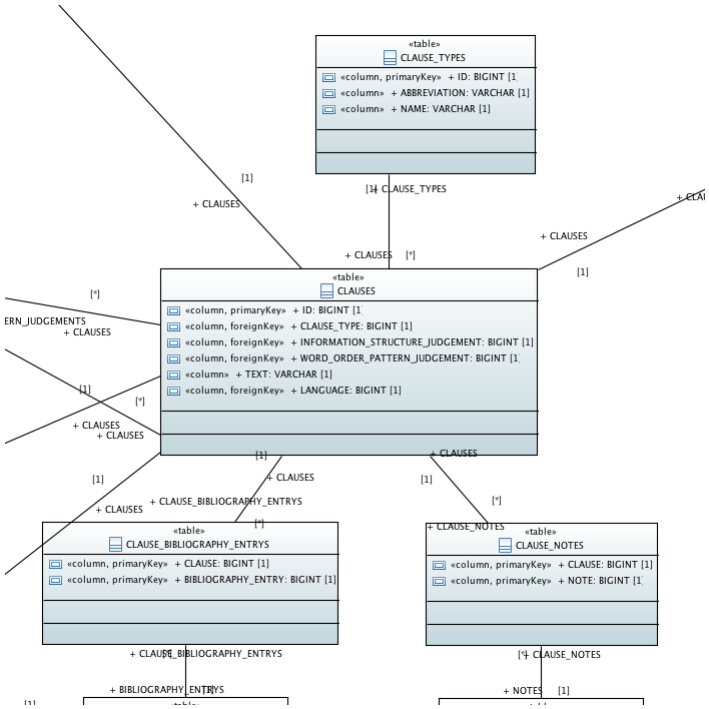

Abbildung 4.2. Auszug aus dem Domänenmodell

Bei der Definition des Modells werden vorab UML-Stereotypen für Datenbanktabellen, Datenbankspalten, Primär- und Fremdschlüssel definiert. Diese Stereotypen können den definierten Entitäten und Attributen dann als aktuelles UML-Profil zugewiesen werden. Die Zuweisung von Stereotypen könnte eine spätere automatische Generierung aller benötigten Dateien ermöglichen. Dazu gehören die Datenbank-Definitionsdateien für Liquibase und die Entity- und Messages-Klassen für die Komponenten Pages, Messages und Storage. Eine Möglichkeit, den Quellcode aus dem UML-Modell zu generieren, wäre mit MOF Model to Text Transformation Language (MOFM2T)[9] gegeben. Das wird zu einem späteren Zeitpunkt evaluiert und ist nicht Bestandteil dieser Arbeit.

4.4.3. Datenbankschlüssel

In der Anwendungssoftware CoRS-O-Mat werden die Datenobjekte durch Universally Unique Identifier (UUID)[10] referenziert, an Stelle von üblichen numerischen, automatisch inkrementierenden Primärschlüsseln. Eine UUID ist ein 16 Byte langer Binärwert, üblicherweise dargestellt durch 32 Hexadezimalzeichen in fünf Gruppen, getrennt durch Bindestriche, z. B. 550e8400-e29b-41d4-a716-446655440000.

Datenbankschlüssel müssen einzigartig sein, zwei generierte UUIDs könnten aber identisch sein, auch wenn die Wahrscheinlichkeit dafür verschwindend gering ist. Zeitstempelbasierte UUIDs

[9] Webseite von MOFM2T [http://www.omg.org/spec/MOFM2T/]
[10] Vgl. [WEB.IETF.UUID]

sind in einem Intervall von 100 Nanosekunden einzigartig, man müsste also schon 10 Milliarden UUIDs pro Sekunde generieren, um Duplikate zu erzeugen[11]. Damit die Datenbank performant arbeiten kann, werden Indizes auf den Schlüsselspalten erzeugt. Damit der Index optimal angewandt werden kann, muss eine monotone Ordnung innerhalb der generierten UUIDs existieren. Das muss beim Generieren der UUIDs berücksichtigt werden.

Die Verwendung von generierten UUIDs ermöglicht einen großen Vorteil: Ein Umzug von einem Server auf einen Anderen oder das Austauschen der zugrundeliegenden Datenbanktechnologie wird erheblich erleichtert. Es reicht aus, die Daten in die neue Datenbank zu kopieren, alle Referenzen, die über UUIDs gesetzt sind, bleiben weiterhin gültig. Ein Umzug der Daten ohne UUIDs hätte mit Sicherheit zur Folge, dass alle Referenzen auf eventuell neu generierte numerische IDs angepasst werden müssten. Das ist ein komplexer und fehleranfälliger Vorgang, dessen Komplexität bei zunehmendem Datenbestand immer größer wird[12].

Es gibt verschiedene Typen von UUIDs: zeitstempelbasierte, pseudo-zufällige und namensbasierte UUIDs. CoRS-O-Mat verwendet zeitstempelbasierte UUIDs, da diese automatisch eine monotone Ordnung besitzen und dadurch gut durch die Datenbank indiziert werden können. Da diese Art auf der Systemzeit basiert, kann es problematisch sein, falls die Systemzeit zurückgesetzt wird oder im Fall der Zeitumstellung für Sommer- und Winterzeit. Dafür gibt es ein Feld innerhalb der UUID (clock sequence), welches in diesem Fall um eins erhöht oder auf einen zufälligen Zahlenwert gesetzt wird. Es ist weiterhin zu beachten, dass bei der Art der Verwendung von zeitstempelbasierten UUIDs erhebliche Performanzunterschiede existieren. Eine Laufzeituntersuchung, bei der die Datengröße und die Laufzeit von INSERT- und SELECT-Abfragen mit drei verschiedenen Repräsentationsarten von UUIDs gegen eine MySQL-Datenbank getestet wurden, hat aufgezeigt, wie eine laufzeitoptimale Repräsentation von UUIDs aussehen sollte[13]. Die getesteten Varianten inklusive der Datentypen sind wie folgt:

- UUID als binary(16) und PRIMARY KEY
- UUID als binary(16) und PRIMARY KEY mit INDEX und zusätzlich eine ID als BIGINT mit AUTO INCREMENT und PRIMARY KEY
- Eine umgestellte Variante der UUID als binary(16) und PRIMARY KEY

Untersucht wurden der Arbeitsspeicherverbrauch für die Daten und den Index und das Laufzeitverhalten der Abfragen für ca. 25 Millionen Inserts. Die umgestellte UUID aus Variante drei ist in allen Testfällen die performanteste und stellt den besten Schlüsselkandidaten für CoRS-O-Mat

[11] Vgl. [WEB.LIEVEN.DOCLO.UUID]
[12] Vgl. [WEB.TODD.FREDRICH.REST] und [WEB.LIEVEN.DOCLO.UUID]
[13] Vgl. [WEB.KARTHIK.APPIGATLA.UUID]

dar. Diese UUID-Art ist eine Standard-zeitstempelbasierte UUID mit dem Unterschied, dass die ersten drei Zahlenblöcke neu angeordnet werden. Die Gründe für diese Umstellung werden klar, wenn man die Zusammenstellung einer UUID näher betrachtet. Der Aufbau einer UUID ist nach RFC 4122 wie folgt[14]:

Feld-name	Datentyp	Oktet	Beschreibung
time_low	uint32	0-3	Zeitstempel, niederwertigstes Feld
time_mid	uint16	4-5	Zeitstempel, mittleres Feld
time_hi_and_version	uint16	6-7	Oberstes Feld des Zeitstempels zusammen mit der Versionsnummer
clock_seq_hi_and_reserved	uint8	8	Oberstes Feld der Clocksequenz zusammen mit der UUID-Variante
clock_seq_low	uint8	9	Unteres Feld der Clocksequenz
node	uint48	10-15	Eindeutige Node-Identifikationsnummer (z. B. basierend auf der MAC-Adresse des Netzwerkadapters)

Tabelle 4.1. Aufbau einer UUID nach RFC 4122

Anhand der Beispielhaften UUID 58e0a7d7–eebc–11d8-9669-0800200c9a66 soll das Verfahren der Umstellung demonstriert und erläutert werden. Die letzten zwei Blöcke (9669-0800200c9a66) werden überwiegend konstant sein, wenn die UUIDs auf demselben Server generiert werden, d. h. die ersten drei Blöcke (58e0a7d7–eebc–11d8) setzen sich aus dem aktuellen Zeitstempel zusammen. Das erste bit aus dem dritten Block (11d8) bezeichnet die Version der UUID, in diesem Fall zeitstempelbasiert und deshalb 1. Also wird der Zeitstempel beschrieben durch die Bits 58e0a7d7–eebc–1d8, wobei die niederwertigsten Bits, also die Bits mit der größten Werteänderung, ganz am Anfang stehen, was im Sinne einer monotonen Zahlenordnung nicht optimal ist. Dreht man diese Darstellung um, so erhält man eine optimal monoton geordnete Folge von Zahlen. Die umgestellte und von Bindestrichen bereinigte UUID sieht dann wie folgt aus: 11d8eebc58e0a7d796690800200c9a66.

Das Java-API stellt eine UUID-Klasse zur Erzeugung von UUIDs zur Verfügung, es können aber leider keine zeitstempelbasierten UUIDs generiert werden, was womöglich an einer unzu-

[14] Vgl. [WEB.IETF.UUID]

reichenden Genauigkeit des Java-Timers liegen könnte[15]. Zur Implementierung der benötigten Funktionalität kann aber auf externe Generatoren zurückgegriffen werden, wie z. B. JUG (Java UUID Generator) von FasterXML[16].

Die Komponente Storage arbeitet intern mit den umgestellten UUIDs, welche direkt in die Datenbank gespeichert und als Schlüssel zur Referenzierung der Entitäten verwendet werden.

4.5. Sicherheit

Die Sicherheit der Webanwendung CoRS-O-Mat ist ein wichtiges Thema im Softwareentwurf, denn es muss gewährleistet sein, dass die gespeicherten Daten weder einer unauthorisierten Manipulation noch einem unauthorisierten Zugriff ausgesetzt werden können. Die Sicherheitsmaßnahmen bei CoRS-O-Mat umfassen folgende Vorkehrungen:

- Verbindungen über HTTPS
  Pages forciert eine Kommunikation über eine verschlüsselte HTTPS-Verbindung, ungesicherte HTTP-Verbindungen werden nicht zugelassen und auf HTTPS umgeleitet.

- Input-Validierung
  Alle Daten, die in das System gelangen, müssen einer syntaktischen und semantischen Validierung unterzogen werden[17]. Bei der syntaktischen Validierung wird geprüft, ob der Typ der ankommenden Daten zum erwarteten Datentyp passt, d. h. ob es sich z. B. um eine valide E-Mail-Adresse handelt, bei der semantischen Validierung wird geprüft, ob ein syntaktisch korrekter Datensatz im Kontext sinnhaft ist, ob z. B. ein Geburtsdatum nicht in der Zukunft liegt.

- Output-Escaping
  Um zu verhindern, dass in die Daten eingeschleuste Kommandos durch den Browser ausgeführt werden, müssen alle Ausgaben in Escape-Sequencen übesetzt werden, damit Steuerzeichen durch Escapezeichen ersetzt werden und somit vom Browser ausschließlich angezeigt werden[18].

- Sichere Konfiguration aller eingesetzten Frameworks
  Alle eingesetzten Frameworks werden so konfiguriert, dass sie den Sicherheitsanforderungen entsprechen.

- Verwendung des Java Sicherheitsframeworks Apache Shiro[19]

[15] Vgl. [WEB.TATU.SALORANTA.JUG]
[16] Webseite von JUG [http://wiki.fasterxml.com/JugHome]
[17] Vgl. [SCHADOW.2014], S. 50
[18] Vgl. [SCHADOW.2014], S.62
[19] Webseite von Apache Shiro [http://shiro.apache.org]

Das Sicherheitsframework Shiro von Apache wird eingesetzt, um einerseits die Benutzerauthentifizierung mit JWT zu realisieren und um andererseits die Zugriffsberechtigungen zu verwalten und zu realisieren. Shiro lässt die Konfiguration von Benutzerrollen zu, ermöglicht aber darüber hinaus, Zugriffsberechtigungen mit sogenannten Permissions zu realisieren. Dadurch können Benutzerrechte fein granuliert werden, ohne dass für jede Variation eine neue Benutzerrolle erzeugt werden muss[20].

Im Folgenden werden weitere Entwurfsentscheidungen beschrieben, welche sicherheitsrelevante Aspekte adressieren.

4.5.1. Zugriffskontrolle

CoRS-O-Mat ist ein öffentliches Projekt und gestattet daher jedem Benutzer, der die Webseite aufruft, einen lesenden Zugriff auf die Daten der Software. Zum Anlegen, Bearbeiten oder Löschen von Daten muss sich der Benutzer mit Benutzername und Passwort authentifizieren.

4.5.1.1. Authentifizierung mit JSON Web Token (JWT)[21]

Die geplante zustandslose Architektur von CoRS-O-Mat erlaubt es nicht, traditionelle serverbasierte Authentifizierung zu verwenden, wie z. B. HTTP Basic Authentication oder HTTP Digest Access Authentication, da diese Verfahren darauf beruhen, dass die entsprechenden Informationen auf dem Server gespeichert sind und der Zustand in der aktuellen Session abgelegt wird. Üblicherweise erzeugt der Client dann einen lokalen Session-Cookie, der die entsprechende Zustandsinformation, ob der Benutzer eingeloggt ist oder nicht, bei jeder Anfrage mitschickt.

Probleme der serverbasierten Authentifizierung:

- Server-Sessions
  Der Server erzeugt für jeden eingeloggten Benutzer eine Session, üblicherweise im Hauptspeicher oder auch auf der Festplatte. Je mehr Nutzer sich anmelden, desto mehr Speicher wird verbraucht.

- Skalierbarkeit
  Wenn die Authentifizierungsinformationen auf einem Serverspeicher abgelegt sind, dann lässt sich die Anwendung schlechter skalieren, da die Information, ob ein Benutzer eingeloggt ist oder nicht, unter Umständen nicht auf einen anderen Server repliziert werden kann.

- Cross-Origin Resource Sharing (CORS)

[20] Vgl. [SCHADOW.2014], S. 232-233

[21] RFC7519: JSON Web Token (JWT) [https://tools.ietf.org/html/rfc7519]

Wenn die Applikation parallel auf mehreren Geräten genutzt wird, kann eine Anfrage abgelehnt werden, insbesondere bei der Nutzung der AJAX-Technologie, wenn sich ein berechtigter Nutzer in der Zwischenzeit mit einem anderen Gerät angemeldet hat. Die bereits laufende Applikation ist dann unter Umständen nicht mehr authentifiziert, weil die Anfrage jetzt von einer anderen Quelle abgesendet wird.

- Cross-Site Request Forgery (CSRF)
  CSRF beschreibt eine Angriffsmethode, die bei einer serverbasierten Authentifizierung möglich ist, bei der einem angemeldeten Benutzer eine authorisierte aber nicht gewünschte Anfrage ungewollt untergeschoben wird, z. B. per mail oder auf einer manipulierten Webseite. Ein solcher manipulierter URI könnte eine Anfrage absenden, die ausnutzt, dass der Benutzer z. B. gerade beim Online-Banking angemeldet ist.

Die genannten Probleme werden durch eine tokenbasierte Authentifizierung gelöst. Wenn ein Benutzer sich mit Benutzername und Passwort am System anmelden möchte, dann wird er durch die Applikation an einem Authentication Server validiert. Bei erfolgter Authorisierung generiert die Applikation ein signiertes Token und teilt dieses dem Client mit. Dieses Token muss bei jeder weiteren Anfrage mitgeschickt werden. Das Problem der Server-Sessions wird gelöst, da jeder Client ein gültiges Token haben muss, welches bei jeder Anfrage erneut validiert wird. Dadurch ist es nicht nötig, dass der Server Daten bezüglich der Anmeldung speichert, noch nicht einmal das Token selbst, denn die Validierung des Tokens erfolgt allein durch den Aufbau des Tokens. Auch das Problem der Skalierbarkeit ist gelöst, denn mit der Verwendung von Tokens zur Authentifizierung kann jeder beliebige Server die Gültigkeit der Anfrage ermitteln. Das Problem des Cross-Origin Resource Sharing (CORS) ist auch behoben, denn nun kann der Server so konfiguriert werden, dass er Anfragen von verschiedenen Quellen zulässt mittels des Serverparameters `Access-Control-Allow-Origin: *`. Bemerkenswert ist, dass wenn diese Direktive gesetzt ist, der Server automatisch keine HTTP-Authentifizierungsinformationen und auch keine Cookies mehr akzeptiert. Das letzte Problem des CSRF-Angriffs wird dadurch gelöst, dass die abzusendende Seite im Besitz eines gültigen Tokens sein muss und sich nicht mehr den Umstand zunutze machen kann, dass der Server gespeichert hat, ob der Benutzer angemeldet ist oder nicht.

Es gibt verschiedene Ansätze, das Token im Client zu speichern, üblicherweise im browsereigenen LocalStorage oder SessionStorage mit dem Zugriff über JavaScript oder in einem Cookie. Das Cookie wird in diesem Fall ausschließlich zur lokalen Speicherung des Tokens verwendet und nicht zur Authentifizierung am Server. Die erste Variante erfordert einen HTML5-kompatiblen Browser und eine aktivierte JavaScript Laufzeitumgebung. Die Variante der Speicherung im Cookie erfordert, dass der Benutzer Cookies aktiviert hat. Bei CoRS-O-Mat wird das Authentifizierungstoken in einem Cookie gespeichert und serverseitig ausgelesen und validiert. Auf diese Weise ist die Authentifizierung bei CoRS-O-Mat unabhängig vom Vorhandensein von JavaScript beim Benutzer.

Die folgende Abbildung zeigt die Authentifizierung mit JWT als Sequenzdiagramm. Die Komponente Realm übernimmt dabei die Funktion als Authentifizierungsserver. Die Antwort auf eine erfolgreiche Authentifizierung wird mit Hilfe der POST-redirect-GET-Methode generiert[22]

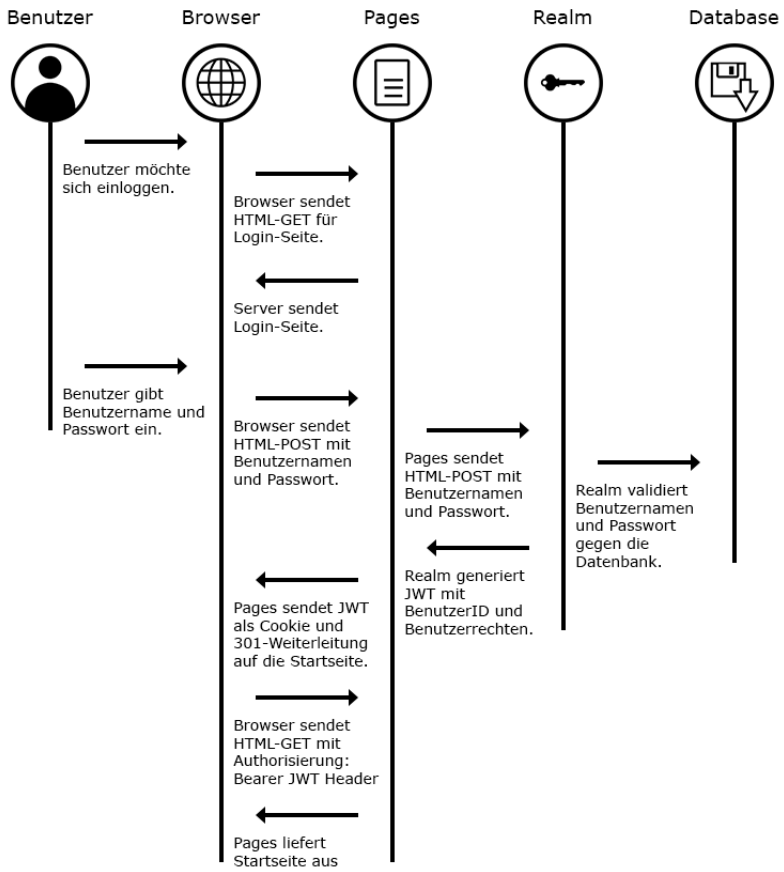

Abbildung 4.3. Authentifizierung mit JSON Web Token (JWT)

Als Standard kommen bei CoRS-O-Mat JWTs zum Einsatz, da JWT im Internet bereits eine breite Verwendung gefunden hat als universelles Tokenformat. Es ist von der IETF als RFC 7519 spezifiziert. Ein JWT ist aus drei Teilen aufgebaut, einem Header, einer Payload und einer Signatur. Der Header enthält Informationen zum Typ des Tokens und zum verwendeten Hashing-Algorithmus, z. B. `{type: 'JWT', alg: 'HS256'}` und wird Base64-enkodiert. Die Payload enthält die Daten, die die Zugriffsinformationen des Benutzers beschreiben. Diese Daten können das Token auch als ein spezielles Token auszeichnen mit speziellen Berechtigungen, die daran geknüpft sein können. Die Payload wird Base64-enkodiert. Die Signatur entsteht unter Zuhilfenahme des definierten Algorithmus aus dem Hashwert des kodierten Headers, der kodierten Payload und eines geheimen Schlüssels. JWT-Authentifizierung und -Authorisierung wird in

[22]Vgl. Abschnitt 4.6.3, „RESTful JSF mit der POST-redirect-GET-Methode“

CoRS-O-Mat unter Zuhilfenahme des Open Source Frameworks Shiro von der Apache Foundation implementiert.

Mit JWT ist es möglich, dass ein Benutzer sich über einen Token einer anderen Anwendung via OAuth2[23] gegenüber CoRS-O-Mat authentifiziert. Es ist z. B. denkbar, dass Benutzer ihren Google- oder Facebook-Account zur Anmeldung nutzen. In dem Fall wird das Token von der jeweiligen Anwendung an CoRS-O-Mat übertragen. Dies ist grundsätzlich mit jedem Server möglich, der die Authentifizierung über Webtokens unterstützt. Diese Funktionalität soll im Rahmen dieser Arbeit nicht implementiert werden, aber durch die Verwendung von JWT zur Authentifizierung soll die spätere Implementierung optimal vorbereitet werden.

4.5.1.2. Benutzerrollen

Die Authorisierung, also die Ermittlung und Anwendung der Berechtigungen eines Benutzers erfolgt ausschließlich aufgrund der definierten Benutzerrollen. Berechtigungen auf Benutzerebene sind nicht vorgesehen. Die folgenden Annahmen liegen der Definition der Benutzerrollen zugrunde:

- Eine Benutzerrolle ist eine Menge von Berechtigungen.
- Jeder Benutzer kann einer oder mehreren Benutzerrollen zugeordnet werden.
- Die Rollen sind disjunkt, d. h. sie haben keine inhaltlichen Überschneidungen.

4.5.1.3. Beschreibung der Benutzerrollen

- Öffentlicher Benutzer
  Ein öffentlicher Benutzer hat Zugriff auf die öffentlichen Daten des Systems über die Webadresse des Projekts.
  Er kann die Daten nur lesen, nicht jedoch Daten in das System eingeben.
  Er benötigt keine Anmeldung.
- Registrierter Benutzer
  Ein registrierter Benutzer hat ein Benutzerkonto im System und kann sich mit seinem persönlichen Passwort anmelden.
  Er kann neue Datensätze anlegen und nur die von ihm angelegten Daten bearbeiten, annotieren oder löschen.
- Datenerfasser
  Ein Datenerfasser kann sich am System anmelden und er kann alle Datensätze bearbeiten und neue anlegen.

[23] Webseite der IETF zum RFC 6749 (OAuth2) [http://tools.ietf.org/html/rfc6749]

- Benutzerverwalter
  Ein Benutzerverwalter kann neue Benutzerkonten anlegen und Benutzerkonten bearbeiten.
  Er kann Benutzerkonten neue Passwörter zuweisen.
  Er kann den Benutzerkonten Benutzerrollen zuweisen.

- Systemverwalter
  Ein Systemverwalter kann technische Parameter des Systems bearbeiten.

4.6. Das Zusammenspiel der verwendeten Frameworks

Bei CoRS-O-Mat kommen verschiedene Frameworks zum Einsatz, deren gleichzeitige Verwendung unter bestimmten Voraussetzungen zu Problemen führen kann. In diesem Abschnitt wird erläutert, welche Maßnahmen und Entscheidungen getroffen wurden, um ein unproblematisches Zusammenspiel der verwendeten Frameworks zu ermöglichen.

Im ersten Teil wird der mögliche Konflikt zwischen JSF und jQuery mobile beschrieben. Wenn JSF in Verbindung mit der AJAX-Technologie verwendet wird, dann kann das zu einen Konflikt mit dem clientseitigen JavaScript-Framework jQuery mobile führen, denn in diesem Fall würden beide Frameworks versuchen, den Request zu bearbeiten. In dem darauf folgenden Abschnitt wird erläutert, ob Java ManagedBeans, Context & Dependency Injection Beans (CDI Beans) oder Enterprise Java Beans (EJB) zum Einsatz kommen und wie sich diese Spezifikationen unterscheiden. Abschließend wird beschrieben, was RESTful JSF bedeutet und wie es unter Einsatz der beschriebenen Frameworks umgesetzt werden kann.

4.6.1. JSF und jQuery mobile

JSF ist als Standard-Benutzeroberfläche für webbasierte Java EE Anwendungssoftware festgelegt. Als Implementierung kommt bei CoRS-O-Mat Oracles Referenzimplementierung Mojarra zum Einsatz. Seit Version 2.0 ist AJAX (Asynchronous JavaScript and XML) Bestandteil der Spezifikation. JQuery mobile ist ein Touch-optimiertes JavaScript-Framework mit dem Ziel, webbasierte Anwendungssoftware auf dem stetig wachsenden Markt der Smartphones und Tablet-PCs ohne große manuelle Aufwände lauffähig zu machen. Entwickelt wird es von der jQuery Foundation. Sowohl Mojarra, als auch jQuery mobile nutzen JavaScript, um die Antwort vom Server zu verarbeiten und umzusetzen. Bei der gleichzeitigen Benutzung beider Frameworks gibt es Einschränkungen, die bei der Implementierung beachtet werden mussten. In den folgenden Abschnitten werden die relevanten Aspekte beider Frameworks dargestellt und danach wird das Problem und der gewählte Lösungsansatz skizziert.

Mit JSF hat der Entwickler unter anderem die Möglichkeit, Teile der Webseite mittels der AJAX-Technologie durch den Browser rendern zu lassen, ohne die gesamte Webseite aktualisieren zu müssen. Dafür gibt es ein spezielles Tag (`<f:ajax render=':idEinesElements' />`), in

welchem man die ID der Elemente definieren kann, welche als einziges neu gerendert werden sollen, nachdem eine bestimmte Aktion ausgelöst wurde (z. B. ein Klick auf einen Link). Dabei sendet das Element, welches die Aktion auslöst (z. B. ein Link oder auch ein Timer) eine asynchrone Anfrage an den Server, der den entsprechenden HTML-Code erzeugt und zurückgibt. Die Antwort wird von einer JavaScript-Funktion ausgewertet und direkt in den Dokumentenbaum (DOM) eingebaut, dabei werden die alten Objekte gelöscht, bzw. überschrieben. Der Vorteil dieser Technologie besteht darin, dass kein kompletter Seitenaufbau stattfinden muss und eben nur bestimmte Teile der Seite aktualisiert werden können. Dadurch verhält sich die angezeigte Webseite eher wie eine Desktopanwendung und sie wird für den Benutzer leichter zugänglich im Gegensatz zu einer Webseite im klassischen Sinn, bei der immer eine ganze Seite ausgetauscht wird und damit dem Benutzer keine kontinuierliche Erfahrung geboten werden kann.

JQuery mobile ist in der Lage, die Webseite und deren Elemente im Sinne des Responsive Web Designs so anzuordnen, dass die Bedienung je nach Geräteklasse optimiert wird. Um eine jQuery mobile konforme Webseite zu bauen, müssen die verwendeten HTML-Tags mit jQuery mobile-spezifischen Attributen angereichert werden (z. B. `"data-role='page'"` oder `"data-role='button'"`). Das Framework ist mit den zusätzlichen Informationen in der Lage, jedem Element seine Rolle und sein Aussehen innerhalb der Webseite zuzuordnen. JSF verhinderte bisher die Definition von Attributen, die nicht in der JSF-Spezifikation definiert wurden, damit jQuery mobile dennoch mit der verwendeten JSF-Implemetierung funktioniert, muss mindestens Version 2.2 der JSF-Spezifikation verwendet werden, denn in Version 2.2 wurden die sogenannten Passthrough-Attribute[24] definiert, welche vom Server einfach durchgereicht werden. So können die benötigten spezifischen Attribute in die Seite gerendert werden, was dazu führt, dass jQuery mobile korrekt funktionieren kann. Der Browser muss auch in der Lage sein, eine Webseite im spezifizierten HTML5-Standard anzuzeigen, was alle aktuell am Markt verfügbaren Browser können. Darüber hinaus ist jQuery mobile aber auch kompatibel zu älteren Browsern[25].

Problematisch wird die Benutzung beider Frameworks einerseits bei der Seitennavigation mittels jQeury mobile und andererseits beim asynchronen Aktualisieren bestimmter Teilbereiche der Seite mittels JSF-AJAX. JQuery mobile übernimmt üblicherweise die Navigation zu anderen Webseiten, lädt diese bei Anforderung asynchron vor und stellt diese dann dar, wenn sie komplett geladen wurden. Dadurch ist es unter anderem möglich, aus einem Repertoire von Übergangsanimationen einen animierten Seitenübergang zu definieren. Wenn Seiteninhalte zwischendurch auch durch JSF-AJAX partiell aktualisiert werden, dann kann jQuery mobile den aktualisierten Elementen ihre definierte Rolle nicht mehr zuweisen und die angezeigte Webseite wäre fehlerhaft. Um dem vorzubeugen, muss auf eine Seitennavigation durch JSF-AJAX

[24] API-Dokumentation zu Passthrough-Attributen [http://docs.oracle.com/javaee/7/javaserver-faces-2-2/vdldocs-facelets/p/tld-summary.html]

[25] Vgl. Browserunterstützung von jQuery mobile [https://jquerymobile.com/browser-support/1.4/]

verzichtet werden, um an dieser Stelle jQuery die Ausführung zu ermöglichen. Das erreicht man durch Verwendung JSF-spezifischer Komponenten, welche die Webseite ohne AJAX darstellen. Es sollte also beachtet werden, folgende JSF-Tags nicht zu verwenden: `<f:ajax>`, `<h:commandLink>` und `<h:button>`, da diese Komponenten auf JavaScript-Funktionalität basieren. Beispielsweise wird ein in JSF notierter commandLink, `<h:commandLink action="#{clauseControl.save()}" />`, wie folgt in eine HTML-Seite gerendert: `<a href="#" onclick="mojarra.jsfcljs(document.getElementById('form-clauses-index'),{'form-clauses-index:j_idt22':'form-clauses-index:j_idt22'},'');return false" class="ui-link">Save</a>`. Ein Klick auf diesen Link wird durch den definierten onclick-Handler verarbeitet, die reguläre Linkfunktionalität wird abgeschaltet. Im folgenden Beispiel ist ein commandButton dargestellt, die Passthrough-Attribute sind mit einem vorangestellten „p:“ gekennzeichnet: `<h:commandButton p:data-role="button" p:data-icon="minus" value="Save" action="#{clauseControl.save()}" >`. Dieser wird folgendermaßen in die Webseite gerendert: `<input type="submit" name="form-clauses-index:j_idt22" value="Save">`. Dieser Button schickt das Formular regulär ab ohne Einwirkung von JavaScript.

Weiterhin muss beachtet werden, dass die Daten des umschließenden Formulars nicht asynchron durch jQuery mobile abgesendet werden. Das erreicht man durch die Annotation `p:data-ajax="false"` im Formularelement.

Dadurch bleibt eine dauerhafte Verbindung zur darunterliegenden Java-Bean-Klasse bestehen und die Session des Servers ist immer synchron zum Zustand der Benutzersicht.

Es tritt bei der partiellen Aktualisierung durch JSF ein weiteres Problem auf, bei dem aktualisierte Komponenten ihr von jQuery mobile zugewiesenes Aussehen verlieren und auf das systemspezifische Standardaussehen und -layout zurückfallen. Das ist ein bekanntes Problem und wurde durch die Entwickler des jQuery mobile Frameworks bereits adressiert[26]. Es kommt dadurch zustande, dass jQuery mobile initial, nachdem die Seite vollständig geladen wurde, allen Elementen ihr Aussehen und Layout basierend auf der aktuellen Größe des Fensters zuweist. Wenn nun zwischendurch Elemente ausgetauscht werden, greift an der Stelle dieser Mechanismus nicht mehr. Um aktualisierte Elemente wieder korrekt auf der Webseite darstellen zu lassen, gibt es eine Funktion, die nach einem erfolgten AJAX-Request auf den entsprechenden Elementen aufgerufen werden muss: `$(".elements").enhanceWithin();`. Das Ausführen dieser Funktion garantiert, dass alle Elemente entsprechend der Richtlinien des Responsive Web Designs nach den CSS-Definitionen von jQuery mobile dargestellt werden. Bei CoRS-O-Mat wird dieses Problem umgangen, indem auf den Einsatz von JSF mit partiellem Rendern der Seite gänzlich

[26] API-Dokumentation zur enhanceWithin()-Funktion [https://api.jquerymobile.com/enhancewithin/]

verzichtet wird. Dadurch wird auch eine Unabhängigkeit von JavaScript für die Kernfunktionalität der Anwendung erreicht.

4.6.2. ManagedBeans, CDI Beans und EJBs

Es gibt verschiedene Ausprägungen von Bean Mechanismen in Java, JSF ManagedBeans, CDI Beans und EJBs, deren Funktionalität sich auf den ersten Blick kaum unterscheidet. Es soll hier ein Versuch unternommen werden, die Unterschiede aufzuzeigen und darzulegen, welche Arten von Beans bei CoRS-O-Mat verwendet werden und warum.

JSF wurde initial mit einem eigenen einfachen Bean Management und Dependency Injection Mechanismus implementiert, den ManagedBeans. Diese haben einen geringeren Funktionsumfang als CDI Beans und EJBs. ManagedBeans sollten generell eher nicht verwendet werden, wenn man eine Java EE-Anwendung implementiert, hier sollte man auf die vielseitigeren und als Standard definierten CDI Beans zurückgreifen. Wenn die Funktionen von EJBs benötigt werden, die nicht von CDI Beans abgedeckt werden, sollte man diese allein oder zusammen mit CDI Beans verwenden. EJBs unterscheiden sich von CDI Beans hauptsächlich in folgenden Punkten: Remoting und Security[27]. Remoting wird verwendet, um Aufrufe auf EJBs über verschiedene JVMs hinweg mittels Remote Method Invocation (RMI) abzusenden. Dabei können die EJBs aufgerufen werden, ohne über Web Services oder REST lose gekoppelt zu sein. Mit Security ist hier die Möglichkeit der Annotation mit `RolesAllowed` gemeint, mit der sich ein Authorisierungsmechanismus bequem implementieren lässt. Die Authorisierungsprüfung übernimmt in diesem Fall der Container. Für CoRS-O-Mat werden CDI Beans und EJBs verwendet.

4.6.3. RESTful JSF mit der POST-redirect-GET-Methode

Bei der Implementierung von CoRS-O-Mat wurde besonderen Wert darauf gelegt, die wichtigen Aspekte eines RESTful JSF zu implementieren. Folgende Anforderungen sollen umgesetzt werden:

- Es soll dem Benutzer zu jeder Zeit möglich sein, den Browser zu aktualisieren und damit eine Aktualisierung der gerade angezeigten Webseite zu bewirken. Dabei soll verhindert werden, dass der Browser versucht, ein bereits abgeschicktes Formular erneut abzusenden.

- Die Browser-Funktionalitäten Zurück und Vorwärts sollen genau so funktionieren, wie der Benutzer es auch erwartet. D. h. mit einem Druck auf den Zurück-Button des Browsers soll innerhalb der Browserhistorie genau einen Schritt zurück gesprungen werden, bei Vorwärts dementsprechend einen Schritt nach vorne.

- Es soll dem Benutzer ermöglicht werden, Lesezeichen anzulegen und zu verwenden.

[27] Vgl. [WEB.LIMBURG.RÖWEKAMP.BEANS] und [WEB.GIBSON.BEANS]

- Die URLs der Anwendung sollen in einem lesbaren Format in der Adressleiste des Browsers erscheinen. Also anstelle eines beispielhaften URLs der Form `www.corsomat.de/users/show.xhtml?id=12345678901234` soll an dieser Stelle ein URL der Form `https://www.corsomat.de/users/max-mustermann` angezeigt werden.

Was bedeutet nun RESTful JSF in diesem Zusammenhang, was sind die Vorteile und wie wurde es implementiert?

REST (Representational State Transfer) ist ein Softwarearchitekturmodell für verteilte Systeme, insbesondere für Webservices, bei dem Ressourcen die entscheidende Rolle spielen. Ressourcen sind in einer solchen Webanwendung wie CoRS-O-Mat z. B. eine Liste aller registrierten Benutzer oder die Informationsseite zu einem bestimmten Benutzer. Ressourcen müssen folgende Anforderungen erfüllen[28]:

- Adressierbarkeit.
  Die Resource ist eindeutig durch einen URI adressierbar, wie z. B. ein bestimmter Benutzer durch die Adresse `https://www.corsomat.de/users/max-mustermann` oder die Liste aller Benutzer durch `https://www.corsomat.de/users/`. Das Erzeugen der RESTful URIs wird in der Komponente Pages durchgeführt und mit dem Open-Source Framework PrettyFaces[29] realisiert, so dass aus Benutzersicht lesbare URIs existieren, systemintern aber Ressourcen weiterhin über ihre UUIDs referenziert werden.

- Zustandslosigkeit
  Alle nötigen Information müssen bei jeder Anfrage mitgesendet werden, da die Kommunikation zustandslos ist, d. h. die Ausführung der Anfrage basiert nicht auf einem Sitzungsspeicher oder einem ähnlichen, zustandsbehafteten Mechanismus, sondern allein auf der eingegangenen Anfrage. Dadurch ist das System optimal skalierbar, da Anfragen bei Bedarf einfach auf verschiedene Server verteilt werden können. Dadurch wird auch gewährleistet, dass die Vor- und Zurück-Funktionen des Browsers das erwartete Ergebnis zurückliefern. Das ist bei üblicher sessionbasierter AJAX-Anwendungssoftware meist nicht der Fall, wenn dort der aktuelle Zustand auf den vorangegangenen Aktionen basiert. Ein Aktualisieren der Seite führt in diesem Fall zum Zurückspringen an den Anfang der Anwendung, weil die erfolgten Benutzeraktionen asynchrone AJAX-Ausführungen anstoßen, welche nicht über einen URI nach außen, also auch für den Browser, sichtbar gemacht werden.

- einheitliche Operationen und POST-redirect-GET
  REST wird im Webumfeld in der Regel mit HTTP und HTTPS verwendet, ist aber nicht an ein spezifisches Protokoll gebunden. Durch HTTP ergeben sich die Standardoperationen GET

[28] Vgl. [BURKE.2013], S.5-13

[29] Webseite von PrettyFaces [http://www.ocpsoft.org/prettyfaces/]

zum Lesen einer Resource, POST zum Erzeugen einer neuen Resource, PUT zum Ändern einer bekannten Resource oder zum Erzeugen, falls sie nicht existiert und DELETE zum Löschen einer Resource[30]. Dabei wird verlangt, dass eine mehrfache Ausführung einer Anfrage für GET, PUT oder DELETE sich nicht anders auswirkt, als eine einzelne. In einer Webanwendung sollte insbesondere verhindert werden, dass Formulare durch drücken der Aktualisierungstaste mit den selben Daten erneut abgesendet werden können. Das kann erreicht werden durch Anwendung der Methode POST-redirect-GET, welche definiert, dass die Antwort auf ein abgesendetes Formular nicht die resultierende Webseite ist, sondern eine 301-Weiterleitung dorthin[31]. Dadurch wird eine weitere GET-Anfrage erzwungen und das stellt sicher, dass eine Aktualisierung der Webseite nicht ein erneutes Absenden des Formulars zur Folge hat.

- Ressourcen haben unterschiedliche Repräsentationen
  Ressourcen sind von der Darstellung entkoppelt und können in verschiedenen Formaten ausgeliefert werden, wie z. B. HTML, JSON oder XML.

Durch die Implementierung von RESTful JSF ergeben sich die folgenden Vorteile: Die Komponente Pages ist gut skalierbar auf verteilten Systemen, da auf dem Server keine spezifische Session existieren muss. Weiterhin wird die Benutzerfreundlichkeit gesteigert durch die Verwendung von benutzerfreundlichen URIs, die auch als Lesezeichen gespeichert werden können. Die Nutzung der browsereigenen Vor- und Zurück-Funktionalität funktioniert für den Benutzer, wie er es erwarten würde. Er kann auch zu jedem Zeitpunkt die gerade angezeigte Webseite aktualisieren, ohne dass dies unerwartete Folgen hätte, wie z. B. ein erneutes absenden eines Formulars.

Es soll hier skizziert werden, wie die verschiedenen Aspekte von RESTful JSF mit POST-redirect-GET implementiert wurden. Die Features von REST wurden unter Zuhilfenahme von Stateless EJBs und den Annotationen von JAX-RS implementiert. Die Webseiten der Komponente Pages müssen Parameter, wie z. B. IDs von Objekten per GET-Parameter an aufzurufende Seiten übergeben, da keine Serversession existiert, welche diese Werte vorhalten könnte. JSF definiert dafür das Element `<f:viewParam name="paramName" value="#{bean.paramValue}" />`, welches den Wert eines GET-Parameters mit dem Namen `paramName` einem Wert `paramValue` in der Bean-Klasse zuordnet. POST-redirect-GET lässt sich mit JSF einfach verwirklichen, dazu gibt es den Parameter `faces-redirect=true`, der als GET-Parameter an einen Navigations-URI angehängt werden muss. Das weist JSF an, das Navigationsziel nicht auszuliefern, sondern eine Weiterleitung dorthin. Dadurch wird auch die Adressleiste des Browsers mit der richtigen Adresse aktualisiert.

[30] Vgl. [WEB.FIELDING.ET.AL.HTTP]
[31] Vgl. [WEB.HOFSTETTER.RESTFUL]

Kapitel 5. CoRS-O-Mat Implementierung

Dieses Kapitel verschafft Einblick in die konkrete Implementierung von CoRS-O-Mat und die dabei verwendeten Werkzeuge bei der Entwicklung.

5.1. Software und Entwicklungstools

5.1.1. Integrierte Entwicklungsumgebung (IDE) und Projektsetup

Als integrierte Entwicklungsumgebung kommt bei der Implementierung von CoRS-O-Mat Eclipse Mars für Java EE zum Einsatz. Der modulare Aufbau der IDE und die tiefe Integration von Maven sind die Hauptgründe für die Verwendung von Eclipse. Es wurde zudem ein Git-Repository zur Versionierung des Quellcodes eingerichtet.

Alle Komponenten des Systems sind als Maven-Projekte realisiert, zusätzlich gibt es ein übergeordnetes Projekt, ein sogenanntes Maven Parent-Projekt, das gemeinsame Abhängigkeiten bündelt und dadurch die Projektwartung vereinfacht. Ein weiteres Projekt sorgt unter Verwendung von Liquibase dafür, dass die benötigte Datenbank unter Anwendung des definierten Schemas konstruiert wird. Die folgende Auflistung soll die Projektstruktur veranschaulichen:

- de.binfort.corsomat.maven
  Maven ist das übergeordnete Maven Parent-Projekt. Es löst gemeinsame Abhängigkeiten auf und verteilt diese an alle erbenden Projekte.

- de.binfort.corsomat.cors.maven
  CoRS.Maven ist das Maven Parent-Projekt für die CoRS-Domäne, definiert die Projekte Pages, Messages und Storage als erbende Module.

- de.binfort.corsomat.cors.pages
  Pages erzeugt die Benutzeroberfläche und führt die Benutzereingaben aus. Pages wird als WAR-Datei auf einem Server veröffentlicht.

- de.binfort.corsomat.cors.messages
  Messages stellt Datenbankabstraktionsobjekte als Austauschformat zur Verfügung. Sie wird als JAR-Datei in den Komponenten Pages und Storage eingebunden.

- de.binfort.corsomat.cors.storage
  Storage führt das objektrelationale Mapping zwischen Datenbank und Entitys aus und sorgt für die Persistenz der Daten. Storage wird als WAR-Datei auf einem Server veröffentlicht.

- de.binfort.corsomat.cors.protection
  Protection ist die Authentifizierungskomponente, die die Aufgaben übernimmt, Web-Tokens zu validieren, Tokens zu erzeugen und bei Bedarf auch Tokens von Webdiensten von Drittanbietern entgegenzunehmen. Sie wird als WAR-Datei auf einem Server veröffentlicht.

- de.binfort.corsomat.cors.database
  Dieses Projekt beinhaltet das Liquibase-Framework. Es initialisiert die Datenbank nach dem konfigurierten Schema und befüllt sie mit initialen Daten. Auch Backups einer laufenden Datenbankinstanz können mit Liquibase gemacht und eingespielt werden. Ein Umzug der Datenbank wird insofern dadurch erheblich vereinfacht, als dass die Anweisungen für Liquibase in einer universellen XML-Auszeichnungssprache formuliert werden, die für alle unterstützten Datenbanken identisch ist.

- de.binfort.corsomat.cors.documentation.uml
  Dieses Projekt enthält das definierte Datenmodell als UML, welches mit Papyrus erzeugt wurde.

- de.binfort.corsomat.cors.documentation.docbook
  Dieses Projekt ist der vorgesehene Ort für die geplante Softwaredokumentation. Sie wird unter Zuhilfenahme von DocBook erzeugt.

Jedes Projekt kann zusätzlich eigene Abhängigkeiten definieren, welche dann durch Maven aufgelöst und in das Zielverzeichnis kopiert werden. Das Buildmanagementwerkzeug Maven vereinfacht das Bauen der Anwendung erheblich. Sollte sich z. B. eine der Komponenten ändern, so sind die anderen automatisch auf dem aktuellen Stand. Ein manuelles kopieren von JAR-Dateien, wie in herkömmlichen Java-Projekten üblich, ist hier nicht nötig. Abhängigkeiten zu anderen Laufzeitbibliotheken, üblicherweise JAR-Dateien, werden automatisch aufgelöst und nach dem Herunterladen in die Anwendung eingepackt. Auch hier entfällt ein manuelles Kopieren oder die manuelle Konfiguration des Build-Path-Parameters in den Projekteinstellungen.

5.1.2. Kontinuierliche Lieferung

Um den Software-Build-Prozess weiter zu vereinfachen und zu automatisieren, wird CoRS-O-Mat nach dem Prinzip der Kontinuierlichen Lieferung gefertigt. Der dafür benögtigte Server ist ein Eclipse Hudson Server für kontinuierliche Integration. Damit sind die beteiligten Entwickler in der Lage, durch einen Commit in das Versionierungssystem Git und einen Push zum Server, einen Build anzustoßen, der dann von Hudson durchgeführt wird. Wenn der Build erfolgreich war und alle definierten Software-Tests bestanden wurden, dann liegt als Ergebnis eine neue Version der Software bereit.

5.1.3. Projektmanagement

Für das Projekt- und Änderungsmanagement wird die freie Software Redmine eingesetzt. Sie enthält u. a. eine Benutzer- und Rollenverwaltung, ein Ticketsystem mit Arbeitszeiterfassung, ein Wiki für die Projektdokumentation und eine Darstellung der Aktivitäten im Git-Repository mit der Möglichkeit, den Quellcode einzusehen.

5.1.4. Projektdurchführung mit Scrum

Nach Festlegung der Scrum-Benutzerrollen Product Owner, Scrum Master und Entwicklungsteam wurden die Anforderungen an die Software vom Product Owner als User Stories formuliert und in das Wiki von Redmine eingetragen. Die Ideen dafür wurden von allen beteiligten Prozessteilnehmern gemeinsam entwickelt. Die folgende Abbildung zeigt einen Auszug aus den Scrum User Stories im Redmine Wiki.

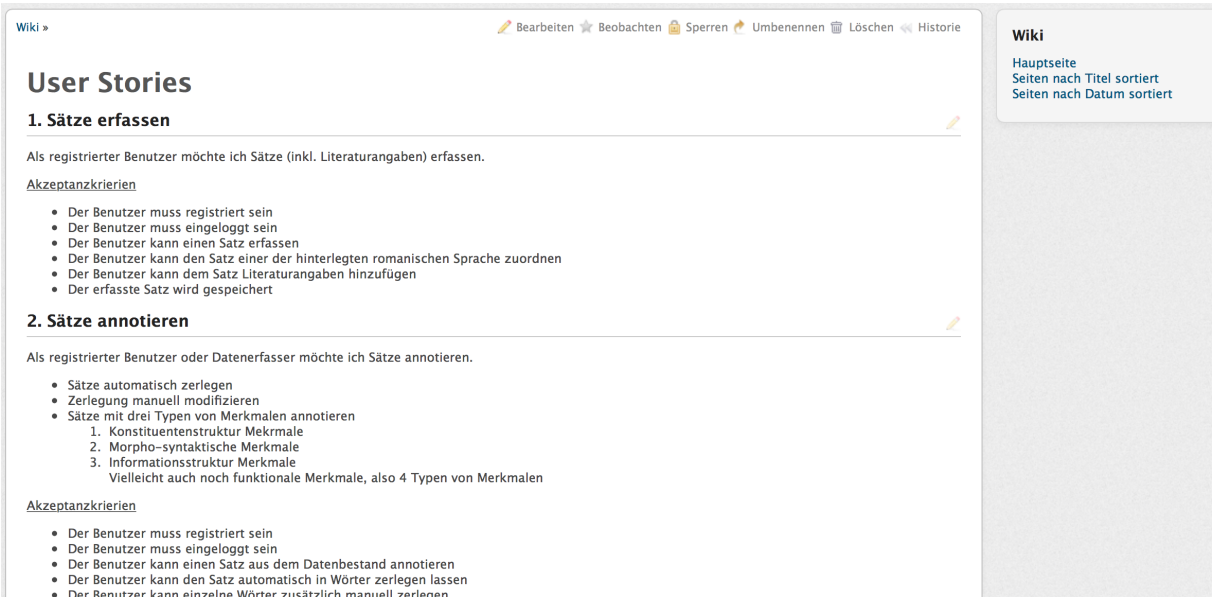

Abbildung 5.1. Scrum User Stories im Redmine Wiki

Im nächsten Schritt wurden die User Stories in einzelne Arbeitspakete zerlegt, priorisiert und in das Ticketsystem von Redmine überführt. Diese Tickets bilden das Product-Backlog nach Scrum. Die folgende Abbildung zeigt das Product-Backlog im Redmine Ticketsystem.

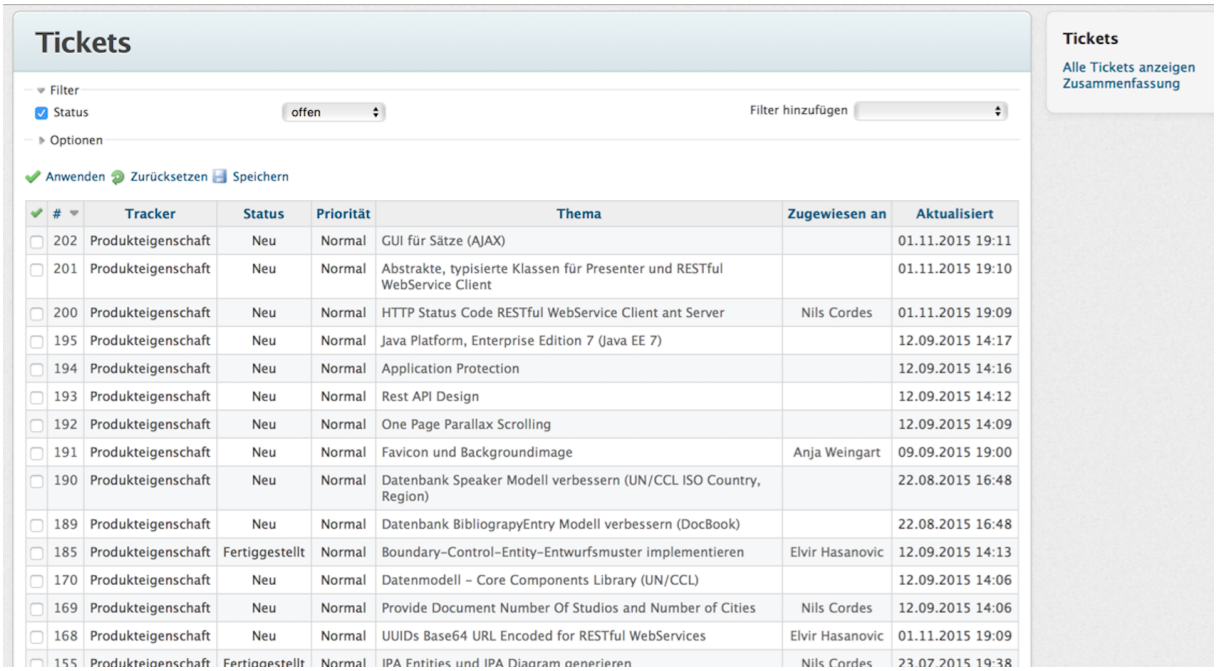

Abbildung 5.2. Scrum Product-Backlog in Redmine

Hoch priorisierte Arbeitspakete werden dem Backlog entnommen und in einem Sprint umgesetzt. Dabei werden die Arbeitspakete weiter in Tasks zerlegt und jeweils einem Bearbeiter zugewiesen. Daily Scrum Meetings wurden zweimal pro Woche durchgeführt. Die Abbildung zeigt den Abschnitt Roadmap von Redmine. In dieser Ansicht werden die offenen und geschlossenen Versionen angezeigt. Jeder Version sind Tickets zugeordnet, die vollständig bearbeitet werden müssen, damit die Version als geschlossen markiert werden kann. Eine Version in Redmine ist einem Sprint nach Scrum gleichzusetzen. Das Ergebnis einer Version ist das jeweilige Imkrement mit der dazugehörenden Versionsnummer.

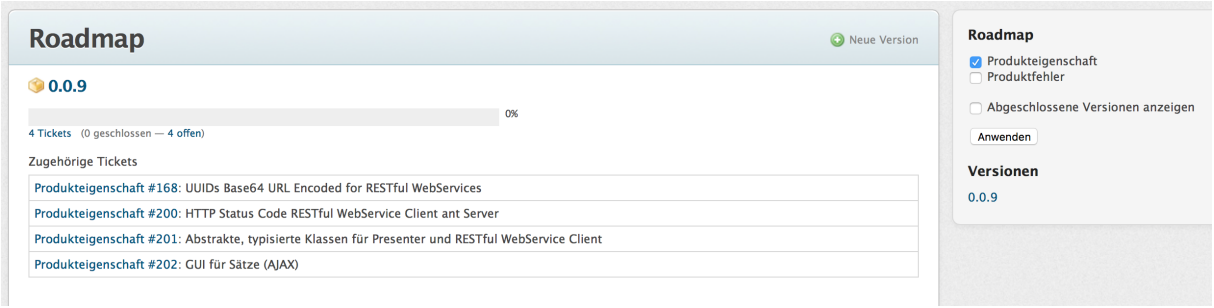

Abbildung 5.3. Scrum-Sprint in Redmine

Einige der durchgeführten Aufgaben, wie beispielsweise die Erstellung des Datenmodells der Kundendomäne wurden in enger Zusammenarbeit des Entwicklungsteams mit dem Product Owner umgesetzt.

5.1.5. Datenbank

Als lokale Entwicklungsdatenbank wird eine Apache Derby Datenbank verwendet. Das Produktivsystem wird mit der freien Java-basierten Datenbank H2 Database Engine (H2DB)[1] arbeiten. H2DB wurde für CoRS-O-Mat ausgewählt, weil sie gegenüber anderen relationalen Datenbanken mehrere Vorteile bietet, die hier aufgezählt werden sollen. H2DB unterstützt Clustering. Das bedeutet, dass mehrere Instanzen der Datenbank auf verschiedenen Servern gleichzeitig operieren können[2]. Das hat positive Auswirkungen sowohl auf die Performanz, denn die verschiedenen Instanzen können Daten parallel ausliefern als auch auf die Ausfallsicherheit der Datenbank, denn falls eine Instanz ausfällt, wird das durch das Vorhandensein anderer Instanzen kompensiert. H2DB kann Volltextindizierung, d. h. bei der Implementierung einer Volltextsuche über einem String-Datenfeld kann man die Suche performant innerhalb der Datenbank stattfinden lassen[3]. H2DB basiert vollständig auf Java, man benötigt zur Ausführung lediglich eine JVM. H2DB ist eine der performantesten relationalen Datenbanken für Java-basierte Anwendungen[4]

5.2. Implementierung

In den folgenden Abschnitten werden Details der Implementierung von CoRS-O-Mat dargestellt. Aufgrund der Fülle des Programmcodes wird an dieser Stelle auf ein vollständiges Programmlisting verzichtet. Das Projekt CoRS-O-Mat kann im Projektarchiv von SourceForge[5] eingesehen und heruntergeladen werden.

5.2.1. Vermeidung von Coderedundanz

Bei CoRS-O-Mat soll es möglichst keine Coderedundanzen geben. Auf der Java-Seite wird das erreicht durch Kapselung und Vererbung, wie u. a. im folgenden Beispiel der Klasse `AbstractPersistenceControl<EntityType>` gezeigt wird. Auf der XHTML-Seite wird es erreicht durch die Verwendung von Templates, die gemeinsam genutzte Codeabschnitte zur Wiederverwendung definieren. Das trifft z. B. auf das grundlegende Seitengerüst zu oder auch auf Eingabekomponenten von Datensatz-neu-anlegen- oder Datensatz-bearbeiten-Formularen zu. Diese XHTML-Templates liegen in einem speziellen Ordner (/WEB-INF), der nicht von außen erreichbar ist sondern nur von projektinternen Webseiten aufgerufen werden kann. Diese Strategie erleichtert die geplante Umsetzung eines Codegenerators erheblich, da die Umfänge der generierten Seiten ein minimales Maß an Redundanzen besitzen.

---

[1] Webseite der H2 Database Engine [http://www.h2database.com/html/main.html]

[2] Vgl. Dokumentation der H2DB zu Clustering [http://www.h2database.com/html/advanced.html#clustering]

[3] Vgl. http://www.h2database.com/javadoc/org/h2/fulltext/FullText.html

[4] Vgl. Performancevergleich relationaler Datenbanken für Java [http://www.h2database.com/html/performance.html]

[5] CoRS-O-Mat Projektarchiv bei SourceForge [http://sourceforge.net/projects/corsomat/]

5.2.2. UUID Generator

Komponente: Storage/persistence

Die UUIDs für CoRS-O-Mat werden mit Hilfe von Java UUID Generator (JUG) von fasterXML[6] erzeugt und dann zur verbesserten Verarbeitung umgestellt, so dass sie in der Datenbank eine monotone Ordnung besitzen[7].

Das folgende Listing zeigt die Generator-Klasse für UUIDs:

```
 1 // ...
 2 public class TimeBasedUniqueId implements Serializable {
 3
 4  private static final long serialVersionUID = 1L;
 5
 6  private static final TimeBasedGenerator timeBasedGenerator = Generators
 7    .timeBasedGenerator(EthernetAddress.fromInterface());
 8
 9  // ...
10
11  public static final String generate() {
12   UUID uuid = timeBasedGenerator.generate();
13   List<String> uuidParts = Arrays.asList(uuid.toString().split("-"));
14   String uniqueId = uuidParts.get(2) + uuidParts.get(1) +
uuidParts.get(0) + uuidParts.get(3) + uuidParts.get(4);
15   return uniqueId;
16  }
17
18  // ...
19
20 }
```

5.2.3. JPA-Entitys

Komponente: Storage/persistence

Die JPA-Entitys bestehen aus annotierten Attributen und Gettern/Settern, sie enthalten keine Logik. Die Klassen werden von JPA aus der Datenbank generiert.

Das folgende Listing zeigt beispielhaft die Entity-Klasse der Tabelle `FEATURE`.

Die folgenden Java Annotations wurden verwendet:

- Table

[6] Webseite von JUG [http://wiki.fasterxml.com/JugHome]

[7] Vgl. Abschnitt 4.4.3, „Datenbankschlüssel“

Table definiert den Tabellennamen aus der Datenbank, zu der diese Entity zugehörig ist.

- NamedQuery
  NamedQuery definiert benannte Queries in Java Persistence Query Language[8].

- Id
  Ein mit Id annotiertes Feld zeichnet den Primärschlüssel dieser Entity aus. Dieser ist eine UUID mit dem Datentyp String.

- ManyToMany und JoinTable
  Mit diesen Annotationen werden die Fremdschlüsselbeziehungen der jeweiligen Tabellenspalte beschrieben. Die Daten stammen aus dem Datenbankschema.

Im Konstruktor der Entity-Klasse wird eine zeitstempelbasierte und umgestellte UUID für jedes neue Entity generiert.

```
 1 // ...
 2 @Entity
 3 @Table(name = "FEATURES")
 4 @NamedQuery(name = "Feature.findAll", query = "SELECT f FROM Feature f")
 5 public class Feature implements Serializable {
 6  private static final long serialVersionUID = 1L;
 7
 8  @Id
 9  private String id;
10
11  private String abbreviation;
12
13  private String name;
14
15  // bi-directional many-to-many association to ClauseWord
16  @ManyToMany
17  @JoinTable(name = "CLAUSE_WORD_FEATURES", joinColumns =
{ @JoinColumn(name = "FEATURE_ID") }, inverseJoinColumns = {
18    @JoinColumn(name = "CLAUSE_WORD_ID") })
19  private List<ClauseWord> clauseWords;
20
21  // bi-directional many-to-many association to Constituent
22  @ManyToMany
23  @JoinTable(name = "CONSTITUENT_FEATURES", joinColumns =
{ @JoinColumn(name = "FEATURE_ID") }, inverseJoinColumns = {
24    @JoinColumn(name = "CONSTITUENT_ID") })
25  private List<Constituent> constituents;
```

[8]Dokumentation der Java Persistence Query Language von Oracle [https://docs.oracle.com/javaee/6/tutorial/doc/bnbtg.html]

```
26
27  public Feature() {
28   this.id = TimeBasedUniqueId.generate();
29  }
30
31  public String getId() {
32   return this.id;
33  }
34
35  public void setId(String id) {
36   this.id = id;
37  }
38
39  public String getAbbreviation() {
40   return this.abbreviation;
41  }
42
43  public void setAbbreviation(String abbreviation) {
44   this.abbreviation = abbreviation;
45  }
46
47  public String getName() {
48   return this.name;
49  }
50
51  public void setName(String name) {
52   this.name = name;
53  }
54
55  public List<ClauseWord> getClauseWords() {
56   return this.clauseWords;
57  }
58
59  public void setClauseWords(List<ClauseWord> clauseWords) {
60   this.clauseWords = clauseWords;
61  }
62
63  public List<Constituent> getConstituents() {
64   return this.constituents;
65  }
66
67  public void setConstituents(List<Constituent> constituents) {
68   this.constituents = constituents;
69  }
70
71 }
```

5.2.4. Entity PersistenceBean

Komponente: Storage/persistence

Entity PersistenceBean ist eine Stateless EJB und sorgt für die Verfügbarkeit der Methoden delete, list, read und save. Sie agiert als Boundary gegenüber der Schicht Storage/transfer.

Das folgende Listing zeigt die Klasse FeaturePersistenceBean.

Die folgenden Java Annotations wurden verwendet:

- Stateless
  Stateless registriert die Klasse als Stateless EJB.

```
 1 // ...
 2 @Stateless
 3 public class FeaturePersistenceBean implements Serializable {
 4
 5  private static final long serialVersionUID = 1L;
 6
 7  @Inject
 8  private FeaturePersistenceControl persistenceControl;
 9
10  public boolean delete(Object id) {
11   return persistenceControl.delete(id);
12  }
13
14  public List<Feature> list() {
15   return persistenceControl.list();
16  }
17
18  public Feature read(Object id) {
19   return persistenceControl.read(id);
20  }
21
22  public Feature save(Feature entity) {
23   persistenceControl.save(entity);
24   return entity;
25  }
26
27 }
```

5.2.5. Entity PersistenceControl

Komponente: Storage/persistence

Entity PersistenceControl ist eine CDI Bean, welche die generisch typisierte Klasse AbstractPersistenceControl<Feature> erweitert. Sie agiert als Control der Persistenzschicht und sorgt für die Ausführung der Methoden delete, list, read und save auf dem Entity. Die Implementierung der Methoden befindet sich in der Klasse AbstractPersistenceControl.

Das folgende Listing zeigt die Klasse FeaturePersistenceControl.

Die folgenden Java Annotations wurden verwendet:

- Dependent
  Eine Klasse, die mit Dependent annotiert wurde, ist zur Laufzeit an den Lebenszyklus seines Kontexts gebunden.

```
 1 // ...
 2 @Dependent
 3 public class FeaturePersistenceControl extends
AbstractPersistenceControl<Feature> {
 4
 5  private static final long serialVersionUID = 1L;
 6
 7 }
```

Das folgende Listing zeigt die Klasse AbstractPersistenceControl.

Die folgenden Java Annotations wurden verwendet:

- Inject
  Inject ist ein CDI-Mechanismus, um den Container anzuweisen, eine Instanz der annotierten Klasse zur Laufzeit zu injizieren und zu verwalten.

- PostConstruct
  Mit PostConstruct annotierte Methoden werden ausgeführt, nachdem die Injektion stattgefunden hat.

- PersistenceContext
  Mit dem Attribut unitName wird der Name der Datenbankinstanz definiert. Dadurch kann JPA dem EntityManager den richtigen Kontext zuordnen.

Ausnahmen, die z. B. durch den EntityManager geworfen werden könnten, werden abgefangen und als Fehlereintrag im Errorlog vermerkt.

```
 1 // ...
 2 public abstract class AbstractPersistenceControl<EntityType> implements
Serializable {
 3
```

```
 4  private static final long serialVersionUID = 1L;
 5
 6  @Inject
 7  private transient Logger log;
 8
 9  @PersistenceContext(unitName = "persistence-unit_corsomat-storage")
10  private EntityManager entityManager;
11
12  protected Class<EntityType> entityClass;
13
14  private CriteriaBuilder criteriaBuilder;
15
16  @PostConstruct
17  private void initialize() {
18   this.criteriaBuilder = this.entityManager.getCriteriaBuilder();
19  }
20
21  public Boolean delete(Object id) {
22   Boolean result = false;
23   try {
24    EntityType entity = this.entityManager.find(this.entityClass, id);
25    this.entityManager.remove(entity);
26    this.entityManager.flush();
27    result = true;
28    log.debug("EntityManager deleted " + this.entityClass.getSimpleName()
+ " with ID: " + id);
29   } catch (Exception exception) {
30    log.error("EntityManager failed to delete " +
this.entityClass.getSimpleName() + " with ID: " + id);
31   }
32   return result;
33  }
34
35  public List<EntityType> list() {
36   List<EntityType> resultList = null;
37   try {
38    CriteriaQuery<EntityType> criteriaQuery =
this.criteriaBuilder.createQuery(entityClass);
39    Root<EntityType> criteriaQueryRoot = criteriaQuery.from(entityClass);
40    criteriaQuery.select(criteriaQueryRoot);
41    TypedQuery<EntityType> typedQuery =
this.entityManager.createQuery(criteriaQuery);
42    resultList = typedQuery.getResultList();
43    log.debug("EntityManager queried " + resultList.size() + " " +
entityClass.getSimpleName());
44   } catch (Exception exception) {
45    log.error("EntityManager failed to query for list ");
46   }
```

```
47   return resultList;
48  }
49
50  public EntityType read(Object id) {
51   EntityType entity = null;
52   try {
53    entity = this.entityManager.find(this.entityClass, id);
54    log.debug("EntityManager found " + this.entityClass.getSimpleName() +
" with ID: " + id);
55   } catch (Exception exception) {
56    log.error("EntityManager failed to find " +
this.entityClass.getSimpleName() + " with ID: " + id);
57   }
58   return entity;
59  }
60
61  public void save(EntityType entity) {
62   try {
63    this.entityManager.persist(entity);
64    this.entityManager.flush();
65    this.entityManager.refresh(entity);
66    log.debug("EntityManager persisted " +
this.entityClass.getSimpleName());
67   } catch (Exception exception) {
68    log.error("EntityManager failed to persist " +
this.entityClass.getSimpleName());
69   }
70  }
71
72 }
```

5.2.6. RESTful Entity TransferBean

Komponente: Storage/transfer

Entity TransferBean agiert als Boundary der Transferschicht nach Außen. Sie ist nach der API-Spezifikation JAX-RS 2.0 implementiert[9]. Die in der Spezifikation definierten Annotationen sorgen für die Sichtbarkeit und Erreichbarkeit des REST-Webservices. Das folgende Codebeispiel veranschaulicht die Implementierung der REST-Schnittstelle am Beispiel des Entitys `Clause`.

Die folgenden Java Annotations wurden verwendet:

- Stateless

[9] Vgl. [BURKE.2013], S.27-42

Stateless registriert die Klasse als Stateless EJB.

- Inject
  Inject ist ein CDI-Mechanismus, um den Container anzuweisen, eine Instanz der annotierten Klasse zur Laufzeit zu injizieren und zu verwalten.

- Consumes
  Consumes ist eine REST-Annotation, die definiert, welche Datenformate zum Empfang akzeptiert werden.

- Produces
  Produces ist eine REST-Annotation, die definiert, welche Datenformate als Ausgabe erzeugt werden können.

- Path
  Path ist eine REST-Annotation, die festlegt, über welchen Pfad (URI) die Klasse und die jeweiligen Methoden erreichbar sind.

- GET, POST, PUT, DELETE
  GET, POST, PUT und DELETE sind REST-Annotationen, die das Mapping zwischen den Klassenmethoden und den HTTP-Methoden zum Lesen und Schreiben von Daten definieren.

Ausnahmen, die z. B. durch das TransferControl geworfen werden könnten, werden abgefangen und als Fehlereintrag im Errorlog vermerkt. Nach Außen werden Serverfehler (Fehlercode 500) weitergegeben, damit sich ein potentieller Angreifer nicht über Fehlerausgaben ein Bild der internen Anwendungsstruktur machen kann.

```
 1 // ...
 2 @Stateless
 3 @Path("clauses")
 4 public class ClauseTransferBean implements Serializable {
 5
 6  private static final long serialVersionUID = 1L;
 7
 8  @Inject
 9  private transient Logger log;
10
11  @Inject
12  protected ClauseTransferControl transferControl;
13
14  @DELETE
15  @Path("{id}")
16  public Response delete(@PathParam("id") String id) {
17   try {
18    if (transferControl.delete(id)) {
```

```
19     return Response.ok().build();
20    } else {
21     log.error("Failed to DELETE details with ID " + id);
22     return Response.serverError().build();
23    }
24   } catch (Exception exception) {
25    log.error("Failed to DELETE details with ID " + id);
26    throw new
WebApplicationException(Response.Status.INTERNAL_SERVER_ERROR);
27   }
28  }
29
30  @GET
31  @Produces({ MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML })
32  public ClauseSet get() {
33   try {
34    return transferControl.assemble();
35   } catch (Exception exception) {
36    log.error("Failed to GET container");
37    throw new WebApplicationException(Response.Status.NOT_FOUND);
38   }
39  }
40
41  @GET
42  @Path("{id}")
43  @Produces({ MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML })
44  public ClauseDetails getById(@PathParam("id") String id) {
45   try {
46    return transferControl.assemble(id);
47   } catch (Exception exception) {
48    log.error("Failed to GET Details for ID " + id);
49    throw new WebApplicationException(Response.Status.NOT_FOUND);
50   }
51  }
52
53  @POST
54  @Produces({ MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML })
55  public Response post(ClauseDetails details) {
56   try {
57    ClauseDetails newDetails = transferControl.create(details);
58    URI uri = URI.create("/" + newDetails.getId());
59    return Response.created(uri).build();
60   } catch (Exception exception) {
61    log.error("Failed to POST Details");
62    throw new WebApplicationException(exception,
Response.Status.INTERNAL_SERVER_ERROR);
63   }
64  }
```

```
65
66  @PUT
67  @Path("{id}")
68  @Consumes({ MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML })
69  public Response put(@PathParam("id") String id, ClauseDetails details)
{
70   try {
71    ClauseDetails updatedDetails = transferControl.update(details);
72    URI uri = URI.create("/" + updatedDetails.getId());
73    return Response.seeOther(uri).build();
74   } catch (Exception exception) {
75    log.error("Failed PUT Details for ID " + id);
76    throw new WebApplicationException(exception,
Response.Status.INTERNAL_SERVER_ERROR);
77   }
78  }
79
80 }
81
```

5.2.7. Entity TransferControl

Komponente: Storage/transfer

Entity TransferControl agiert als Control der Transferschicht. Sie ist dafür zuständig, ein Messages-Objekt zum Versenden zu erzeugen oder ein empfangenes Messages-Objekt zu einem neuen oder bestehenden Entity zuzuordnen.

Die folgenden Java Annotations wurden verwendet:

- Dependent
  Eine Klasse, die mit Dependent annotiert wurde, ist zur Laufzeit an den Lebenszyklus seines Kontexts gebunden.

- Inject
  Inject ist ein CDI-Mechanismus, um den Container anzuweisen, eine Instanz der annotierten Klasse zur Laufzeit zu injizieren und zu verwalten.

- EJB
  EJB injiziert eine EJB-Instanz in den aktuellen Kontext.

```
1 // ...
2 @Dependent
3 public class ClauseTransferControl implements Serializable {
4
5  private static final long serialVersionUID = 1L;
```

```
 6
 7  @Inject
 8  private transient Logger log;
 9
10  @EJB
11  private ClausePersistenceBean persistenceBean;
12
13  @Inject
14  private LanguageTransferControl languageTransferControl;
15
16  public ClauseSet assemble() {
17   ClauseSet set = null;
18   List<Clause> list = persistenceBean.list();
19   if (list != null) {
20    set = new ClauseSet();
21    set.setClauses(new ArrayList<ClauseDetails>());
22    for (Iterator<Clause> iterator = list.iterator();
iterator.hasNext();) {
23     Clause currentListItem = iterator.next();
24     ClauseDetails details = assemble(currentListItem.getId());
25     set.getClauses().add(details);
26    }
27    log.debug("Container assembled and contains " +
set.getClauses().size() + " details");
28   }
29   return set;
30  }
31
32  public ClauseDetails assemble(Object id) {
33   ClauseDetails details = null;
34   Clause entity = persistenceBean.read(id);
35   if (entity != null) {
36    details = new ClauseDetails();
37    details.setId(entity.getId());
38
details.setLanguage(languageTransferControl.assemble(entity.getLanguage()));
39    details.setText(entity.getText());
40   }
41   return details;
42  }
43
44  public ClauseDetails create(ClauseDetails details) {
45   Clause entity = dismantle(details);
46   entity = persistenceBean.save(entity);
47   return assemble(entity);
48  };
49
50  public Boolean delete(Object id) {
```

```
51   return persistenceBean.delete(id);
52  };
53
54  public Clause dismantle(ClauseDetails details) {
55   Clause entity = new Clause();
56   entity.setId(details.getId());
57
entity.setLanguage(languageTransferControl.dismantle(details.getLanguage()));
58   entity.setText(details.getText());
59   return entity;
60  }
61
62  public List<Clause> dismantle(ClauseSet set) {
63   List<Clause> list = new ArrayList<Clause>();
64   for (Iterator<ClauseDetails> iterator = set.getClauses().iterator();
iterator.hasNext();) {
65    ClauseDetails currentDetails = iterator.next();
66    Clause entity = dismantle(currentDetails);
67    list.add(entity);
68    log.debug("Conainer dismantled to " + list.size() + " entities");
69   }
70   return list;
71  }
72
73  public ClauseDetails update(ClauseDetails details) {
74   ClauseDetails updatedDetails = null;
75   Clause entity = persistenceBean.read(details.getId());
76   if (entity != null) {
77    entity = dismantle(details);
78    entity = persistenceBean.save(entity);
79    updatedDetails = assemble(entity);
80   }
81   return updatedDetails;
82  }
83
84 }
85
```

5.2.8. Objektrelationale Konverter

Komponente: Pages

Bei CoRS-O-Mat kommen objektrelationale Konverter zum Einsatz. Wenn beispielsweise eine Fremschlüsselbeziehung zwischen mehreren Tabellen der Datenbank besteht und ein neuer Datensatz angelegt werden soll, so wird die Festlegung des Werts für den Fremdschlüssel auf GUI-Ebene üblicherweise mit einer Drop-Down-Liste umgesetzt. Als Beispiel sei hier das Anlegen eines neuen Satzes in einer bestimmten Sprache anzuführen, wobei eine neue Zeile in der

Tabelle CLAUSES angelegt werden soll. Es besteht eine Fremdschlüsselbeziehung zur Tabelle LANGUAGES, welche als Auswahl einer der vorhandenen Sprachen als Drop-Down-Liste dargestellt wird. Diese Listen speichern als Werte aber nur Strings und keine Objekte. Um Typsicherheit gewährleisten zu können, müssen sogenannte Konverter programmiert werden, die den Zeichenwert des gewählten Eintrags zur Laufzeit in ein Objekt übersetzen. Es folgt ein Beispiel für einen solchen Konverter.

Die folgenden Java Annotations wurden verwendet:

- Named
  Named definiert den Namen, der für die Injektion von CDI-Klassen verwendet werden soll anstelle des standardmäßig vergebenen Namens.

- Inject
  Inject ist ein CDI-Mechanismus, um den Container anzuweisen, eine Instanz der annotierten Klasse zur Laufzeit zu injizieren und zu verwalten.

- Override
  Override annotiert, dass eine vorhandene Methode der Superklasse überschrieben werden soll.

```
 1 @Named("languageConverter")
 2 public class LanguageConverter implements Converter {
 3
 4  @Inject
 5  private LanguageControl languageControl;
 6
 7  @Override
 8  public Object getAsObject(FacesContext context, UIComponent component,
String value) {
 9   return (LanguageDetails) languageControl.fetch(value);
10  }
11
12  @Override
13  public String getAsString(FacesContext context, UIComponent component,
Object value) {
14   return ((LanguageDetails) value).getId();
15  }
16
17 }
```

5.2.9. Datenimport

Die bisher gesammelten Daten des Projekts CoRS müssen in die Software CoRS-O-Mat übernommen werden. Die initiale Überlegung, einen Parser zu programmieren, der die Daten voll-

automatisch aus der Word-Datei in die CoRS-O-Mat-Datenbank importiert, musste verworfen werden. Es gibt im Projekt CoRS kein strukturiertes Datenmodell, d. h. die Daten liegen nicht in normalisierter Form vor, so dass ein automatisches Parsen nicht möglich sein kann. Stattdessen wird ein automatisierter Teilimport der Daten realisiert, damit der manuelle Arbeitsaufwand für den Datenimport möglichst klein ausfällt. Die folgende Abbildung zeigt einen Ausschnitt aus der CoRS Datentabelle. Hier wird das Problem deutlich, die Daten sind nicht normalisiert, d. h. es gibt hier pro Spalte zum Teil mehrere Zeilen Einträge und die Daten sind uneinheitlich in der verwendeten Annotationssymbolik.

1	Order	i-status of preverbal element(s)	Inf. focus	Contr. focus/ emph.	Comments	V t	V type (or Pred type)	Italian	Romanian	Spanish
(1)	V S		all		unexpressed location/tim	una	eventive + implicit loc. argument	(1) E entrato Dante (Pinto 1997:23) (2) All'improvviso è entrato il cane dalla finestra But, when the implicit goal is not given: * Parte Maria (Benincà 1988, ap. Tortora 2001:314) ?? E partita Maria (Tortora 2001:314) (but Maria may be given here, if she is supposed to leave; perhaps – other contexts: what's the noise? A plane is leaving) Tortora (2001:316): goal must be present: () E sceso Lo Spitfire (*per 5 minuti)	(1) A intrat Dante (Giurgea & Weingart 200 (2) A murit Fellini (3) Dintr-o dată a intrat cãinele pe fereastră	(1) Entró Dante Ha llegado Juan (Giurgea Llegó tu hermano (Gutiér
(2)	V S		all		R: neuter order with (epi passive *se* (vs. habitual p *se*, = "middle")	una	episodic, passive	Benincà (1988:124): Si vendono appartamenti a Padova	S-au emis tot felul de părери despre acest act (oral corpus, 42) REFL-have.3PL expressed all sort of opinions about this act	the same?
(3)	V S		all		unexpressed location/tim c-given	une	eventive, +implicit loc. arg.	Ha telefonato Beatrice (Pinto 1997:23) Ha suonato il postino (Benincà 1988:124)	A telefonat Beatrice	Ha llamado Bea
(4)	V S		all		unexpressed location/tim c-given	una	eventive, - implicit loc. argument, +independent location,	(1) E' morto Fellini (Pinto 1997:23) (2) È bruciata la casa	A ars casa	Se quemó la casa. Se estropeó la maquina. Murió Fellini (murió Fellini una muerte unerg)
(5)	V S		all		I: cf. Pinto, only poss. with the indef. art.;	une	eventive, - implicit loc. argument, +independent location,	(1) Ha pianto un bambino (Pinto 1997:101), but: (2) # Hanno pianto due bambini (Pinto 1997:181) But: (3) Hanno pianto dei bambini #Ha pianto Beatrice (Pinto 1997:28) # Hanno pianto venti bambini (Pinto 1997:101) (4) [context: chè cosa si sente?] Piange il biambino (informant) (5) # Ha riso Gigi (Pinto 94, 178, ap Zubizarreta 98:119) Z:* (6) # Hanno starnutito tre leoni (Adger 1996, 118, ap. Zubizarreta 98:118) Z: But, with verbs of saying (Silvio Cruschina, p.c.):	(1) A plâns un copil / Au plâns niște copii (Weingart 2009) has cried a child / have cried some children (2) Au plâns doi copii have cried two children (3) Plânge copilul cries child-the (4) Au intrat (trei mii de mineri) în grevă (trei mii de mineri) have entered three thousand of miners in strike (..) (5) A tuşit cineva has caughed somebody (6) A ţipat un copil has yelled a child	Ha llorado un niño. Han llorado dos niños. Se rió Juan.(Zubizarreta 1 Han estornudado tres leon (Zubizarreta 1998:119)

Abbildung 5.4. Auszug aus der CoRS Datentabelle

5.2.10. Sätze zerlegen

Eine wichtige Funktionalität von CoRS-O-Mat ist das Zerlegen von Sätzen, die der Benutzer neu erfassen möchte. Dabei wird der eingegebene Satz nach fest definierten Regeln automatisch zerlegt, das Ergebnis wird dem Benutzer als Zwischenergebnis präsentiert und er hat dann nochmal die Möglichkeit, weitere Zerlegungen durchzuführen, bevor der eingegebene Satz endgültig gespeichert wird.

Kapitel 6. Diskussion: Moderne Software und Zugänglichkeit

Bei dem Entwurf von CoRS-O-Mat im Rahmen dieser Arbeit ist aufgefallen, dass es durchaus zu Widersprüchen kommen kann, wenn Software nach modernen Standards entwickelt werden soll, die zugänglich ist im Sinne der Barrierefreiheit, kompatibel ist zu älteren Geräten und dabei moderne Technologien verwendet.

Von einer modernen webbasierten Software wird üblicherweise erwartet, dass sie dem Benutzer kontinuierliche und eindeutige Rückmeldungen beim Ausführen von Benutzeraktionen gibt, dass die Benutzeroberfläche nicht einfriert, wenn längere Aktionen ausgeführt werden und dass moderne Eingabemechanismen wie z. B. Drag'n'Drop unterstützt werden. Solche Funktionalitäten lassen sich in einer Webanwendung ausschließlich durch die Verwendung von JavaScript umsetzen. Gleichzeitig soll eine Webanwendung zugänglich und barrierefrei sein, was wiederum stark gegen die Verwendung von AJAX mit JavaScript spricht. Schaltet man in einer modernen Webanwendung im Browser das JavaScript aus, so ist die Wahrscheinlichkeit groß, dass diese danach nicht mehr korrekt funktioniert. Das liegt daran, dass selbst einfache Requests, die mit Betätigung eines Buttons abgeschickt werden sollen, üblicherweise und unnötigerweise direkt in JavaScript implementiert wurden. Barrierefreiheit ist in einer solchen Anwendung üblicherweise auch nicht gegeben, da ScreenReader für blinde Menschen z. B. gar keinen HTML-Quellcode zum vorlesen vorfinden, da die Elemente alle dynamisch zur Laufzeit generiert werden. Das kann für ScreenReader problematisch sein.

CoRS-O-Mat verwendet AJAX und JavaScript nur implizit im Frontend, da jQuery mobile auf JavaScript basiert und ist so programmiert, dass bei deaktiviertem JavaScript der Kern der Anwendung weiterhin funktioniert und benutzbar ist. Bei der Wahl des Frontend-Frameworks fiel die Entscheidung dennoch zugunsten von jQuery mobile aus, da die primäre Zielsetzung von CoRS-O-Mat die geräteübergreifende Verfügbarkeit ist. Zudem enthält das Framework eine konfigurierbare grafische Benutzeroberfläche, so dass keine zusätzlichen CSS-Anweisungen implementiert werden müssen, was den Entwicklungsaufwand deutlich reduziert, besonders im Hinblick auf die browserübergreifende Kompatibilität. Das Frontend ließe sich bei Bedarf durch eines ersetzen, welches ohne JavaScript auskommt.

Neben JavaScript sind auch Session-Cookies in Verruf geraten, da diese z. B. in der jüngsten Vergangenheit exzessiv durch die weltweiten Werbenetzwerke genutzt wurden, um Benutzer unabhängig von einer Anmeldung zu identifizieren und dessen Aktionen zu verfolgen. Aus diesem Grund werden Cookies heutzutage von einigen Benutzern deaktiviert.

CoRS-O-Mat sollte als vollständig zustandslose Anwendung konzipiert werden, d. h. ohne zustandsbehaftete Session auf dem Server und ohne Zustandsinformationen in einem Session-Cookie. Jeder Request enthält alle notwendigen Informationen. Bei der Authentifizierung

nach OAuth2 mit JSON Web Token (JWT) gibt es an dieser Stelle aber bereits Schwierigkeiten, denn das Token muss vom Browser bei jedem Request erneut mitgeschickt werden. Dafür muss das Token lokal beim Benutzer gespeichert werden. Um das umzusetzen, hat man generell zwei Möglichkeiten: Entweder der Token wird in einem Session-Cookie gespeichert oder mittels JavaScript im sogenannten WebStorage. Da das WebStorage API nur in neueren HTML5-konformen Browsern umgesetzt wurde, speichert CoRS-O-Mat das Token in einem Session-Cookie, welcher nur als Speicher verwendet wird und nicht, um den Nutzer über verschiedene Webseiten hinweg zu identifizieren oder zu verfolgen.

Kapitel 7. Fazit

Die folgenden Abschnitte fassen die wichtigsten Aspekte dieser Arbeit noch einmal zusammen und geben einen Ausblick auf künftige Entwicklungen der Software.

7.1. Zusammenfassung

Die Anwendungssoftware CoRS-O-Mat wurde entwickelt, um das geisteswissenschaftliche Forschungsprojekt CoRS zu digitalisieren und um einen Relaunch des Projekts initiieren zu können. Auch andere wissenschaftliche Projekte im Bereich der digitalen Geisteswissenschaften sollen CoRS-O-Mat anpassen und einsetzen können. Besonders kleinere Projekte mit dem Bedarf nach einer leichtgewichtigen, aber leistungsstarken Anwendung können von der Anwendungssoftware profitieren. Mit geringen Änderungen ist es möglich, CoRS-O-Mat an die Anforderungen anderer Projekte anzupassen. Das wurde erreicht durch eine komponentenbasierte Architektur, in der die Komponenten über die REST-Schnittstelle lose miteinander gekoppelt sind. Zudem sind die Komponenten isoliert, d. h. keine Komponente kennt die internen Mechanismen einer anderen Komponente. Dadurch ist es zum Einen möglich, einzelne Komponenten bei Bedarf auszutauschen und zum Anderen kann man durch Mocking der externen Abhängigkeiten die interne Funktionalität durch UnitTests einfacher testen, als bei komplexen, direkt verdrahteten Komponenten. Zum Datenaustausch verwenden die Komponenten die Messages-Objekte der Komponente Messages. Das sind im Wesentlichen transaktional annotierte EJBs, die besonders einfach eingebunden werden können, da sie im Kern einfach nur POJOs sind. Ein weiteres Architekturmerkmal ist die datenbankunabhängige Implementierung der Komponente Storage. Liegt ein vernünftiges Domänenmodell vor und ist eine Datenbank bereits in Betrieb, so kann die Datenbank an das System angebunden werden, sofern ein JDBC-Treiber für die entsprechende Datenbank existiert, was für die meisten Datenbanken der Fall sein dürfte. Existiert noch keine Datenbank, so kann diese mit Papyrus modelliert und mit Liquibase erzeugt werden. Die entsprechenden Entitys werden durch JPA automatisch generiert.

CoRS-O-Mat wurde unter Berücksichtigung der aktuellen Java EE Standards und unter Nutzung der verfügbaren Referenzimplementierungen der offiziellen Spezifikationen entwickelt. Dadurch ist sichergestellt, dass die Anwendung nur gegen offizielle Standards programmiert und keine proprietären Interfaces implementiert wurden, damit sie mit allen anderen standardkonformen Java EE Architekturkomponenten interoperabel ist. Um eine unkomplizierte Verwendung zu ermöglichen, wird CoRS-O-Mat als freie Software unter der European Union Public Licence (EUPL)[1] im Softwarearchiv von Sourceforge[2] publiziert. Durch die Orientierung an modernen Entwurfsmustern, wie z. B. Responsive Web Design, Progressive Enhancement, ECB und MVC

[1] European Union Public Licence (EUPL) [https://joinup.ec.europa.eu/software/page/eupl]

[2] Webseite der CoRS-O-Mat Anwendungssoftware bei SourceForge [http://sourceforge.net/projects/corsomat/]

und der browserbasierten Architektur erreicht CoRS-O-Mat ein Maximum an Plattform- und Geräteunabhängigkeit.

Vergleichbare Software ist entweder zu schwergewichtig, weil sie beispielsweise die Installation einer großen Desktopanwendung voraussetzt oder sie ist nicht in dem Maße anpassbar, dass man sie mit einfachen Mitteln in einem neuen Projekt verwenden kann. Mobilgeräte und plattformübergreifende Nutzung werden auch nicht zwingend von jeder aktuellen Software unterstützt. Diese Aspekte sind aber wichtig für die Produktivität und Zugänglichkeit moderner Software.

7.2. Ausblick

Die hier vorgestellten Funktionalitäten sind nicht Bestandteil dieser Arbeit, sondern stellen geplante Umsetzungen dar, die in einem Folgeprojekt umgesetzt werden sollen.

7.2.1. Generative Grammatik

Ein Hauptziel des Projekts CoRS ist es, eine übereinzelsprachlich gültige generative Grammatik zu entwickeln, die für alle untersuchten Sprachen gültige Satzstellungsmuster erzeugt und ungültige ausschließt. Die Entwicklung dieser Funktionalität ist unter Nutzung der Methodik des maschinellen Lernens geplant. Einerseits soll es dem Benutzer ermöglicht werden, Grammatiken zu definieren und manuell gegen die Daten zu testen, andererseits soll das Programm selbständig lernen und eigene Vorschläge unterbreiten können.

7.2.2. Codegenerierung aus UML

Um die Anpassbarkeit von CoRS-O-Mat weiter zu vereinfachen, kann die gesamte Komponente Storage und Teile der anderen Komponenten automatisch generiert werden. Die Software Papyrus von der Eclipse Foundation ist ein Editor für EMF-basierte Modellierungssprachen, wie UML, und läuft als Plug-In innerhalb von Eclipse. CoRS-O-Mat verwendet Papyrus bereits bei der UML2-Modellierung der Datendomäne. Papyrus ist in der Lage, aus einem gegebenen Diagramm Java-Code zu generieren. Es verwendet dafür MOF Model to Text Transformation (MOFM2T)[3]. Da die Klassen der Komponente Storage keine Businesslogik enthalten, wäre es sinnvoll, alle oder zumindest den Großteil aller Klassen automatisch zu erzeugen.

7.2.3. Dokumentation

Es soll eine Softwaredokumentation erzeugt werden, die auf verständliche Weise darstellt, wie die Anwendungssoftware CoRS-O-Mat aufgebaut ist, wie sie funktioniert und welche Maßnahmen nötig sind, um sie an andere Anforderungen anzupassen. Diese Dokumentation soll auf Basis

[3]Webseite von MOFM2T [http://www.omg.org/spec/MOFM2T/]

von DocBook 5[4] erzeugt werden. DocBook-Dokumente sind in XML verfasst und ermöglichen die Ausgabe in verschiedenen Formaten, wie z. B. PDF, EPUB, HTML oder Text durch XSL-Transformationen. DocBook-Dokumente lassen sich in das verwendete Versionierungssystem einbinden.

[4]DocBook 5: The Definitive Guide [http://www.docbook.org/tdg5/en/html/docbook.html]

Anhang A. CoRS-O-Mat Domänenmodell

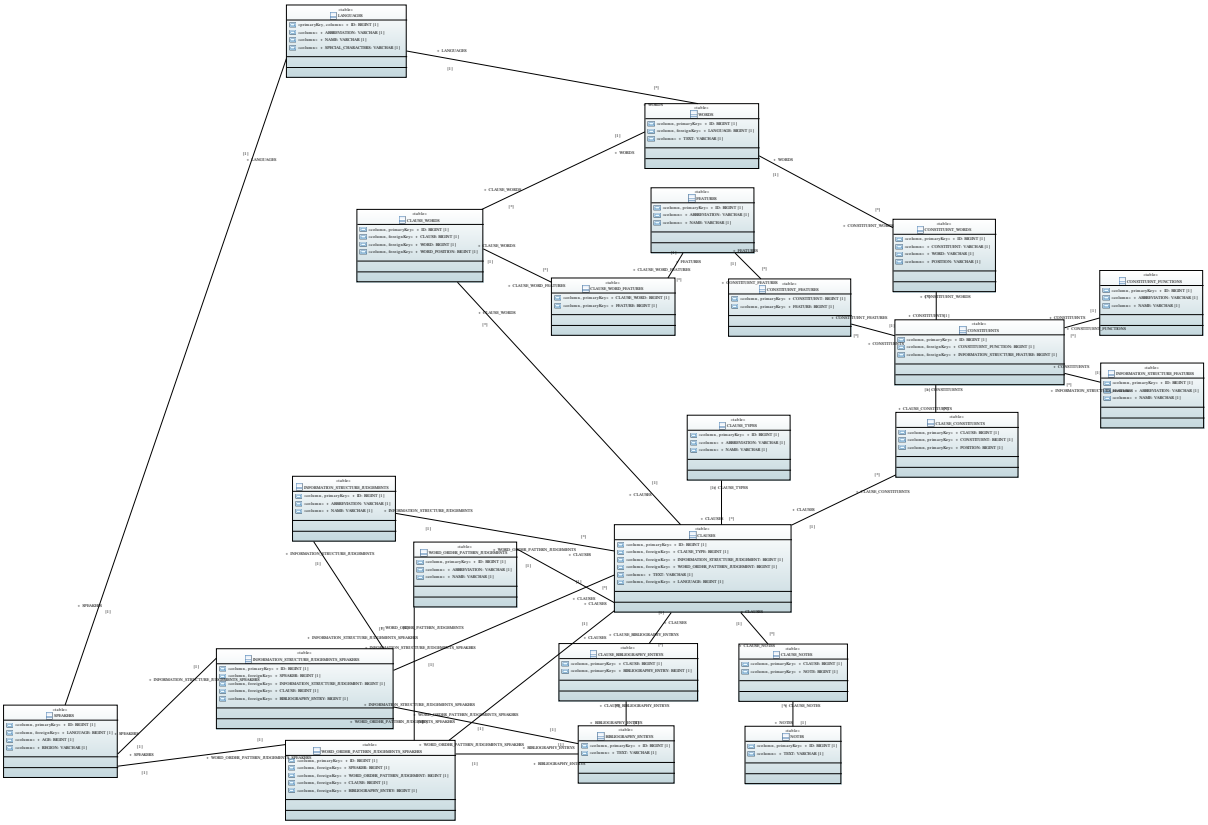

Abbildung A.1. CoRS-O-Mat Domänenmodell als UML Diagramm

Literaturverzeichnis

BURKE.2013: BURKE, Bill. *RESTful Java with JAX-RS 2.0: Designing and Developing Distributed Web Services*. Zweite Auflage. Sebastopol: O'Reilly Media, Inc., November 2013. ISBN 987-1-449-36134-1.

OECHSLE.2013: OECHSLE, Rainer. *JAVA-Komponenten: Grundlagen, prototypische Realisierung und Beispiele für Komponentensysteme*. Erste Auflage. München: Carl Hanser Velag, 2013. ISBN 978-3-446-43176-8.

BIEN.2011: BIEN, Adam. *Real World Java EE Night Hacks: Dissecting the Business Tier*. Erste Auflage. press.adam.bien.com, April 2011. ISBN 978-1-4476-7231-9.

BIEN.2012: BIEN, Adam. *Java EE Patterns: Rethinkening Best Practices*. Zweite Auflage. press.adam.bien.com, September 2012. ISBN 978-1-300-14931-6.

GUPTA.2013: GUPTA, Arun. *Java EE 7 Essentials*. Erste Auflage. Sebastopol: O'Reilly Media, Inc., August 2013. ISBN 978-1-449-37017-6.

HERDEN.ET.AL.2006: HERDEN, Sebastian, GÓMEZ, Jorge und RAUTENSTRAUCH, Claus, et al. *Software-Architekturen für das E-Business*. Erste Auflage. Berlin: Heidelberg: New York: Springer Verlag, 2006. ISBN 978-3-540-25821-6.

WIRDEMANN.2011: WIRDEMANN, Ralf. *Scrum mit User Stories*. Zweite Auflage. München: Wien: Carl Hanser Verlag, 2011. ISBN 978-3-446-42660-3.

SCHADOW.2014: SCHADOW, Dominik. *Java Web Security: Sichere Webanwendungen mit Java entwickeln*. 1. Auflage. Heidelberg: dpunkt.verlag GmbH, 2014. ISBN 978-3-86490-146-1.

GONCALVES.2013: GONCALVES, Antonio. *Beginning Java EE 7*. 1. Auflage. Apress Media LLC, 2013. ISBN 978-1-4302-4626-8.

CHOMSKY.2000: CHOMSKY, Noam. Minimalist Inquiries: The Framework. In MARTIN, R., MICHAELS, D. und URIAGEREKA, J. (ed.). *Step by Step: Essays on Minimalist Syntax in Honor of Howard Lasnik*. Cambridge: MIT press, 89-155.

CHOMSKY.2001: CHOMSKY, Noam. Derivation by Phase. In KENSTOWICZ, Michael (ed.). *Ken Hale. A life in language*. Cambridge: MIT press, 1-52.

CHOMSKY.2004: CHOMSKY, Noam. Beyond Explanatory Adequacy. In BELLETTI, A. (ed.). *Structures and Beyond: The Cartography of Syntactic Structures*. New York: Oxford University Press, 104-131.

CHOMSKY.2008:

WEINGART.2011: WEINGART, Anja. *Vorstellung des Datenbankentwurfs für das DFG-Projekt „Bausteine Romansicher Syntax“: Vortrag im Colloquium für Doktoranden*. Berlin: Freie Universität, 2011-03-11.

GIURGEA.REMBERGER.2012: ION, Giurgea und REMBERGER, Eva-Maria. Zur informationsstrukturellen Konfiguration und Variation postverbaler Subjekte in den romanischen Null-Subjekt-Sprachen. *Zeitschrift für Sprachwissenschaft*. 2012, 31, 1, 43-99.

MENSCHING.WEINGART.2016: MENSCHING, Guido und WEINGART, Anja. The Null Subject Parameter and the lexicon in minimalist syntax. *Rethinking Parameters*. 2016.

WEB.FSF: *Freie Software. Was ist das? - GNU-Projekt - Free Software Foundation* [online]. Free Software Foundation, Inc., 1996-2015 [Zugriff am 2015-12-31]. Verfügbar im World Wide Web unter: <https://www.gnu.org/philosophy/free-sw.html>.

WEB.CC: *Was ist CC? Creative Commons* [online]. Creative Commons Deutschland, 2007-2015 [Zugriff am 2015-12-31]. Verfügbar im World Wide Web unter: <http://de.creativecommons.org/was-ist-cc/>.

WEB.UML: *OMG UML* [online]. Object Management Group, Inc., 1997-2016 [Zugriff am 2015-12-31]. Verfügbar im World Wide Web unter: <http://www.omg.org/getting-started/what_is_uml.htm>.

WEB.ADAM.BIEN.SIMPLICITY: BIEN, Adam. *Java EE 6: Simplicity by Design* [online]. Oracle, 2011 [Zugriff am 2015-12-31]. Verfügbar im World Wide Web unter: <http://www.oracle.com/au/products/database/o11java-195110.html>.

WEB.TODD.FREDRICH.REST: FREDRICH, Todd. *What to Use as Identifiers (IDs) in REST APIs* [online]. 2013-08-04 [Zugriff am 2015-12-31]. Verfügbar im World Wide Web unter: <http://toddfredrich.com/ids-in-rest-api.html>.

WEB.LIEVEN.DOCLO.UUID: DOCLO, Lieven. *What to Use as Identifiers (IDs) in REST APIs* [online]. 2015-05-30 [Zugriff am 2015-12-31]. Verfügbar im World Wide Web unter: <http://www.insaneprogramming.be/blog/2015/05/30/the-uuid-discussion/>.

WEB.TATU.SALORANTA.JUG: SALORANTA, Tatu. *More on Java UUID Generator (JUG), a word on performance* [online]. 2010-10-14 [Zugriff am 2015-12-31].

Verfügbar im World Wide Web unter: <http://www.cowtowncoder.com/blog/archives/2010/10/entry_429.html>.

WEB.KARTHIK.APPIGATLA.UUID: APPIGATLA, Karthik. *Store UUID in an optimized way* [online]. Percona, 2014-12-19 [Zugriff am 2015-12-31]. Verfügbar im World Wide Web unter: <https://www.percona.com/blog/2014/12/19/store-uuid-optimized-way/>.

WEB.IETF.UUID: LEACH, P., MEALLING, M. und SALZ, R. *RFC4122: A Universally Unique IDentifier (UUID) URN Namespace* [online]. IETF, 2005 [Zugriff am 2015-12-31]. Verfügbar im World Wide Web unter: <https://tools.ietf.org/html/rfc4122>.

WEB.GIBSON.BEANS: GIBSON, Andy. *Comparing JSF Beans, CDI Beans and EJBs* [online]. 2012-02-07 [Zugriff am 2015-12-31]. Verfügbar im World Wide Web unter: <http://www.andygibson.net/blog/article/comparing-jsf-beans-cdi-beans-and-ejbs/>.

WEB.LIMBURG.RÖWEKAMP.BEANS: LIMBURG, Arne und RÖWEKAMP, Lars. *Wofür braucht man in Java EE 7 eigentlich noch EJBs?* [online]. JAXenter, 2015-02-02 [Zugriff am 2015-12-31]. Verfügbar im World Wide Web unter: <https://jaxenter.de/wofuer-braucht-man-java-ee-7-eigentlich-noch-ejbs-13386>.

WEB.HOFSTETTER.RESTFUL: HOFSTETTER, Nicolas. *RESTful JSF with POST-redirect-GET* [online]. codebulb.ch, 2015-05-17 [Zugriff am 2015-12-31]. Verfügbar im World Wide Web unter: <http://www.codebulb.ch/2015/05/restful-jsf-with-post-redirect-get-part-1.html>.

WEB.FIELDING.ET.AL.HTTP: FIELDING, R., GETTYS, J. und MOGUL, J., et al. *HTTP/1.1: Method Definitions* [online]. w3.org, 1999 [Zugriff am 2015-12-31]. Verfügbar im World Wide Web unter: <http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html>.