Friedrich-Alexander-Universität Erlangen-Nürnberg
Technische Fakultät, Department Informatik

MARKUS WENZEL

BACHELOR THESIS

EXTEND AND INTEGRATE A VISUAL EDITOR INTO THE SWEBLE WIKI

Eingereicht am 30.09.2015

Betreuer:
Dipl.-Inf. Hannes Dohrn
Prof. Dr. Dirk Riehle, M.B.A.
Professur für Open-Source-Software
Department Informatik, Technische Fakultät
Friedrich-Alexander University Erlangen-Nürnberg

Versicherung

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

_____

NÜRNBERG, 30.09.2015

License

_____

NÜRNBERG, 30.09.2015

Abstract

The Sweble Wiki is a Wikipedia like implementation for the usage of a format called Wiki Object Model (WOM). This format helps to structure wiki articles that are stored in text blobs. The Sweble Wiki offers two kinds of editing components: a plain text editor which works with MediaWiki Markup and a XML editor. While MediaWiki developed a visual editing extension, the Sweble Wiki misses this possibility. This thesis covers implementations of word processors. It also develops and integrates a visual editor to the current Sweble Wiki which offers the possibility to create and edit WOM formatted articles in a way of “What-You-See-Is-What-You-Get”.

Zusammenfassung

Das Sweble Wiki ist eine Wikipedia ähnliche Implementierung für die Verwendung des Wiki Object Model (WOM) Formats. Dieses Format hilft der Strukturierung von Wiki-Artikeln die in Text Blobs gespeichert werden. Das Sweble Wiki zwei Arten von Bearbeitungskomponenten an: einen reinen Text Editor der mit MediaWiki Markup Texten arbeitet und einen XML-Editor. Während MediaWiki eine Erweiterung zur visuellen Verarbeitung entwickelt, vermisst das Sweble Wiki so eine Möglichkeit. Diese Arbeit beschäftigt sich mit Implementierungen von Textverarbeitungsprogrammen. Ebenfalls wird ein visueller Editor entwickelt und in das derzeitige Sweble Wiki integriert welches die Möglichkeit bietet WOM formatierte Artikel in einer „Was-Du-Siehst-Ist-Was-Du-Bekommst“ Art zu erzeugen und zu editieren.

Inhaltsverzeichnis

1 Einführung

Das Sweble Wiki ist eine Wikipedia ähnliche Implementierung für die Verwendung des Wiki Object Model (WOM) Formats. Dieses Format hilft der Strukturierung von Wiki-Artikeln die in Text Blobs gespeichert werden. Das Sweble Wiki zwei Arten von Bearbeitungskomponenten an: einen reinen Text Editor der mit MediaWiki Markup Texten arbeitet und einen XML-Editor. Während MediaWiki eine Erweiterung zur visuellen Verarbeitung entwickelt, vermisst das Sweble Wiki so eine Möglichkeit.

Zu den Zielen dieser Arbeit gehörte es die bereitgestellte Komponente auf die neueste Version zu aktualisieren. Diese in das Sweble Wiki System integriert werden und eine Kommunikation zwischen einander ermöglichen. Ebenso sollten Dialogbasierte Refaktorsierungen und Transformationen bereitgestellt werden, die dem Server Daten liefern mit denen diese Operationen angewendet werden können. Diese waren unter anderem das setzen von Links mithilfe von Auto-Completion Techniken und das einbetten von Bildern mithilfe eines Dialog zur Positionierung des Bildes. Weiterhin soll der visuelle Editor den WOM für das Wikitext Markup verstehen um damit die Verarbeitung von Artikeln anzubieten und diese in eine WOM Repräsentation umwandeln zu können. Schließlich sollte eine Dokumentation entstehen mit deren Hilfe man weitere WOM Knoten in eine editierbare Form bringt und diese innerhalb des visuellen Editor intuitiv bearbeiten kann genauso wie eine Dokumentation zum Modularen System und dessen Nutzung.

Der erste Teil der Arbeit befasst sich mit der Recherche und Analyse relevanter Arbeiten um das Thema visuelles Editieren von Wiki-Artikeln. Darunter fällt die Beschreibung des Sweble Wiki und des WOM Formats. Im Anschluss werden aktuell gültige Forschungen und Technologien ermittelt. Dies sind zum einen Entwurfsmuster für die Entwicklung einer visuellen Editor Komponenten. Zum anderen werden aktuelle Trends aus dem Bereich Webentwicklung betrachtet. Im Anschluss wird die bestehende Komponente Analysiert und die Funktionsweise Beschrieben.

Auf diesen Erkenntnissen aufbauend wird die Fragestellung der Forschung dieser Arbeit aufgestellt. Aus diesen werden die Anforderungen für die Lösung abgeleitet. Im Anschluss wird eine Lösung modelliert, implementiert und ausgewertet sowie ein Ausblick zur weiteren Forschung aufgezeigt.

Der zweite Teil befasst sich mit der Ausarbeitung der Forschung. Hier wird die implementierte Lösung detaillierter erörtert. Es werden Hinweise zu dem Aufbau des visuellen Editor Moduls gegeben. Ebenso wird die serverseitige Komponente näher betrachtet. Zu Schluss werden dann noch Implementierungen von Erweiterungen beispielhaft erarbeitet, um den Einstieg in das System, für zukünftige Entwickler, zu erleichtern.

2 Forschung

2.1 Einleitung

Wikipedia gehört zu den größten Webseiten im World Wide Web. Monatlich besuchen mehr als 45 Millionen Personen den internationalen Auftritt der Online-Enzyklopädie (Quantcast, 2015). Pro Jahr verzeichnet die Wikimedia Foundation einen Artikelzuwachs von ca. 150.000 Artikeln allein auf der deutschen Ausgabe (Wikipedia, 2015). Die so entstandenen Datenmengen werden im MediaWiki, der Software hinter Wikipedia, in Text Blobs gespeichert.

Maschinen haben es jedoch schwer diese unstrukturierte Zeichenkette zu verarbeiten. So sind Refaktorsierungen und Transformationen mit einem hohen Aufwand verbunden. Um dieses Problem anzugehen haben Dohrn und Riehle 2011 den Sweble Wiki Parser vorgestellt. Im Gegensatz zu gängigen Lösungen, die auf regulären Ausdrücken oder unpräzisen Parser Methoden basieren, bietet der Sweble Parser an Wikitext Markup, mithilfe von kontextsensitiven Grammatiken, in ein strukturierten abstrakten Syntaxbaum umzuwandeln (Riehle & Dohrn, 2013).

Das Sweble Wiki ist eine Wiki Webanwendung, die auf den Erkenntnissen aus dem Parser aufbaut. Derzeit bietet sie Funktionalitäten zum Speichern und wiedergewinnen von Artikeln an. Ebenso kann man Artikel mithilfe von Wikitext Markup oder XML bearbeiten.

Jedoch birgt diese Art der Bearbeitung eine Gefahr von denjenigen Redakteuren „monopolisiert zu werden die gewillt sind ihre Zeit in die Erlernung von Wikitext“ zu investieren (MediaWiki, 2015b). Aus diesem Grund wurde bei MediaWiki die Entwicklung einer visuellen Editor Komponente begonnen. Sie bietet eine Alternative für das einfache editieren von Wikipedia Artikeln, da noch nicht alle Features von Wikitext implementiert sind. Solch eine Editor Komponente, die es dem Nutzer erlaubt in einer „Was-Du-Siehst-Ist-Was-Du-Bekommst“ Art und Weise, Artikel zu bearbeiten fehlt dem Sweble Wiki derzeit gänzlich.

Zur Lösung dieses Problems wurde bereits eine Editor Komponente entwickelt (Haase, 2014). Da zu dieser Zeit noch kein reales Sweble Wiki System existierte wurde eine Komponente für den Sweble Parser geschrieben die die Bearbeitung von Artikeln erlaubte. Als visuellen Editor wurde dabei die Standalone Form des MediaWiki VisualEditors gewählt. Ferner wurde eine NodeJS Applikation entwickelt die sich um die Bereitstellung des visuellen Editors kümmert und die Hin- und Rücktransformationen von Wiki Markup in JavaScript Objekten anbietet. Hierfür wurde der Versuch unternommen die damalige Implementierung des WOM in JavaScript Klassen abzubilden und auf deren Basis die jeweiligen Konvertierungen vorzunehmen. Die Speicherung der Artikel übernimmt dabei eine MediaWiki Instanz die die Artikel im konvertierten MediaWiki Markup hält.

Das sich die Basistechnologien dieser zwei Komponenten unterscheiden muss die damalige Entwicklung analysiert, bewertet und auf die neuen Gegebenheiten angepasst werden.

2.2 Related Work

2.2.1 Sweble Wiki

Das Sweble Wiki ist eine J2EE Web Anwendung das von der Open Research Group an der Friedrich-Alexander-Universität Erlangen-Nürnberg entwickelt wird. Der Kern des ganzen ist das von Dohrn und Riehle beschriebene Wiki Object Model (Dohrn & Riehle, 2011). Dieser

Kern lässt sich Modular erweitern. So stehen für die Serialisierung des WOM unterschiedliche Transformer bereit. Für den Endanwender des Sweble Wiki wurde das Wicket-Framework[1] eingesetzt.

Abbildung1: PlainText und XML Editoren des Sweble Wiki

2.2.2 Relevante technologische Entwicklungen

Kollaboratives Schreiben

Das Sweble Wiki soll die Möglichkeit bieten unterschiedlichen Nutzern gleichzeitig die Bearbeitung der Artikel zu ermöglichen. Dabei kann es auch vorkommen das Änderungen von einem Benutzer an einem Artikel durch einen anderen Benutzer überschrieben werden, da die jeweils ursprünglichen Ressourcen diese Änderungen nicht enthalten. Dies ist auch ein gängiges Problem in der Softwareentwicklung wenn am Code von unterschiedlichen Entwicklern gearbeitet wird (Spinellis, 2005).

Eine Lösung für dieses Problem in der Softwareentwicklung sind Versionsverwaltungen wie Subversion, Git oder CVS. Dabei wird der Stand des jeweiligen Dokumentes oder Codes mit einer Laufvariablen versehen und bei der Übergabe an das System überprüft ob eine neuere Version vorliegt. Ist dies der Fall wird der Nutzer darauf hingewiesen das eine neuere Version vorliegt und er wird gebeten seine Änderungen in die neue Version einzubringen. Dieser Vorgang wird in der Softwareentwicklung üblicherweise „merge“ genannt (Collins-Sussman, Fitzpatrick, & Pilato, 2004).

Im Sweble Wiki wird eine ähnliche Form bereits durch Commits realisiert. Anders als bei der Versionsverwaltung wird dem User hier kein Konflikt gegeben den es zu lösen gilt. Es werden

1 http://wicket.apache.org/

Änderungen an einem Artikel eindeutig identifiziert und sollen bei Bedarf Rückgängig gemacht werden können.

Es gibt Entwicklungen, die es ermöglichen dieses Problem in Echtzeit zu lösen. Zu den bekanntesten Lösungen zählt dabei das von Google entwickelte Google Docs[2]. Diese Produktfamilie bietet es an mehreren Benutzern simultan an einem Dokument zu arbeiten. Hierbei wird in Echtzeit den Nutzern die aktuelle Cursorposition aller anderen Nutzern mit angezeigt. Somit können die Nutzer Textstellen erkennen, die derzeit von anderen Nutzern bearbeitet werden, was es erleichtert Konflikte gar nicht erst auftreten zu lassen. Diese Funktion wird in den Produkten Google Docs, Google Sheets und Google Slides angeboten. Zu der bekanntesten Open-Source Lösung zählt hier Etherpad Lite[3]. Etherpad Lite ist eine Neuentwicklung, basierend auf dem von Google veröffentlichten Kernbibliotheken, von EtherPad[4] [5].

Auch MediaWiki hat für ihre VisualEditor Extension so eine Möglichkeit vorgesehen, die derzeit schon parallel zum Kernentwicklerteam von Freiwilligen auf Basis von Etherpad Lite entwickelt wird (MediaWiki, 2014).

2.2.3 Untersuchung der aktuellen VisualEditor Komponente

Wie bereits in der Einleitung erwähnt gab es schon Bestrebungen seitens des Lehrstuhls einen visuellen Editor anzubieten. Dieser Aufgabe hat sich Michael Haase angenommen und ein System entwickelt mit dem es möglich war Wiki Artikel, auf Basis eines in WOM umgewandelten Wikitext, zu bearbeiten und speichern zu können. Da sich die Aufgabenstellung seiner Arbeit nicht stark von dieser unterscheidet möchte ich im folgenden das erstellte System analysieren und nach Möglichkeit für die Wiederverwendung bewerten.

MediaWiki VisualEditor Extension

Auf der Clientseite hat sich Haase dazu entschlossen die Standalone Variante des VisualEditor Extension des MediaWiki Paketes zu verwenden. Der VisualEditor ist ein Rich-Text-Editor Modul das derzeit schon aktiv in der Wikipedia angeboten wird und es Artikel Redakteuren erleichtern soll Inhalte zu erstellen und zu bearbeiten. Das Modul setzt dabei auf ein Model-View-Controller Muster um die Anzeige von der zugrundeliegenden Datenstruktur zu trennen (MediaWiki, 2015b).

Die MediaWiki VisualEditor Extension wird unter der MIT Lizenz veröffentlicht. Um Konflikte mit anderen Lizenzen zu vermeiden werden ebenso nur mit der MIT Lizenz kompatible Fremdbibliotheken, wie jQuery oder OOjs, eingesetzt. Innerhalb des Zeitraums vom 27ten August bis zum 27ten September wurden 101 Commits in das Versionskontrollsystem übertragen. Daran beteiligt waren 13 Autoren. Insgesamt haben an diesem Modul 73 Entwickler gearbeitet[6].

2 https://www.google.de/intl/de/docs/about/
3 http://etherpad.org/
4 http://etherpad.org/#thanks
5 https://github.com/ether/etherpad-lite/wiki/FAQ
6 https://github.com/wikimedia/mediawiki-extensions-VisualEditor/pulse/monthly (Aufgerufen am 27.09.2015, 19:32 Uhr)

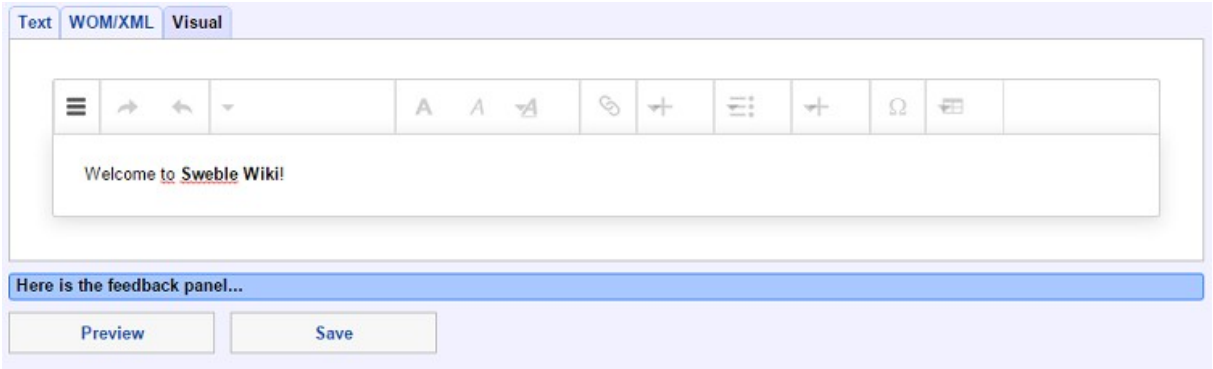

Abbildung 1: Sweble Wiki VisualArticleEditor

Sweble Parser

Da die MediaWiki VisualEditor Extension nur mit HTML Dokumenten umgehen kann, wurde von MediaWiki ein spezieller Wikitext Parser namens Parsoid[7] entwickelt. Dies ist eine NodeJS Komponente die Wikitext in die von MediaWiki beschriebene HTML/RDFa Repräsentation umwandelt und an den VisualEditor ausliefert. Aufgrund der Tatsache, dass Parsoid nur Wikitext in HTML und HTML in Wikitext umwandelt, wurde die Verwendung eines Solchen Parsers obsolet. Zum Einsatz für die Verwendung von WOM wurde der von Riehle und Dohrn entwickelte Sweble Parser eingesetzt. Der Sweble Parser ist eine Anwendung die es erlaubt Wikitext Markup in einen abstrakten Syntaxbaum zu transformieren (Riehle & Dohrn, 2013). Dieser bietet Schnittstellen an um den entstandenen Baum in ein JSON Format konvertieren zu können. Ebenso bietet er eine Rücktransformation des JSON in Wikitext Markup an.

NodeJS App

Die von Haase entwickelte NodeJS App stellt eine Webserver Komponente bereit, die eine REST Schnittstelle anbietet, über die Artikel geholt und gespeichert werden können. Ebenso stellen sie Aufrufe auf den Sweble Parser bereit, der geholtes Wikitext Markup in die bereits erwähnte JSON Repräsentation überführt. Analog dazu findet auch das Speichern in der gleichen Art und weise statt.

WOM3Tools

Die WOM3Tools sind eine Sammlung an unterschiedlichsten Komponenten. Die wohl wichtigste Komponente ist eine Abbildung des bis dahin bekannten WOM in JavaScript. Diese bietet ebenso wie das von Dohrn und Riehle beschriebene WOM die wichtigsten Funktionen des WOM für die Bearbeitung dessen an. Ebenso beinhalten die WOM3Tools Konverter Klassen. Diese wandeln HTML Dokumente in einen WOM um, damit diese später persistiert werden können. Ebenso bieten sie die Konvertierung von WOM in HTML um um diese ausliefern zu können. Ferner stellen die WOM3Tools Schnittstellen bereit, die mithilfe des Sweble Parsers zum einen Wikitext in eine WOM Repräsentation bringen und zum anderen bearbeitete WOM Bäume wieder zurück in Wikitext Markup konvertieren.

7 https://www.mediawiki.org/wiki/Parsoid

MediaWiki Server

Um Artikel abrufen und speichern zu können hat sich Haase dafür entschieden eine MediaWiki Installation für diese Aufgaben aufzusetzen. Für diese Zwecke hat er die NodeJS App um Services, für den Zugriff auf die MediaWiki API, erweitert. Um diese Schnittstellenaufrufe durchführen zu können hat er die „nodemw“ Bibliothek benutzt.

2.3 Fragestellung

Auf der Recherche und Analyse bisheriger Forschungen und Entwicklungen aufbauend stellt sich für die Erarbeitung einer passenden Lösung folgende Forschungsfragen die es im weiteren Verlauf der Arbeit zu beantworten gilt:

- Wie kann der VisualEditor in das bestehende Sweble Wiki integriert werden?
- Welche Komponenten und Schnittstellen müssen geschaffen werden, um ein WOM an den VisualEditor ausliefern zu können?
- Welche Komponenten und Schnittstellen müssen geschaffen werden, um bearbeitete oder neue Dokumente in ein WOM umwandeln zu können?
- Welche Anpassungen sind nötig um mit dem VisualEditor WOM exklusive Elemente bearbeiten zu können?

2.4 Anforderungen

2.4.1 Funktionale Anforderungen

Intuitive Bedienbarkeit

Textverarbeitungsprogramme unterscheiden sich größtenteils in ihrem angebotenen Funktionsumfang den sie für die Bearbeitung von Dokumenten anbieten. Die meisten von ihnen bieten jedoch bekannte Buttons, Symbole und Orte an denen sich gängige Funktionen finden lassen. Das markanteste Detail ist dabei die „Toolbar“ in der sich Funktionen wie Formatierungshilfen, Formatvorlagen und Funktionen für die Einfügung von Objekten wie Bilder und Tabellen finden lassen.

Somit wird von der Anwendung erwartet sich ebenso einfach wie gängige Textverarbeitungsprogramme bedienen zu lassen.

Browserunabhängigkeit

Die Standards für HTML Dokumente und CSS Stylesheets werden von der W3C entwickelt und festgelegt, genauso wie JavaScript von Ecma International. Trotzdem unterscheiden sich die Implementationen teilweise stark von den Browserherstellern. Derzeit dominieren mit knapp 2/3 der Marktanteile die Browser Chrome, Internet Explorer und Firefox (Statista, 2015).

Es wird von der Anwendung erwartet browserunabhängig zu sein. Da dies eine große Anforderung darstellt wird sie auf die neuesten Versionen der Desktopvarianten der oben genannten beschränkt. Im speziellen sind das Chrome 45, Internet Explorer 11 und Firefox 41.

Erweiterbarkeit

Zwar wurde das WOM bisher für die Abbildung von MediaWiki Markup entwickelt, jedoch sieht die Implementieren des WOM vor weitere Knoten in Zukunft entwickeln zu können. Einer dieser Möglichkeiten wäre die Erweiterung um BBCode und dessen Erweiterungen. Somit ist es derzeit nicht absehbar, welchen Umfang das WOM in Zukunft haben wird.

Deshalb ist wird von der Anwendung erwartet, dass es sich erweitern lässt um zukünftige Entwicklungen des WOMs zu berücksichtigen.

Lokalität

Änderungen an einem Dokument können sich auf unterschiedliche Arten auf eben jenes auswirken. Einerseits kann ein vollständig neues Dokument ohne jegliche Metadaten erzeugt werden. Andererseits könnte innerhalb des bestehenden Dokuments Teile direkt ersetzt und somit Metadaten, wie nicht sichtbare Attribute oder Informationen über den Autor, erhalten bleiben.

Innerhalb des WOM werden dazu derzeit schon einige Knoten erzeugt die solch eine Art von Informationen enthalten. Ein Beispiel wäre die Round-Trip-Data für die Serialisierung eines WOM Baumes in ein gegebenes Markup Format wie beispielsweise dem Wikitext Markup.

Aus diesem Grund wird von der Anwendung erwartet, dass Änderungen am Dokument sich nur auf relevante WOM Knoten auswirkt und die Änderungen damit lokal sind.

2.4.2 Nicht funktionale Anforderungen und Randbedingungen

Wartbarkeit

Die Entwicklungsarbeit in großen Softwareprojekten werden von mehreren Entwicklern geleistet. Damit sowohl die Kosten als auch die Zeit zur Weiterentwicklung möglichst niedrig bleiben ist sollte die zu entwickelnde Software ordentlich dokumentiert werden. Diese soll neue Entwickler dabei unterstützen sich leichter in die Problemdomäne einzuarbeiten und damit Releasezyklen einzuhalten. Da auch die visuelle Editor Komponente in Zukunft weiterentwickelt und ausgebaut werden soll ist auch eine verständliche Dokumentation eine der Anforderungen an die Anwendung.

Lizenzkompatibilität

Sowohl das Sweble Wiki als auch die Komponente des visuellen Editors wird unter der Apache Lizenz in der Version 2.0 veröffentlicht. Aus diesem Grund ist es zwingend erforderlich nur Komponenten, Module und Pakete zu verwenden, die mit dieser Lizenz kompatibel sind. Eine Auflistung der kompatiblen[8] und inkompatiblen[9] Lizenzen findet sich auf den Webseiten der Apache Software Foundation.

Java & HTML/CSS/JavaScript als Basistechnologien

Die Entwicklung von Sweble basiert derzeit auf Java unter Zuhilfenahme von Wicket als Webframework. Aus diesem Grund sollte sich die Komponente so einfach wie möglich in die bestehende Architektur einbringen können. Deshalb wird für die Serverseitige Auslieferung

8 http://www.apache.org/legal/resolved.html#category-a
9 http://www.apache.org/legal/resolved.html#category-x

des zu bearbeiteten Dokumentes Java verwendet während der Client mit den von Browsern gängigen Technologien HTML/CSS/JavaScript realisiert werden soll.

Maven Projekt zur besseren Integration in das Sweble Wiki

Das Sweble Wiki wird derzeit modular in Maven Projekten entwickelt um die Bereitstellung aller relevanten Dateien und die Übersetzung zu vereinfachen. Ebenso soll der hier entwickelte visuelle Editor in die gegebene Projekt Landschaft eingefügt werden um damit Kongruent zu der bisherigen Entwicklung zu sein.

2.5 Architektur und Design

2.5.1 Komponenten

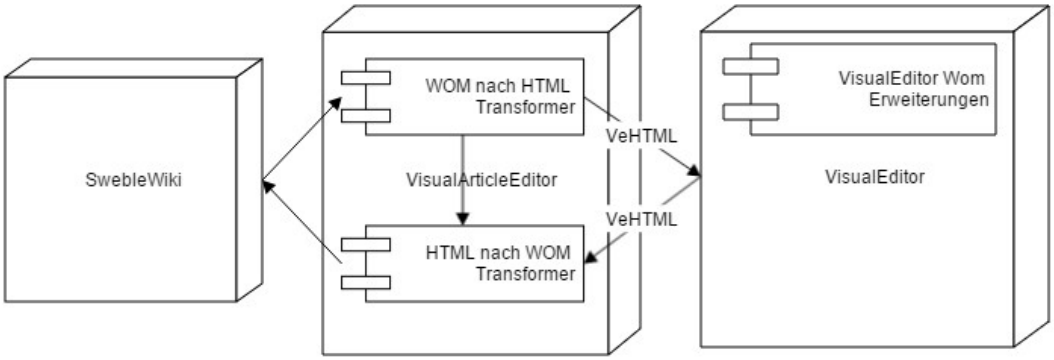

Abbildung 2: Komponenten-Diagramm

Sweble Wiki

Das Sweble Wiki stellt eine Webserver Anwendung dar. Bei Aktivierung des visuellen Bearbeitungsmodus liefert diese den WOM den es zu bearbeiten gilt aus. Ebenso sendet er die nötigen Skriptdateien und Stylesheets an den Browser aus. Nach der Bearbeitung eines Artikels nimmt er den erzeugten WOM entgegen und bietet die Möglichkeit an den bearbeiteten Artikel zu persistieren.

Modul des Visuellen Artikel Editors

Das Sweble Wiki ist modular aufgebaut. Ebenso wie für den Text oder XML Editor wurde für den visuellen Editor ein Modul erstellt in dem alle erforderlichen Klassen und Ressourcen gebündelt werden sollen. Diese sind die nachfolgenden Transformerklassen ebenso wie die Skriptdateien und Stylesheets die an den Client ausgeliefert werden müssen.

WOM nach HTML Transformer

Der „WOM nach HTML Transformer“ soll sich um die Serialisierung eines WOM kümmern. Dabei werden die einzelnen Knoten eines WOM Baumes mithilfe eines Visitor Musters iterativ besucht und in ein für den VisualEditor verständliches Format gebracht. Um die Kernentwicklung des ViusalEditors nicht gänzlich zu ersetzen werden vornehmlich bekannte HTML Tags verwendet, während für WOM exklusive Knoten eigene Tags geschaffen werden. Ebenso stellt er eine Schnittstelle bereit um WOM Knoten im Baum eindeutig zu identifizieren.

Hierfür wird eine Map erstellt die jeden Knoten mit einer ID versieht. Dies soll später helfen die Lokalität der Änderungen zu wahren, in dem nicht geänderte Knoten wieder mit ihren alten Informationen erzeugt werden sollen.

HTML nach WOM Transformer

Der „HTML nach WOM Transformer“ beschreibt das Gegenstück zum „WOM nach HTML Transformer“. Dieser parst das bearbeitete Dokument des VisualEditor in eine Document Object Model (DOM) Repräsentation um einen WOM Artikel daraus erzeugen zu können. Ebenso soll darin die Änderungen am Original Artikel verglichen werden und Änderungen lokal in das Dokument eingebracht werden. Hierfür nutzt die Komponente die beschriebene Schnittstelle des „WOM nach HTML Transformer“ um ein identifizierbaren WOM Baum zu erhalten. Mit dessen Hilfe können die Änderungen lokal bewerkstelligt werden.

VisualEditor

Der VisualEditor ist die Standalone Variante der MediaWiki VisualEditor Extension. Um ihn an den Browser des Nutzers senden zu können wird der VisualEditor als Komponente bereitgestellt um ihn auf verschiedene Seiten einbetten zu können. Die genutzte Version soll die aktuellste des Gerrit-Repositories von MediaWiki sein[10].

VisualEditor Wom Erweiterungen

Zwar bringt der VisualEditor von Haus aus eine gelungene HTML Unterstützung mit, jedoch mangelt es ihm an den WOM exklusiven Knoten. Diese sind beispielsweise WomExtLink und WomIntLink Knoten. An diesem Punkt muss angesetzt werden und der VisualEditor muss um solche Komponenten erweitert werden. Konkret gehört die Bereitstellung von Datenmodellen und Oberflächendarstellungen dazu, ebenso wie die Erstellung von Widgets, Tools und Inspectors zum bearbeiten der Eigenschaften der Knoten.

VeHTML

Das VeHTML ist die Repräsentation eines WOM für den VisualEditor. Dies ist das Austauschformat mit dem der VisualEditor die Bearbeitung anbieten kann. Bereitgestellt wird sie vom „WOM nach HTML Transformer“. Um sie wieder in eine WOM Repräsentation zurück zu transformieren nutzt man dafür den „HTML nach WOM Transformer“.

10 https://gerrit.wikimedia.org/r/#/c/240243/

2.5.2 Datenfluss

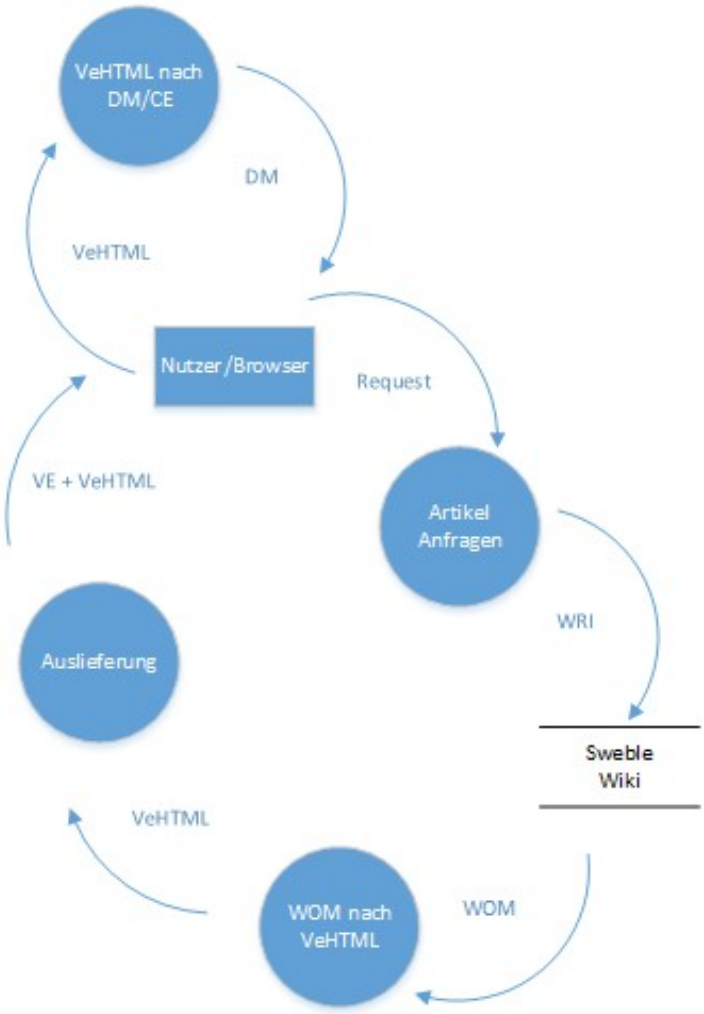

Abbildung 3: Datenfluss-Diagram zur Abfrage von Artikeln

Nutzer/Browser (Terminator)

Bei der Nutzung des Systems spielt der Nutzer die wichtigste Rolle. Er bestimmt durch seine Aktionen den Datenfluss. Dies geschieht durch das Navigieren auf die Artikelseiten und dem Klicken des Links welches ihm die Bearbeitung erlaubt. Sollte der VisualEditor für die Bearbeitung aktiviert werden sendet eine Anfrage an das Sweble Wiki. Nach dem Abschluss aller relevanten Prozesse liegt ein bearbeitbarer Artikel im Browser gerendert vor. Ab diesem Zeitpunkt kann er Änderungen am Dokument vornehmen und abschließend das Dokument speichern oder die Vorschau aktivieren. In beiden Fällen wird eine Anfrage mit dem bearbeiteten Dokument an den Server geschickt wo er wieder in ein WOM transformiert und je nach Kontext persistiert oder nur zum anzeigen gerendert wird.

Artikel anfragen (Prozess)

Dieser Prozess wird gestartet sobald sich der Nutzer entscheidet einen Artikel mithilfe des VisualEditors zu bearbeiten. Hierbei wird ein WOM Artikel aus dem Wiki angefragt welches bearbeiten werden soll. Diese Anfrage beinhaltet die WRI des Artikels und wird vom Sweble Wiki verarbeitet.

Sweble Wiki (Datenspeicher)

Das Sweble Wiki stellt eine Datenbank zur Verfügung, die alle persistierten WOM Artikel enthält. Entscheidet sich der Nutzer nun einen Artikel bearbeiten zu wollen so benötigt das Sweble Wiki die WRI eines Artikels und liefert dafür den WOM zurück. Dieser wird im Anschluss einem Prozess übergeben der ihn in eine für den VisualEditor verständliche Form überführt.

WOM nach VeHTML (Prozess)

Im Anschluss startet der Prozess zum aufbereiten des übergebenen WOM in ein HTML Format das der VisualEditor interpretieren kann. Das Ergebnis ist ein HTML Dokument das an den Prozess der Auslieferung übergeben wird.

Auslieferung (Prozess)

Die Auslieferung übernimmt das übergebene VeHTML und liefert es inklusive der nötigen Skriptdateien und Stylesheets an den Nutzer aus. Nach dem diese Daten an den Client geschickt wurden müssen sie vorher noch in das vom VisualEditor verständlichen View und Model Format umgewandelt werden.

VeHTML nach DM/CE (Prozess)

Damit der VisualEditor mit dem übergebenen VeHTML Dokument arbeiten kann muss dieses vorher in interne Repräsentationen des Dokumentes umgewandelt werden. Genauer gesagt sind das die DataModels der einzelnen DOM Tags. Der VisualEditor stellt dafür Methoden zur Konvertierung bereit. Intern werden parallel zu jedem erzeugen DataModel eine ContentEditable Repräsentation erzeugt um dem Nutzer das „Was-Du-Siehst-Ist-Was-Du-Bekommst“ Erlebnis bieten zu können.

DM nach VeHTML (Prozess)

Ist der Nutzer fertig mit der Bearbeitung des Dokumentes muss das zugrundeliegende DataModel wieder in eine Transportfähige Repräsentation gebracht werden. Dafür werden die Eigenschaften des DataModels wieder in HTML Tags verpackt um ein VeHTML Dokument zu erzeugen. Dieses kann anschließend an den Server übertragen werden.

VeHTML nach WOM (Prozess)

Erhält der Server nun das VeHTML inklusive einer WRI, welches auf den zugrundeliegenden Artikel verweist, sowie eine Beschreibung der Aktion die er erledigt haben will, soll eine Konvertierung von VeHTML nach WOM geschehen um den daraus resultierenden WOM als HTML aufbereiten zu können um dem Nutzer das Feedback geben zu können wie das Endre-

sultat seiner Bearbeitung aussehen wird. Dies geschieht sowohl dann wenn sich der Nutzer für die Vorschau entscheidet, als auch dann wenn er sich für das Persistieren des Artikels entscheidet.

Anzeigen (Prozess)

In jedem Fall wird dem Nutzer eine Ansicht des Artikels gewährt, sollte er sich dafür entscheiden diesen speichern zu wollen oder nur überprüfen wollen, wie sich seine Änderungen auf den Artikel auswirken würden. Für beide Fälle wird ein erzeugtes HTML Dokument aus dem WOM erzeugt, die dem Nutzer ausgeliefert wird um daraufhin die Ansicht im Browser rendern zu können.

Persistieren (Prozess)

Um geänderte Artikel auch anderen Nutzern zugänglich zu machen müssen diese wieder auffindbar gespeichert werden. Dafür bietet das Sweble Wiki Schnittstellen zum persistieren an. Beim persistieren wird dem Sweble Wiki das zuvor erzeuge WOM übergeben um es in der Datenbank abspeichern zu können.

2.6 Implementation

Zu Beginn dieser Arbeit wurde ein Maven Projekt für den VisualArticleEditor erzeugt innerhalb dem folgende Implementationen realisiert wurden:

VisualArticleEditorComponent

Das VisualArticleEditorComponent stellt das DOM-Fragment für den Browser bereit. Hier werden alle relevanten Stylesheets und JavaScript Dateien ausgeliefert, die Textarea Transitzone befüllt und ausgelesen, die Transformationen angestoßen und der erzeugte Baum bereitgestellt.

ArticleResourceToVeHtmlTransformer

Damit der VisualEditor von MediaWiki einen Artikel bearbeiten kann wurde ein Transformer auf Basis des Visitor Design Pattern entwickelt um die Umwandlung eines WOM Baumes auf ein HTML-Fragment erledigen. Dabei werden alle WOM Knoten in einer HashMap mit einem aufsteigenden Integer-Schlüssel gespeichert. Ebenso wurde an jedes HTML-Tag ein „data-wom-id“ mit diesem Schlüssel erzeugt um bei der Rücktransformation bisherige Round-Trip-Daten wieder zu verwenden um die Lokalität von Änderungen zu wahren.

VeHtmlToArticleResourceTransformer

Die Rücktransformation des HTML Modells wird im VeHtmlToArticleResourceTransformer vorgenommen. Um den HTML-Text dabei leichter verarbeiten zu können wird das HTML-Fragment mithilfe des Jsoup Parsers in ein Document Object Model umgewandelt und im Anschluss rekursiv in ein WOM transformiert. Hier wird auch die zuvor erzeugte „data-wom-id“ ausgelesen. Mithilfe dieser können dann die Round-Trip-Daten der alten an die neuen WOM Knoten angehängt werden.

Erweiterungen am VisualEditor

Der VisualEditor wurde um ein „wom:intlink“ Knoten erweitert. Hierbei wurde darauf geachtet modular zum VisualEditor vorzugehen um spätere Updates leichter einbringen zu können. Ebenso wurde ein „BlockImage Inspector“ implementiert, der es dem Nutzer ermöglichen soll Bilder in das Dokument einzubetten. Hierbei wurde nur die Möglichkeit implementiert externe Bilder in den Artikel einbringen zu können. Um auch interne Bilder zu verwenden kann auf den Erkenntnissen aus dem „wom:intlink“ Knoten aufgebaut werden.

2.7 Ergebnis

Mit der derzeitigen Implementierung kann das VisualArticleEditor Modul einige WOM Knoten in eine editierbare Form bringen. So kann es mit internen und externen Links umgehen und diese auch an den VisualEditor übergeben und bearbeiten. Weiterhin modelliert es auch Bilder des WOMs ordentlich und beherrscht nahezu alle Textannotationen. Ebenso wurde bei der Rücktransformation dafür gesorgt, dass Änderungen am Dokument keinen Verlust der Round-Trip-Daten eines Knotens bewirken. Somit ist die Lokalität von Änderungen gewahrt worden.

Der ViusalEditor wurde um die Darstellung und Bearbeitung von Internen Links erweitert. So lassen sich jetzt speziell ausgezeichnete „wom:intlink“ Knoten erzeugen, die auch die Möglichkeit der Auto-Vervollständigung anbieten. Zwar fehlt es hier an der erforderlichen Schnittstelle zum Sweble Wiki, jedoch ist bereits dafür vorgesorgt eine nutzten zu können. Dies würde sich auch auf die fehlenden Sprachdateien positiv auswirken, da diese zur Laufzeit durch einen ähnlichen Aufruf nachgeladen werden.

Die Ausrichtung eines Bildes wurde mittlerweile schon vom Entwicklerteam nachgereicht. Hierfür wurde dann ein Inspektor entwickelt, der es dem Nutzer möglich machen soll eben jene in das Dokument einzubetten um die Möglichkeit der Ausrichtung nutzen zu können.

Im Kapitel 3 werden, neben einer Ausführlichen Beschreibung des Moduls, Beispielhaft die nötigen Schritte zur Erweiterung des VisualArticleEditor Moduls beschrieben. Somit sollte es zukünftigen Entwicklern leichter fallen den Aufbau zu verstehen.

Leider zeigen sich bei dieser Implementation Schwächen in der Rücktransformation eines HTML Dokumentes in ein repräsentativen WOM. So ist es bisher nicht möglich sowohl interne als auch externe Links wieder in WOM Knoten umzuwandeln. Ebenso fehlt das Parsen von Tabellen. Auch fehlt die Implementation für das Parsen von Sektionen gänzlich.

2.8 Ausblick

Das aktuelle Artefakt liefert gute Ergebnisse um ein Dokument zu erzeugen und zu Bearbeiten. Zur Vollendung wäre es nötig die übrigen WOM-Knoten sowohl in den Transformern als auch im VisualEditor zu modellieren.

Weiterhin wäre es sinnvoll die von MediaWiki bereitgestellten Stylesheets an das Sweble Wiki anzupassen, da im Verlauf der Implementierung aufgefallen ist, dass einige der im VisualEditor verwendeten Komponenten sich auf die von Sweble Wiki bereitgestellten Stylesheets ableiten und damit nicht wie von einem Textverarbeitungsprogramm gewohnt aussehen.

Da die Implementierung des VisualEditor von MediaWiki bereits jetzt schon Änderungen an einem Dokument in Transaktionen nach dem Commando Prinzip hält, könnte man Schnittstellen schaffen um diese an den Server zu übermitteln und dort auszuwerten. Dieses Vorgehen

würde dabei zwei interessante Möglichkeiten schaffen. Zum einen würde der Overhead reduziert werden, wenn anstelle vom ganzen Dokument nur geänderte Teile an den Server übertragen werden. Zum anderen könnte man ein Feature schaffen um, wie in heutigen Textverarbeitungsprogrammen üblich, automatisch die letzten Änderungen zu speichern und, bei Abbruch der Verbindung mit dem Redakteur, eine Wiederherstellung anbieten zu können.

Da das Wicket-Framework die Auslieferung der Ressourcen bereitstellt, war es nicht ohne weiteres Möglich JSON Daten per AJAX-Request einzuholen. Deshalb wäre es sinnvoll REST Schnittstellen anzubieten, die beispielsweise die Bereitstellung der Sprachdateien oder die Ergebnisse für Auto-Completion übernimmt. Diese könnten ebenfalls dazu genutzt werden die bereits erwähnten Transaktionen zu empfangen und als Änderungen auf das bestehende WOM anzuwenden.

3 Ausarbeitung der Forschung

3.1 VisualEditor

3.1.1 Das Grunddesign

Das Design des VisualEditor basiert auf einem Model-View-Controller Muster welches bereits im Kapitel 2.2.5 erwähnt wurde. Dieses Muster beschreibt die Trennung von der Benutzeroberfläche zu den den zugrunde liegenden Datenstrukturen. Bei der Entwicklung des VisualEditors haben die Entwickler sehr darauf geachtet diesen leicht erweitern zu können. So stehen viele Fabrikmethoden bereit, die die Registrierung neuer Komponenten übernehmen. Ebenso wurde schon von Grund auf ein Transaktionsschema verwendet um Änderungen an einem Dokument leichter Rückgängig machen zu können und so auch Kollaboratives Schreiben ermöglichen. Danach teilt sich der VisualEditor in die drei Komponenten DataModel, ContentEditable und UserInterface auf.

Das DataModel repräsentiert die zugrundeliegenden Datenstrukturen auf denen ein übergebenes HTML Dokument abgebildet wird. Änderungen auf der Oberfläche werden durch Ereignisse und Methoden direkt in einem Transaktionsschema auf das DataModel angewendet um zu jedem Zeitpunkt einen konsistenten Zustand gewährleisten zu können. Das DataModel stellt ebenso Konverter bereit die HTML Dokumente in ein DataModel überführen und ebenso wieder zurück serialisieren können.

Das ContentEditable stellt für jedes DataModel eine Sicht für die Benutzeroberfläche bereit. Dabei werden die Eigenschaften des DataModels in HTML Elemente überführt um sie so dem Benutzer anzeigen zu können. Weiterhin stellt sie eine Ereignisverarbeitung bereit, die sich um die Änderung des DataModel kümmert.

Das UserInterface kümmert sich um Oberflächenelemente wie die Toolbar, Dialoge und Widgets. Weiterhin stellt es Aktionen bereit, die bei Aktivierung einer Schaltfläche sich auf das DataModel auswirken. Ebenso kümmert es sich um Kontextsensitive Darstellungen wie Tooltips beim fokussieren von bestimmten ContentEditable Knoten wie beispielsweise Links oder Kommentaren.

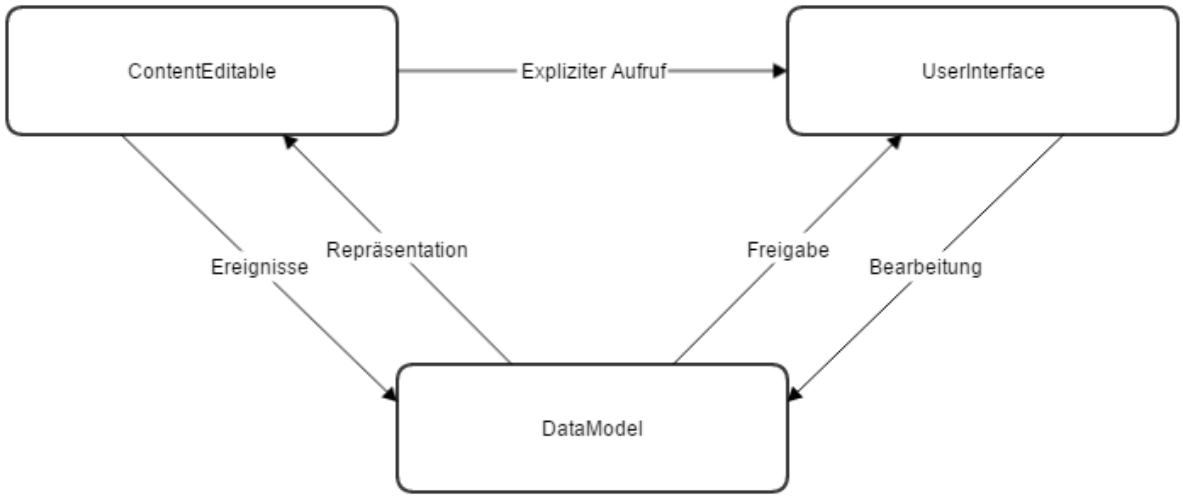

Abbildung 4: Beziehungen der Komponenten

3.1.2 DataModel & ContentEditable

Das VisualEditor DataModel ist ein lineares Datenmodell, welches das geparste HTML Dokument in ein linearen Adressraum überführt auf dem die Bearbeitung stattfinden kann. Für jedes Objekt das einem DataModel entspringt gibt es ein zugehöriges ContentEditable Objekt das sich um die Repräsentation auf der Anzeige dem User zur Bearbeitung bereitsteht. Diese übernimmt Oberflächenereignisse und leitet Kapselt diese in ein Transaktionsschema des DataModels. Im folgenden möchte ich auf besondere Komponenten des DataModels eingehen:

Nodes

Nodes repräsentieren die Dokument Knoten wie sie sowohl im DOM als auch im WOM gefunden werden. Darunter zählen Bilder (ve.dm.ImageNode), Paragraphen (ve.dm.ParagraphNode) oder Headings (ve.dm.HeadingNode).

Vorrangig gibt es nur zwei unterschiedliche Nodetypen. Zum einen gibt es die LeafNodes (ve.xx.LeafNode) die Nodes dahingehend klassifizieren, dass diese keine weiteren Kind-knoten beinhalten können. Ein Beispiel dafür wären CommentNodes (ve.xx.CommentNode) Zum anderen gibt es BranchNodes (ve.xx.BranchNode) die es im Gegensatz zu den LeafNodes erlauben Kind-knoten enthalten zu können. Weiterhin gibt es Abstrakte Klassen die es einem erlauben Knoten weitere Eigenschaften zu geben, wie beispielsweise die Fähigkeit sich ausrichten zu lassen (ve.xx.AlignableNode) oder die Größe zu ändern (ve.xx.ResizableNode).

Alle entwickelten Nodetypen müssen, um sie verwenden zu können, an der NodeFactory (ve.xx.NodeFactory) registriert werden.

Annotations

Annotationen wirken sich direkt auf TextNodes aus. Sie bearbeiten die Eigenschaften besonderer Textstellen. Beispiele sind hier die Annotation von Fett formatiertem Text (ve.xx.-BoldAnnotaion), Links (ve.xx.LinkAnnotation) oder Codeformatierung (ve.xx.CodeAnnotation).

Auch Annotationen werden wie Nodes in HTML Tags überführt. Jedoch wirken sie sich nicht direkt auf den Dokumentenbaum aus.

Die Annotationen müssen zur Verwendung in der AnnotationFactory (ve.xx.AnnotationFacotry) registriert werden.

DataModel Converter

Neben den Datenstrukturen des Dokumentes bietet die DataModel Komponente noch Konvertierungsklassen an, die es ermöglichen HTML Dokumente in ein DataModel zu überführen (ve.dm.Converter.getModelFromDom) und dieses nach den gemachten Änderungen wieder in ein HTML Dokument zu überführen (ve.dm.Converter.getDomFromModel).

3.1.3 UserInterface

Für das UserInterface stehen Komponenten wie Tools (ui/Tools), Widgets (ui/Widgets) und Inspectors (ui/Inspectors) bereit.

Die Tools sind die Registrierung eines Oberflächenelements an der Toolbar. Diese lassen sich in Gruppen zusammenfassen und bei bedarf einklappen. Beim Betätigen eines Toolbar Elements wird ein Commando ausgeführt das in der CommandRegistry registriert wurde. Ein solches Kommando könnte ein Inspektor aufrufen. Ein Inspektor ist ein Modales Fenster, das über ein ausgewählten Knoten erscheint. Dieses kann Details anzeigen, wie das Ziel eines Links, sowie für die Bearbeitung von Attributen genutzt werden. Innerhalb dieser Inspektoren kommen Widgets zum Einsatz. Diese Beschreiben Formulare, die von den Inspektoren genutzt werden können um Benutzereingaben zu validieren und weiter zu verarbeiten.

3.1.4 Bibliotheken

OOjs: Object-oriented JavaScript[11]

OOjs ist eine JavaScript Bibliothek die Entwickler dabei unterstützt Objekt-Orientiere Verhalten in der Skriptsprache verwenden zu können (MediaWiki, 2015a). Die drei Bausteine sind dabei die Vererbungsfunktionen, das Eventmanangement und Fabrikmethoden.

Die Vererbungsfunktionen sind:

```
OO.initClass( baseClass )
```

Diese Funktion ermöglicht es eine übergebenen Klasse ableitbar, mit den von OOjs bereitgestellten Methoden, zu werden.

```
OO.inheritClass( childClass, parentClass )
```

Diese Funktion ermöglicht es einer übergebenen Kind Klasse statische Methoden und Felder einer übergebenen Elternklasse zu nutzen ohne den Typ der geerbten Eigenschaften zu verlieren. Diese Funktion enthält in sich die weitere Bereitstellung der Klasse nach dem Prinzip der initClass Funktion. Somit sind Kind Klassen automatisch ableitbar wenn sie geerbt haben.

```
OO.mixinClass( childClass, parentClass )
```

Diese Funktion ermöglicht es einer übergebenen Kind Klasse statische Methoden und Felder einer übergebenen Elternklasse zu übernehmen und damit den Typ der geerbten Klasse vollständig zu verlieren. Auch bei dieser Funktion gilt die Bereitstellung der Kind Klasse nach dem in initClass beschriebenen Prinzip. Gemixte Klassen sind automatisch ableitbar.

OOjs UI[12]

OOjs UI ist eine Graphical User Interface Bibliothek ähnlich wie Motif, GTK+ oder Qt die auf der OOjs Bibliothek aufbaut. Sie stellt unterschiedliche Funktionen bereit wie beispielsweise Layouts, Widgets und Dialoge. Der VisualEditor arbeitet mit den mitgelieferten Themes und baut seine Inspektoren und Toolbars auf dessen Basis auf.

jQuery[13]

jQuery gehört zu den meist genutzten JavaScript Bibliotheken. BuiltWith gibt für den Einsatz von der Bibliothek eine Abdeckung von 71,6% der Quantcast Top Million an (BuildWith,

11 https://www.mediawiki.org/wiki/OOjs
12 https://www.mediawiki.org/wiki/OOjs_UI
13 http://jquery.com/

2015). Auf ein ähnliches Ergebnis kommt dabei W3Techs das bei den von ihnen beobachteten Seiten eine Abdeckung von 65,9% erkennt, was einem Marktanteil von 95,5% aller Bibliotheken entspricht (W3Techs, 2015). Für den ViusalEditor kommen noch die Extensions jQuery.Client[14], für die Erkennung des Browsers auf dem Client, und jQuery.i18n[15], für die Internationalisierung, zum Einsatz

Innerhalb des VisualEditors findet die Bibliothek weniger Bedeutung, jedoch nutzt der VisualEditor die OOjs Bibliothek in der jQuery Variante die auf jQuery aufsetzt.

3.1.5 VisualEditor interne Anpassungen

Da der VisualEditor durch seine Fabrikmethoden flexibel erweiterbar ist, sind wenige direkte Eingriffe nötig gewesen um den VisualEditor an die Anforderungen anzupassen. Die Erweiterungen werde ich in Kapitel 3.3 näher erläutern. Lediglich die bereitgestellte Oberfläche konnte größtenteils entschlackt werden. So war es nicht mehr nötig die voreingestellte Demo Toolbar anzuzeigen. Diese beinhaltete Sprachoptionen, das hinzuladen von Oberflächen sowie eine Vorschau. In Zukunft wird die Sprache, wie auch schon voreingestellt, vom lokalen System abhängig sein. Da das Sweble Wiki schon eine Vorschau anbietet konnte die bereitgestellte Vorschau aus dem Surface entfernt werden. Der entscheidende Eingriff war jedoch die Verbindung mit dem Eventhandling der Vorschau/Speicher Buttons. Um den zu editierenden Artikel vom Server zu bekommen als auch an diesen zu übermitteln wurde ein unsichtbares Textfeld angelegt das als Transit-zone verwendet wird.

```
<textarea wicket:id="html-model" id="html-model"></textarea>
```

Listing 1: Textarea Transit-zone (VisualArticleEditorComponent.html)

Die von der Standalone Variante angebotene loadHTML Methode wurde damit durch eine Methode zum auslesen der Textarea ersetzt.

```
var page = $('#html-model').val();
```

Listing 2: Laden des Dokumentes aus der Transit-zone (ve.demo.init.js)

Um dieses auch mit dem editierten Artikel wieder Befüllen zu können wurde ein „Onclick“-EventHandler bei der Initialisierung eingerichtet. Dieser wartet darauf, das eine der beiden Submit-Buttons Vorschau oder Speichern gedrückt werden und füllt daraufhin die Textarea mit dem editierten Dokument.

14 https://www.mediawiki.org/wiki/JQuery_Client
15 https://github.com/santhoshtr/jquery.i18n

```
$("form[wicket\\:id='formEditArticleResource']").submit(
    function(event) {
      $('#html-model').text(ve.dm.converter.getDomFromModel(
         target.getSurface().getModel().getDocument()).body.innerHTML
      );
   }
);
```

Listing 3: Hineinspeichern des editierten Dokuments in die Transitzone (ve.demo.init.js)

3.2 Entwicklung neuer Komponenten

Im nun folgenden möchte ich eine Beispielhafte Darstellung einiger Schritte bereitstellen mit deren Hilfe es möglich sein wird neue Knoten zu erzeugen und Dialoge bereitzustellen. Dabei werde ich eine Annotation eines Internen WOM Links erstellen und die nötigen Oberflächenelemente erzeugen, die dafür nötig sind die Eigenschaften eines WOM Links zu ändern. Ferner wird ein Inspektor entwickelt, der es dem Nutzer ermöglichen soll Bilder in das Dokument einzubringen.

WomIntLink

Für die Verarbeitung von externen Links gibt es schon eine brauchbare Unterstützung innerhalb des VisualEditors. Damit man auch interne Links einfacher in das Dokument einfügen kann und Techniken wie Auto-Completion nutzen kann werde ich im folgenden die Modellierung einer WOMIntLink Annotation beschreiben. Prinzipiell handelt es sich hierbei um eine Kopie der bereits implementieren LinkAnnotation, jedoch wurde sowohl Tag als auch Attribut angepasst um ein WomIntLink Syntaktisch besser vergleichen zu können.

Zuerst sollte eine Geeignete HTML Repräsentation gefunden werden. Hierfür habe ich mich für `<wom:intlink target="article">link</wom:intlink>` entschieden. Dieser Link würde dementsprechend auf den Artikel „article“ verweisen und den Titel „link tragen.

Um diesen zu erzeugen muss anschließend ein Visitor implementiert werden. Dieser wurde schon exemplarisch in Listing 4 dargestellt. Ebenso muss eine Rücktransformation angeboten werden. Eine solche könnte wie folgt aussehen:

```
private void parseWomIntLink (S2weDocument target, Wom3Node womnode,
Element element) {
   Wom3IntLink newNode = (Wom3IntLink)
target.createElementNS(Wom3Node.WOM_NS_URI, "intlink");
   newNode.setAttribute("target", element.attr("target"));
   Wom3Title titleNode = (Wom3Title)
target.createElementNS(Wom3Node.WOM_NS_URI, "title");
   newNode.setLinkTitle(titleNode);
   Wom3Text w3t = (Wom3Text) target.createElementNS(Wom3Node.WOM_NS_URI,
"text");
   titleNode.appendChild(w3t);
   w3t.setTextContent(StringEscapeUtils.escapeHtml4(element.text()));
}
```

Listing 6: ParseWomIntLink

Für das DataModel, ContenEditable und den Tools bedienen wir uns den vorhandenen LinkAnnotation Scripten. Hier wird lediglich darauf geachtet, das das matchingTags und die toDomElement Methoden angepasst werden. Ebenso müssen wir noch das Commando registrieren, damit der Inspector bei einem Klick auf das Tool geöffnet wird.

Das LinkAnnotation Widget wurde dahingehend geändert, dass ein TextLookupWidget genutzt wird. Dieses bietet die Auto-Completion Funktion an, da dafür ein TextInputWidget mit dem LookupElement erweitert wurde. Dieses erhält ein jQuery.getJson Objekt das Ergebnisse einer REST Schnittstelle abfragt. Die Stelle „1“ in Listing 7 zeigt diesen Aufruf. „2“ verweist hier auf die Validierung der Ergebnisse. Da zum Zeitpunkt der Entwicklung dieses Widgets keine API zur Verfügung stand musste so die Ergebnisliste gefiltert werden.

```
ve.ui.TextLookupWidget.prototype.getLookupRequest = function () {
    var deferred = $.Deferred();
    var request = null;

    this.isValid().done( function ( valid ) {
        if ( valid ) {

            // 1

            request = $.getJSON( "../../module/demoresults.json");
        } else {
            deferred.resolve( [] );
        }
    } );

    return request !== null ? request : deferred.promise( { abort:
function () {} } );
};

ve.ui.TextLookupWidget.prototype.getLookupMenuOptionsFromData = function (
data ) {
    var items = [],i;
    var value = this.getValue();
    $.each( data, function( key, val ) {
        if(key === "result") {
            for(i = 0; i < val.length; i++) {

                // 2

                if(String(val[i]).toLowerCase()
                  .substring(0, value.length) === value.toLowerCase() ) {
                    items.push(new OO.ui.MenuOptionWidget({
                        data: val[i],
                        label: val[i]
                    }));
                }

            }
        }
    });

    return items;
};
```

Listing 7: TextLookupWidget

3.1 Sweble Wiki Modul

3.1.1 WOM Serialisierung

Zu Beginn dieser Arbeit wurde der ArticleResourceToVeHtmlTransformer bereitgestellt. Dieser ist nach dem Visitor Pattern geschrieben, der es einfach machen soll, WOMs zu durchlaufen. Ein Beispiel für so eine visit Methode kann folgendermaßen aussehen:

```
public void visit(Wom3IntLink n) {
    idCounter++;
    this.WomIdMap.put(idCounter, n);
    String target = n.getAttribute("target");
    sb.append("<wom:intlink data-wom-id=\""+idCounter+"\"
target=\""+target+"\">");
    iterate(n);
    sb.append("</wom:intlink>");
}
```

Listing 4: Visit Methode des Wom3IntLink (ArticleResourceToVeHtmlTransformer.java)

Die WomIdMap wird dabei mit einer Referenz auf das Objekt befüllt mit dessen Hilfe es möglich sein soll den Knoten beim Parsen nach Round-Trip-Daten zu durchsuchen und diese dann auf den neuen Knoten anwenden zu können.

3.1.2 WOM Parsing

Der übergebene HTML-String wird beim Parsen von der Jsoup[16] Bibliothek in einen DOM überführt. Im Anschluss werden alle Kindelemente rekursiv besucht und daraus ein WOM erzeugt.

16 http://jsoup.org/

```
private void parseElement(S2weDocument target, Wom3Node womnode, Element
element) {
   String type = element.tagName();
   switch(type) {
      case …:
         parseProperElement(target, womnode, element, type);
         break;
   }
}

private void parseChilds(S2weDocument target, Wom3Node womnode, List<Node>
childs) {
   for(Node child: childs) {
      parse(target, womnode, child);
   }
}

private void parseProperElement(S2weDocument target, Wom3Node womnode,
Element element, String type) {
   Wom3Node newNode = target.createElementNS(Wom3Node.WOM_NS_URI, type);
   parseChilds(target, newNode, element.childNodes());
   appendAndRepair(target, womnode, newNode, element);
}
```

Listing 5: Pasing des HTML in WOM (VeHtmlToArticleResourceTransformer.java)

Appendix A

Glossar

Document Object Model (DOM): Abstrakter-Syntax-Baum eines HTML Dokuments
http://www.w3.org/DOM/

MediaWiki: Das Wiki System von Wikipedia
https://www.mediawiki.org/wiki/MediaWiki

Rich-Text-Editor: Siehe „Visueller Editor“

Sweble Parser: Ein Parser zur Konvertierung von Wikitext in einen WOM
http://sweble.org/2014/09/sweble-2-0-released/

Sweble Wiki: Ein Wiki System das auf Wiki Object Model aufgebaut ist
https://github.com/sweble/sweble

Transformer: Java Klasse zum umwandeln von Objekten in eine andere Repräsentationsform

Wikitext Markup: Auszeichnungsprache des MediaWiki Systems
https://www.mediawiki.org/wiki/Wikitext

Wiki Object Model (WOM): Abstrakter-Syntax-Baum eines Wikitext Markups
http://sweble.org/downloads/wom-tr.pdf

WYSIWYG: „What-You-See-Is-What-You-get“, „Was-Du-Siehst-Ist-Was-Du-Bekommst“, Eigenschaft von visuellen Editoren.

VisualEditor: Eine MediaWiki Erweiterung um Wiki Artikel visuell bearbeiten zu können
https://www.mediawiki.org/wiki/VisualEditor

Visueller Editor: Eine Komponente die es dem Nutzer erlaubt Dokumente nach WYSIWYG Art zu bearbeiten.

Literaturverzeichnis (APA Stil)

BuildWith. (2015). jQuery Usage Statistics. Retrieved September 28, 2015, from http://trends.builtwith.com/javascript/jQuery

Collins-Sussman, B., Fitzpatrick, B., & Pilato, M. (2004). *Version Control with Subversion*. Retrieved from https://books.google.com/books?hl=de&lr=&id=v1rN2MJ81JUC&pgis=1

Dohrn, H., & Riehle, D. (2011). WOM: An object model for Wikitext, (July), 43.

Haase, M. (2014). A visual editor for the wiki object model.

MediaWiki. (2014). Future/Real-time collaboration. Retrieved September 28, 2015, from https://www.mediawiki.org/wiki/Future/Real-time_collaboration

MediaWiki. (2015a). OOjs - MediaWiki. Retrieved September 27, 2015, from https://www.mediawiki.org/wiki/OOjs

MediaWiki. (2015b). VisualEditor/Design/Software overview. Retrieved September 29, 2015, from https://www.mediawiki.org/wiki/VisualEditor/Design/Software_overview

Quantcast. (2015). Quantcast - Top Ranking International Websites. Retrieved September 30, 2015, from https://www.quantcast.com/top-sites

Riehle, D., & Dohrn, H. (2013). Design and Implementation of the Sweble Wikitext Parser: Unlocking the Structured Data of Wikipedia.

Spinellis, D. (2005). Version Control Systems. *IEEE Software*, *22*(5), 108–109. doi:10.1109/MS.2005.140

Statista. (2015). Marktanteile der meistgenutzten Browserversionen weltweit im Juli 2015, 2015.

W3Techs. (2015). Usage Statistics and Market Share of JavaScript Libraries for Websites, September 2015. Retrieved September 28, 2015, from http://w3techs.com/technologies/overview/javascript_library/all

Wikipedia. (2015). Wikipedia:Statistik. Retrieved September 30, 2015, from https://de.wikipedia.org/wiki/Wikipedia:Statistik