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

MICHAEL HAASE
BACHELOR THESIS

A VISUAL EDITOR
FOR THE WIKI OBJECT MODEL

Eingereicht am 12. Mai 2014

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

Erlangen, 12. Mai 2014

License

Erlangen, 12. Mai 2014

Abstract

The Wiki Object Model (WOM) provides a model for the description of arbitrary textual data, for which there is no user-friendly possibility of editing in existence so far. The generic structure of the WOM allows the mapping of textual data, like wiki markup languages or even programming languages, onto the WOM in order to gain a homogeneous interface for access and modification of the underlying content. This bachelor thesis presents an easy to use visual editor for editing the Wiki Object Model and showcases a graphical user interface for the handling of arbitrary textual documents, based on current web technologies. This will provide an user-friendly tool for first-time users as well as one that illustrates to sophisticated users the advantages and possibilities that the new web technologies, like HTML5 and JavaScript, permit. In addition there is the perspective to create a foundation for further wiki-based research and development, based on the Wiki Object Model.

Zusammenfassung

Mit dem Wiki Object Model (WOM) steht ein Modell zur Beschreibung beliebiger textueller Daten zur Verfügung, für das bisher jedoch noch keine benutzerfreundliche Möglichkeit der Bearbeitung existiert. Die generische Struktur des WOM ermöglicht es, textuelle Daten, wie etwa Wiki-Auszeichnungssprachen oder Programmiersprachen, darin abzubilden und somit eine einheitliche Schnittstelle zum Zugriff und zur Bearbeitung der Inhalte zu erhalten. Diese Bachelorarbeit wird daher einen visuellen Editor zur Bearbeitung des Wiki Object Model vorstellen, der mithilfe aktueller Web-Technologien dem Benutzer die grafische Bearbeitung von beliebigen strukturierten Textdokumenten ermöglicht. Damit soll sowohl Einsteigern ein einfach zu verwendendes Tool an die Hand gegeben, als auch fortgeschrittenen Benutzern die Vorzüge und Möglichkeiten, die aktuelle Web-Technologien, wie HTML5 und JavaScript erlauben, aufgezeigt und eine mögliche Basis geschaffen werden, auf die weitere Forschungen und Entwicklungen im Bereich Wiki-Software und dem Wiki Object Model aufbauen können.

1 Einführung

Diese Bachelorarbeit beschreibt einen visuellen Editor zur Bearbeitung von strukturiertem Text auf Basis des Wiki Object Models.

Mit dem Wiki Object Model (WOM) steht ein Modell zur Beschreibung textueller Daten zur Verfügung, das aufgrund seiner generischen Struktur, die Bearbeitung beliebiger textueller Daten ermöglicht.

Dadurch kann ein einzelner Editor, der eine Manipulation der des WOM Dokumentes zugrunde liegenden Baumstruktur erlaubt, zur Bearbeitung beliebiger Dokumente eingesetzt werden.

Der hier vorgestellte Editor gestattet zunächst nur die Bearbeitung von Wikitext auf Basis seiner WOM Repräsentation, die durch den Sweble Parser[1] generiert wird. Wikitext bezeichnet die Auszeichnungssprache der MediaWiki Software und ist somit das Fundament der derzeit größten Online-Enzyklopädie, der Wikipedia.

Die Arbeit umfasst sowohl die Konzeption, bestehend aus der Auswahl der zu verwendenden Basistechnologien und der Entwicklung einer geeigneten Softwarearchitektur, als auch der Implementierung eines funktionsfähigen Prototyps, der sich der zuvor erarbeiteten Konzepte bedient und in bestehende Wiki-Systeme integrieren lässt.

Im letzten Teil werden die Stärken und Schwächen der Implementierung evaluiert und diskutiert, und Ansätze für mögliche zukünftige Erweiterungen vorgestellt.

[1] `http://sweble.org`

2 Forschung

2.1 Einführung

Als die Wikipedia 2001 das Licht der Welt erblickte, stellte dies einen Meilenstein in der Geschichte des Internets dar und hat den Einsatz von Wikis seither zu einer populären Möglichkeit des Wissensaustausches gemacht. Inhalte werden von freiwilligen, bisher vorwiegend männlichen und technikaffinen Autoren erstellt[1] und müssen dafür in der Auszeichnungssprache Wikitext kodiert werden.

Mit weltweit über 30 Millionen Artikeln[2] in 287 Sprachen[3] ist die Wikipedia die größte Enzyklopädie ihrer Art und befindet sich unter den weltweit zehn meistbesuchten Websites[4], dabei zugleich als einzige ohne kommerzielle Hintergrund.

Dennoch hat sich in den letzten Jahren nicht alles zum Positiven entwickelt. So hatte die Anzahl der aktiven Autoren ihren Höhepunkt im Jahr 2007 erreicht, rund 32000 Autoren, die jeweils mehr als fünf Änderungen pro Monat durchgeführt hatten. Seither nimmt die Anzahl der Autoren jedoch beständig ab.

Die durch die Wikimedia Foundation (2011) durchgeführte Editor Survey kam dabei unter anderem zu dem Ergebnis, dass 61 Prozent der Autoren die Komplexität, mit der Inhalte bearbeitet werden müssen, als zu hoch einschätzen, weshalb die Entwicklung von Tools, die die einfachere Bearbeitung von Artikeln ermöglichen sollen, zu einem der zentralen Aufgaben der nächsten Jahre erklärt wurde.

Auch Jimmy Wales erklärt in einem Interview, er habe die Hoffnung, dass durch einen einfach zu bedienenden, visuellen Editor neue „geeks who are not computer geeks“ zur Mitarbeit an der Wikipedia motiviert werden können und dadurch der bisher geringere Detailgrad im Bereich der nicht technischen Artikel weiter verbessert werden kann. (Simonite, 2013)

In einer von Vora et al. (2010) durchgeführten Studie zur Benutzbarkeit hatte die Mehrzahl der Probanden die Erwartung, für die Bearbeitung von Inhalten, einen

[1] http://blog.wikimedia.org/2012/04/27/nine-out-of-ten-wikipedians-continue-to-be-men/

[2] http://stats.wikimedia.org/EN/TablesArticlesTotal.htm

[3] http://meta.wikimedia.org/wiki/List_of_Wikipedias

[4] http://www.alexa.com/siteinfo/wikipedia.org

Editor, ähnlich gängiger Textverarbeitungs- oder Blogsoftware, vorzufinden und wurde von der Wiki Syntax zunächst abgeschreckt.

All dies zeigt die Notwendigkeit der ursprünglichen Beschreibung eines Wikis, als "the simplest online database that could possibly work"(Cunningham, 2002), im Hinblick auf Benutzerfreundlichkeit, wieder ein Stückchen näher zu kommen.

Neben der von der Wikipedia eingesetzten MediaWiki Software existiert noch eine Vielzahl weiterer Wiki-Softwares[5], die teils ihre eigenen Auszeichnungssprachen mitbringen. Dies macht es schwierig, für möglichst viele Produkte einsetzbare Werkzeuge zu entwickeln.

Diesem Umstand nimmt sich das Wiki Object Model (WOM) (Dohrn & Riehle, 2011) an, das eine Beschreibung beliebiger textueller Daten ermöglicht. Durch dessen generische Struktur können alle in den jeweiligen, von den unterschiedlichen Wiki-Softwares verwendeten Auszeichnungssprachen, vorhandenen Dokumente darin abgebildet werden und auf das WOM aufsetzende Werkzeuge müssen nur noch ein einzelnes Dokumentenformat unterstützen.

Für die WOM Repräsentation eines Dokumentes steht jedoch für menschliche Benutzer bisher keine einfache Möglichkeit der direkten Bearbeitung des Inhalts zur Verfügung, was mit dem im Rahmen dieser Bachelorarbeit entwickelten Editors geändert werden soll.

Des Weiteren soll dieser Editor eine mögliche Basis für zukünftige, auf dem Wiki Object Model aufbauende, Entwicklungen und Forschungsarbeiten im Wiki-Umfeld bieten.

Dazu zählen etwa Transformationen und Refactorings der Struktur eines Wikis (Dohrn & Riehle, 2013). Da die einzelnen Inhalte in aktuell eingesetzter Wiki-Software zumeist auf unstrukturiertem Text basieren, gestaltet sich die Umstrukturierung eines Wikis als sehr aufwändig. Zur Umbenennung einer Kategorie in der MediaWiki Software, muss beispielsweise in allen, der Kategorie angehörenden, Seiten, der Name der Kategorie angepasst werden[6]. Um diese und ähnliche Operationen zu automatisieren schlagen die Autoren die Verwendung von XSL Transformationen (Kay, 2007) auf der XML Repräsentation des WOM vor.

[5] `http://www.wikimatrix.org/`

[6] `http://commons.wikimedia.org/wiki/Commons:Rename_a_category/de`

2.2 Related Work

Schon seit dem Aufkommen grafischer Benutzeroberflächen wird die Frage diskutiert, ob für die Erstellung von Dokumenten die neuen visuellen Möglichkeiten Anwendung finden sollen oder eine logische Auszeichnung, wie etwa in LaTeX, nicht sinnvoller ist. (Lamport, 1988)

Im Gegensatz zu Textverarbeitungsprogrammen wie Microsoft Word[7], in denen Auszeichnungen, wie das Hervorheben einer Überschrift, direkt für den Benutzer sichtbar werden, muss in LaTeXder als Überschrift dienende Text speziell als solcher gekennzeichnet werden, etwa durch `\section{Kapitelüberschrift}`.

Auch Sauer (2006) sieht den allgegenwärtigen Einsatz von visuellen Editoren kritisch, merkt aber an, dass dies für Anwender, die nur den Umgang mit Standard-Textverarbeitungsprogrammen gewohnt sind, noch immer das Mittel der Wahl ist.

Mit der Entwicklung des Internets wurden auch die Browser auf der Seite der Anwender um immer mehr Funktionen erweitert, während gleichzeitig die zur Verfügung stehenden Hardwarekapazitäten stetig angestiegen sind. Dies hatte unter anderem zur Folge, dass zahlreiche neue Möglichkeiten der Interaktion zwischen Webseite und Benutzer ermöglicht wurden, die vorher zum Beispiel aufgrund mangelnder Ressourcen überhaupt nicht denkbar gewesen waren.

Einer der zentralen Aspekte im „Web 2.0“ stellt die stärkere Partizipation der Benutzer dar. Um diese zu ermöglichen wurden die Hürden dafür fortschreitend abgebaut, um auch Benutzer ohne tiefer gehenderes technisches Verständnis zur aktiven Teilnahme anleiten zu können.

Dabei können dem Benutzer zum einen Hilfestellungen angeboten werden und zum anderen kann die eigentliche Auszeichnungssprache hinter einem komplexeren Frontend verborgen werden, sodass der Benutzer damit kaum noch in Berührung kommt.

Um dies zu verdeutlichen, werde ich nachfolgend einen kurzen Überblick über die Entwicklungen der Techniken zum Erstellen oder der Bearbeitung von textuellen Inhalten im Browser der letzten Jahre geben und aktuelle Trends aufzeigen. Im Anschluss daran werde ich das Wiki Object Model vorstellen und die zur Bearbeitung relevanten Teile erläutern.

2.2.1 Entwicklung

HTML Textarea Der im November 1995 veröffentlichte HTML 2.0 Standard (Berners-Lee & Connolly, 1995) enthielt erstmals Formularelemente.

JavaScript Auch im Jahre 1995 wurde die erste Version von JavaScript im Netscape Browser ausgeliefert und zwei Jahre später mit ECMAScript (ECMA International, 1999) standardisiert. Damit war es erstmals möglich, Skripte auf

[7] `http://office.microsoft.com/de-de/word/`

der Seite des Benutzers, und zwar direkt im Browser, auszuführen, um beispielsweise Benutzereingaben zu validieren, dem Benutzer dadurch eine direkte Rückmeldung über etwaige Fehleingaben aufzuzeigen und damit Zeit und Kosten zu ersparen, die durch ein Absenden und erneutes Laden der Webseite entstanden wären.

Formatierungshilfen Neben der Validierung von Benutzereingaben können dem Benutzer auch Hilfen bei der Eingabe von Formatierungen in der verwendeten Auszeichnungssprache gegeben werden.

Dem Benutzer werden beispielsweise Schaltflächen in einer Werkzeugleiste oberhalb des HTML Eingabebereiches zur Verfügung gestellt, die die erforderliche Syntax zum Einfügen der Formatierung in das Textfeld kopiert. Eine weitere, insbesondere in Internetforen wie dem phpBB3[8] Verwendung findende Auszeichnungssprache ist der in Abbildung 2.1 dargestellte BBCode, der eine im Vergleich zu HTML vereinfachte Möglichkeit zur Auszeichnung von Text bietet.

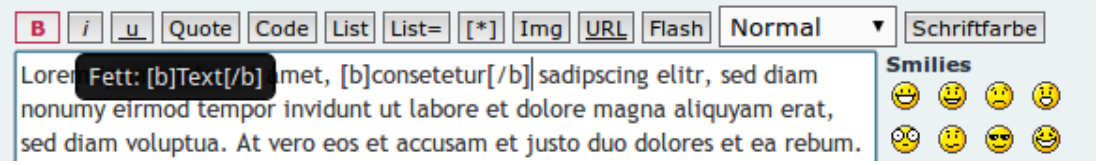

Abbildung 2.1: Formatierungshilfen am Beispiel von BBCode im phpBB3

Inline-Editing Bisher musste der Anwender sich zunächst in der Struktur des in der Auszeichnungssprache vorliegenden Dokumentes zurechtfinden, um die Änderungen an der richtigen Stelle durchführen zu können. Dies stellt mit zunehmender Länge allerdings ein immer zeitraubenderes Unterfangen dar.

Die Idee des Inline-Editierens nimmt sich diesem Umstand an, indem zunächst das gesamte Dokument, bereits vollständig formatiert, im Browser dargestellt wird. Möchte der Anwender einen Teil des Dokumentes bearbeiten, so muss dieser nur auf den entsprechenden Bereich klicken (oder auf ein in näherer Umgebung befindliches Bearbeitungs-Icon). Mittels JavaScript wird ein, ähnlich dem in Abbildung 2.2 sichtbaren, Formularelement mit dem Inhalt des zu bearbeitenden Elementes an eben jener Position erzeugt, an der vorher das ausgewählte Element platziert war.

Daraufhin kann der Anwender die Änderungen vornehmen und anschließend abspeichern.

In Posma (2011) wird die Verwendbarkeit des Inline-Editierens zur Bearbeitung von Wiki-Inhalten untersucht. Dabei hat sich gezeigt, dass dies zwar eine Verbesserung zum Bearbeiten von reinen Wikitext Dokumenten ist, aber gerade erstmalige Benutzer noch Schwierigkeiten haben.

[8] `https://www.phpbb.com`
[9] `http://jquery-in-place-editor.googlecode.com/svn/trunk/demo/index.html`

Abbildung 2.2: Inline-Editing am Beispiel eines jQuery Plugins[9]

Rich Text Editor Rich Text Editoren, wie der in Abbildung 2.3 dargestellte CKEditor, versuchen die gesamte Auszeichnungssprache hinter einer für den Anwender möglichst einfach zu bedienenden Benutzeroberfläche zu verbergen, wie dieser es von gängigen Textverarbeitungsprogrammen, wie Microsoft Word oder LibreOffice[10] gewohnt ist.

Dies bedarf jedoch eines ungleich höheren Implementierungsaufwandes, da dazu der Großteil der Funktionen in JavaScript nachgebildet werden muss, idealerweise in einer für den Anwender konsistenten und den Gewohnheiten entsprechenden Art und Weise. So wird der Benutzer unter anderem erwarten, dass beliebige markierte Textstellen in die Zwischenablage kopiert werden können.

Zur Unterstützung bei der Umsetzung derartiger Anwendungsfälle, stellt die aktuelle Version fünf des HTML Standards (Berjon, Robin et al., 2014), und damit einhergehend die Browserhersteller[11], den Entwicklern inzwischen einfacher zu verwendende Funktionen, wie *contentEditable*, zur Verfügung.

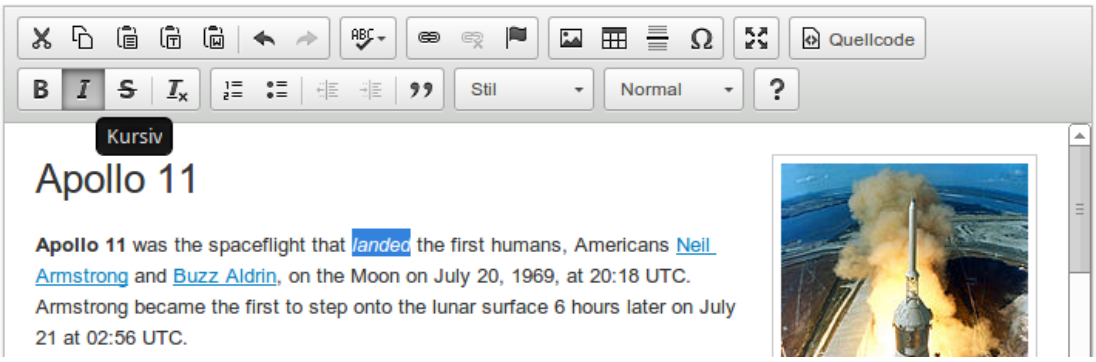

Abbildung 2.3: Rich Text Editor am Beispiel von CKEditor[12]

[10] http://de.libreoffice.org/
[11] https://developer.mozilla.org/de/docs/Rich-Text_Editing_in_Mozilla
[12] http://ckeditor.com/

2.2.2 Aktuelle Trends

Kollaboratives Bearbeiten Kollaboratives Bearbeiten von Texten in Echtzeit bedeutet, dass mehrere Benutzer, von mehreren Endgeräten aus, dasselbe Dokument bearbeiten können und nicht nur die eigenen, sondern auch direkt die Änderungen der jeweils anderen Benutzer, auf dem eigenen Bildschirm sehen können.

Den ersten kommerziell erfolgreichen Versuch stellte *Writely* von Sam Schillace dar, das 2006 von Google gekauft wurde und seither in Form von *Google Docs* bzw. *Google Drive* Bekanntheit erlangt hat[13].

Im Open Source Bereich zeigt sich das unter der Apache License 2.0 veröffentlichte Projekt *Etherpad Lite*[14] als verbreitete Alternative.

Cloud Durch die scheinbare Allgegenwärtigkeit des Internets und der Vielzahl an Endgeräten auf Seiten des Benutzers, die damit verbunden sind, wächst auch der Wunsch, von überall und mit jedem dieser Geräte auf die eigenen Daten zugreifen zu können.

Die Cloud stellt für den Benutzer den Zugriff auf seine Daten und verwendenden Dienste über das Internet bereit. Dazu kann eine Vielzahl von Servern über eine einzelne Schnittstelle im Internet erreichbar sein, bei der die durch die Server bereitgestellte Rechenleistung, durch Zuschalten neuer Instanzen, an die aktuell zu bedienende Lastsituation angepasst werden kann. (Mell & Grance, 2011) Für den Benutzer ist nur die zur Verfügung gestellte Schnittstelle sichtbar, während die dahinter liegende Infrastruktur in einer „Wolke“ verborgen bleibt.

Hier spielt vor allem die große Diversität an zu unterstützenden Plattformen im Hinblick auf unterschiedliche Bildschirmgrößen und Eingabemöglichkeiten eine wichtige Rolle, von Smartphones und Tablets mit Toucheingabe, bis hin zu Laptops und Desktop Computern, und den klassischen Eingabegeräten, Maus und Tastatur.

2.2.3 Wiki Object Model

Mit dem Document Object Model (DOM) (Le Hors, Arnaud et al., 2014) steht eine standardisierte Schnittstelle zum Zugriff und Manipulation von HTML Dokumenten bereit. Die einzelnen HTML Elemente werden im DOM als Knoten repräsentiert, die jeweils einen Eltern-, sowie einen oder mehrere Kindknoten haben können. Die dadurch entstehende Struktur wird als DOM-Baum bezeichnet.

Abbildung 2.4 zeigt ein beispielhaftes HTML5 Dokument auf der linken und den daraus resultierenden DOM-Baum auf der rechten Seite.

[13] `http://readwrite.com/2006/10/10/google_docs_spreadsheets`
[14] `http://etherpad.org/`

```html
<!DOCTYPE html>
<html>
<head>
    <title>Titel</title>
</head>
<body>
    <p>Paragraf</p>
</body>
</html>
```

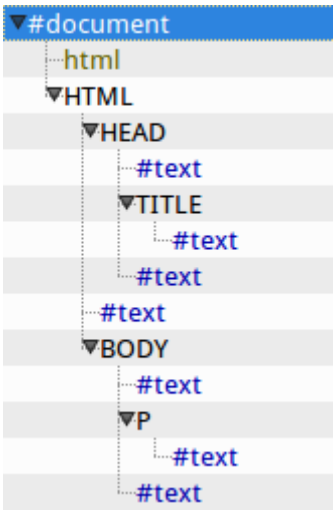

Abbildung 2.4: HTML Dokument und zugehöriger DOM-Baum

Das Wiki Object Model (WOM) (Dohrn & Riehle, 2011) erweitert dieses Konzept auf beliebigen strukturierten Text und ermöglicht so die Interaktion und Bearbeitung von strukturierten Textdokumenten aller Art.

Listing 2.1 zeigt ein einfaches Wikitext Dokument.

```
=Titel=
Paragraf
```

Listing 2.1: Wikitext Dokument

Dazu in Listing 2.2 ein Ausschnitt des zugehörigen WOM-Baumes, der hier in serialisierter Form im JSON-Format vorliegt.

```json
{
  "!type": "article",
  "@xmlns": "http://sweble.org/schema/wom30",
  "@version": "3.0",
  "@title": "Test.wikitext",
  "@xmlns:mww": "http://sweble.org/schema/mww30",
  "!children": [{
      "!type": "body",
      "!children": [{
          "!type": "section",
          "@level": "1",
          "!children": [{
              "!type": "heading",
              "!children": [{
                  "!type": "rtd",
                  "!children": [{
                      "!type": "#text",
```

```
18                          "!value": "="
19                      }]
20                  }, {
21                      "!type": "text",
22                      "!children": [{
23                          "!type": "#text",
24                          "!value": "Titel"
25                      }]
26                  }, {
27                      "!type": "rtd",
28                      "!children": [{
29                          "!type": "#text",
30                          "!value": "="
31                      }]
32                  }
33              ]
34          },
```

Listing 2.2: Ausschnitt eines serialisierten WOM-Baumes

Zur Wiederherstellung des ursprünglichen Dokumentes aus dem WOM-Dokument, beinhaltet das WOM neben dem in Textknoten (Zeile 21-25) gespeicherten textuellen Inhalt auch *Round-Trip Data* (RTD) Knoten (Zeile 15-19 und 27-31). Diese enthalten die zur Auszeichnung im ursprünglichen Dokument verwendeten syntaktischen Elemente, wie hier etwa `=` zur Deklarierung einer Überschrift. Die *RTD* Elemente sind optional, aber für die exakte Rücktransformation und den Erhalt von Formatierungen essenziell.

Da die Bearbeitung von WOM-Dokumenten im Browser und unter Zuhilfenahme der dort verfügbaren DOM-Implementierung erfolgen soll, werde ich in dieser Bachelorarbeit eine Möglichkeit zur verlustfreien Übersetzung von WOM- in HTML- und anschließend zurück in ein WOM-Dokument vorstellen.

2.3 Fragestellung

Nach der Vorstellung der alternativen Ansätze zur Bearbeitung textueller Daten lässt sich daraus nun die folgende Fragestellung ableiten:

> Welches Konzept zur Implementierung eines im Webbrowser ausgeführten Editors, zur Bearbeitung von strukturiertem Text im Wiki Object Model, führt mit dem aktuellen Stand der Technik zur bestmöglichen Benutzerfreundlichkeit?

2.4 Vorgehen

Im nun folgenden Kapitel werde ich mein Vorgehen zur Bearbeitung der im Rahmen dieser Bachelorarbeit gestellten Fragestellung aufzeigen. Dabei werde ich zunächst auf die allgemeinen Ziele eingehen und anschließend im Kapitel 2.4.2 die mit dem Editor in Interaktion tretenden Teilhaber und ihre Motive aufzeigen. Eine für die Entwicklung und den Betrieb gleichermaßen bedeutende Eigenschaft des Softwaresystems ist dessen Architektur, welche ich im Kapitel 2.4.4 vorstellen werde. Abschließend werde ich im Kapitel 2.4.5 auf die Frage der zu verwendenden Basistechnologien eingehen und deren jeweilige Vor- und Nachteile diskutieren, um so ein solides Fundament für die nachfolgende Implementierung bereit zu stellen.

2.4.1 Qualitätsziele

Bei der Konzeption des Editors gilt es verschiedene, aus der Fragestellung hervorgehende Qualitätsziele zu beachten, die eine erfolgreiche Umsetzung der gesteckten Ziele und eine Evaluierung des Resultats erst ermöglichen. Die zentralen Ziele werden nachfolgend aufgeführt, geordnet nach absteigender Relevanz und dabei kurz erläutert.

Lokalität von Änderungen Ein entscheidendes Kriterium für die Akzeptanz eines neuen Editors und die erfolgreiche Integration in bestehende Umgebungen ist das Lokalitätsprinzip. Das bedeutet, dass die Bearbeitung eines Elementes bereits existierender Inhalte im Idealfall auch nur in der Änderung eben jenes Elementes resultiert, das davon direkt betroffen ist.

Listing 2.3 zeigt eine mögliche Wikitext Notation zur Einbindung einer verkleinerten Version des Bildes *Image.jpg*. Der darin eingebettete Kommentar sollte auch nach Änderung einer beliebigen anderen Stelle im Dokument noch erhalten bleiben. Dies gilt nicht nur für Kommentare, sondern für beliebige Formatierungen, die ein potenziell anderer Benutzer dort zuvor eingetragen hat.

```
1 [[ File:Image.jpg | thumb <!-- Comment -->]]
```

Listing 2.3: Wikitext Bild mit Kommentar

Dies ist insbesondere dort von Bedeutung, wo der Anwender Zugriff auf die Versionshistorie eines Dokumentes hat, da nur so die Intention einer früheren Bearbeitung, die möglicherweise durch einen anderen Anwender durchgeführt wurde, klar ersichtlich werden kann. Werden dagegen durch Modifikationen, wie etwa der Bearbeitung eines Paragrafen, sämtliche Formatierungen im gesamten Dokument entfernt, so werden diese Unterschiede auch beim Vergleich der beiden Versionen sichtbar.

Daneben muss bei einem System zur Bearbeitung von Inhalten auch die Möglichkeit erhalten bleiben, diese weiterhin in der zugrunde liegenden Auszeichnungssprache durchzuführen, sei es zur Bearbeitung noch nicht im Editor implementierter

Elemente der Syntax oder einer der zahlreichen Erweiterungen, oder der Akzeptanzsteigerung bei Autoren, die weiterhin auf die Auszeichnungssprache setzen wollen.

Bedienbarkeit Eine einerseits einfache, aber auch komfortable Bedienung des Editors muss gewährleistet sein. Erfahrene Anwender, die seit jeher nur auf blankem Wikitext arbeiten, sollen durch den visuellen Editor nicht zu stark in den Möglichkeiten, die ein direktes Bearbeiten von Wikitext zweifelsfrei bietet, eingeschränkt werden. Aber auch gelegentliche und allem voran Erstbenutzer, sollen durch das Bedienkonzept optimal unterstützt und nicht direkt abgeschreckt werden.

Dabei sollte der Editor insbesondere eine intuitive Benutzeroberfläche bieten, die im Normalfall keine Konsultation eines Benutzerhandbuches erfordert, um die Einstiegshürde möglichst gering zu halten. Kurze und prägnante Tooltips, die den Anwender während der Bearbeitung unterstützen, könnten sich dagegen als hilfreich erweisen.

Wartbarkeit Fehlerbehebung und Erweiterbarkeit müssen durch Dokumentation und Programmstruktur des Source Codes des Editors durchgängig gewährleistet sein.

Da ein komplexeres Softwaresystem in der Regel auch Programmierfehler enthält, die auch durch andere Entwickler behoben werden müssen, bedarf es einer ausreichenden Dokumentation, sowohl der Architektur, der Implementierung, als auch der zugrunde liegenden Konzepte, damit eine schnelle Einarbeitung und Fehlerkorrektur möglich ist.

Gleiches gilt für die Erweiterbarkeit des Editors, wie etwa zur Unterstützung von WOM Erweiterungen oder der Benutzeroberfläche. Eine gute Softwarearchitektur sollte dies unterstützen und nicht verhindern.

Portierbarkeit Der im Rahmen dieser Bachelorarbeit konzipierte Editor ist aufgrund der Verwendung des Sweble Parsers und dessen zur Zeit alleiniger Unterstützung von Wikitext auf eine Integration in die MediaWiki- oder andere, diese Auszeichnungssprache unterstützende Software limitiert. Dennoch sollte die Architektur die Möglichkeit für später erfolgende Erweiterungen des Wiki Object Model zur Unterstützung anderer Auszeichnungssprachen bieten.

2.4.2 Stakeholder

Neben den Qualitätszielen ist auch die Berücksichtigung der Interessen und Absichten der verschiedenen Benutzergruppen für die Softwarearchitektur von Bedeutung. Dafür werde ich die unterschiedlichen Erwartungen an ein solches System aufzeigen:

Erstbenutzer hat einen vermeintlichen Fehler in einem vorhandenen Artikel entdeckt und möchte diesen korrigieren oder gar neue Informationen beisteuern. Der Benutzer sollte bestmöglich angeleitet werden und bei Bedarf auch auf ei-

ne umfangreiche Hilfefunktion zurückgreifen können. Kenntnisse zur Bearbeitung von Texten sind einzig aus Textverarbeitungsprogrammen wie LibreOffice oder ähnlichen bekannt, und er erwartet deshalb eine ähnlich komfortable Benutzung.

Wikipedianer bearbeitet und erweitert Artikel schon immer ohne direkte grafische Repräsentation direkt als Wikitext. Erwartet die Möglichkeit zur Bearbeitung und Zugriff auf jegliche in Wikitext integrierte Funktion.

Wiki-Administrator Neben den Interessen der Wikipedianer soll auch die Unterstützung komplexer, strukturverändernder Aktionen innerhalb eines Wikis, wie zum Beispiel der Umbenennung von Kategorien oder Templatevariablen, durchführbar sein, da diese bisher größtenteils von Hand oder unter der Zuhilfenahme von programmierbaren Bots durchgeführt werden müssen.

Entwickler Zur Fehlerbehebung und Erweiterung muss ausreichende und angemessene Dokumentation vorhanden sein, um sich relativ zeitnah in das Gesamtsystem einarbeiten zu können. Der Aufbau der Softwarearchitektur sowie Codestruktur sollten gängigen Prinzipien folgen.

2.4.3 Randbedingungen

Bei der Umsetzung des Gesamtsystems gilt es verschiedene, vorgegebene Randbedingungen einzuhalten. Dieser Abschnitt wird einen Überblick darüber geben.

Organisatorische Randbedingungen

Zeitplan Die Implementierung des Systems muss innerhalb der für die Bachelorarbeit angesetzten 360 Stunden abgeschlossen sein.

Veröffentlichung als Open Source Die Quelltexte werden unter entsprechender Open Source Lizenz nach Abschluss der Bachelorarbeit veröffentlicht.

Technische Randbedingungen

Implementierung in Java oder JavaScript Clientseitig soll JavaScript eingesetzt werden. Der Sweble Parser selbst ist in Java implementiert. Daher bietet sich auch Java auf Serverseite an, sollte sich das Sytem nicht rein clientseitig realisieren lassen. Allerdings steht mit serverseitigem JavaScript seit einiger Zeit eine Alternative zu gängigen Serverimplementierungen zur Verfügung[15].

[15] `http://nodejs.org/`

Fremdsoftware Lizenzen Der Sweble Parser ist unter der Apache License Version 2.0 veröffentlicht worden. Bei Verwendung von Fremdsoftware muss diese zwingend unter einer dazu kompatiblen Open Source Lizenz stehen (MIT, BSD)[16].

Browserkompatibilität Der Editor sollte auf allen modernen Webbrowsern funktionsfähig sein. Inwieweit die Verwendung von mobilen Browsern auf Smartphones oder Tablets sinnvoll erscheint, bedarf einer gesonderten Evaluierung.

2.4.4 Softwarearchitektur

Nachdem die Qualitätsziele, die Erwartungen der Stakeholder an das System und die einzuhaltenden Randbedingungen bekannt sind, kann mit der eigentlichen Konzipierung der Softwarearchitektur begonnen werden.

Dazu werde ich das System als ganzes zunächst in Bezug zu den es umgebenden Systemen setzen und die zur Kommunikation erforderlichen Schnittstellen definieren. Das Softwaresystem wiederum lässt sich in einzelne Komponenten unterteilen. Diese werden daraufhin erst einmal grob skizziert und anschließend weiter verfeinert.

[16] `http://apache.org/legal/resolved.html#category-a`

Kontextabgrenzung

Dieser Abschnitt zeigt die bei der Verwendung des Editors involvierten Akteure und Systeme auf und veranschaulicht deren Zusammenhänge.

Fachlicher Kontext Der WOM3 Editor steht gemäß Abbildung 2.5 in Interaktion mit dem Benutzer und der als Datenbasis fungierenden Wiki-Software.

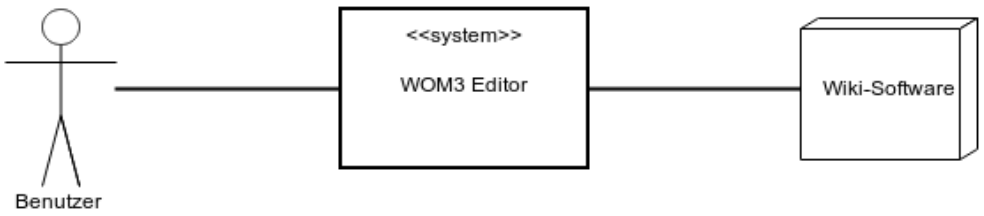

Abbildung 2.5: Fachlicher Kontext

Benutzer Ein Benutzer gemäß der unter Abschnitt 2.4.2 aufgeführten Akteure. Die Interaktion mit dem Editor erfolgt durch Maus- und Tastatureingaben über eine grafische Benutzeroberfläche innerhalb eines Webbrowsers auf einem Endgerät des Benutzers.

Wiki-Software Die verwendete Wiki-Software dient mit ihren Inhalten als Datenbasis für alle Operationen innerhalb des Editors. Während die Datenquelle zu einem späteren Zeitpunkt auf beliebige im Wiki Object Model vorliegende Dokumente erweitert werden kann, beschränke ich mich im Rahmen dieser Bachelorarbeit auf die Verwendung der MediaWiki-Software.

Technischer Kontext Abbilung 2.6 zeigt die mit dem WOM3 Editor in Verbindung stehenden Komponenten auf.

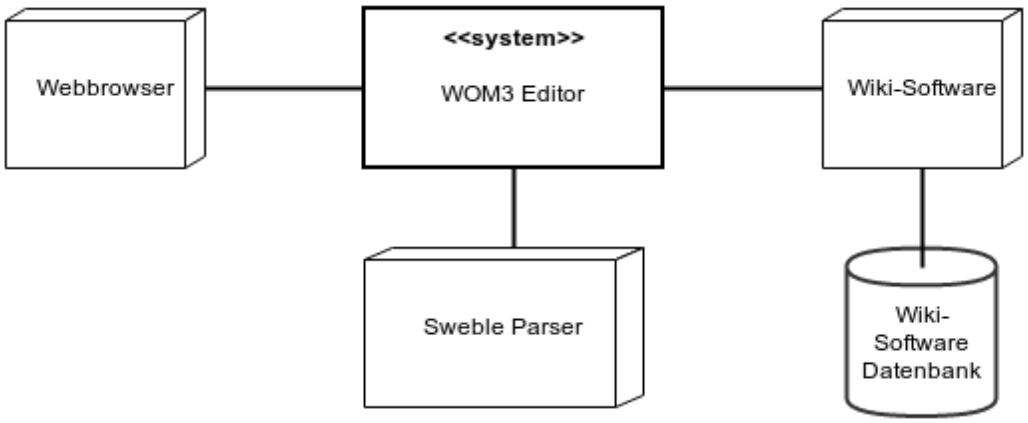

Abbildung 2.6: Technischer Kontext

Webbrowser Der Zugriff auf das System erfolgt mittels eines aktuellen Webbrowsers. Grundvoraussetzungen für einen fehlerfreien Betrieb sind sowohl HTML5 Unterstützung, als auch aktiviertes JavaScript.

Wiki-Software Die Wiki-Software enthält die zu bearbeitenden Daten und stellt diese über eine Schnittstelle der Außenwelt zur Verfügung. Alternativ könnte der Zugriff auch direkt über die darunter liegende Datenbank erfolgen, allerdings sollte dies, sofern möglich, vermieden werden, da sich Schnittstellenspezifikationen im Allgemeinen. nicht so schnell ändern und damit, unter Umständen, nötige Anpassungen, etwa bei Aktualisierungen der Wiki-Software, minimiert werden können.

Sweble Parser Der Sweble Parser konvertiert die zu bearbeitenden, als Wikitext vorliegenden Daten in das Wiki Object Model Version 3 (WOM3), auf Basis dessen die weitere Bearbeitung der Inhalte erfolgt. Im Anschluss an die Bearbeitung ist der Parser auch für die Rücktransformation in Wikitext zuständig.

Bausteinsicht

Der Editor lässt sich damit in die folgenden zwei Komponenten unterteilen:

Benutzerschnittstelle Die Benutzerschnittstelle bildet die eigentliche Interaktionskomponente für den Anwender. Alle Eingaben werden hier entgegen genommen und die zu bearbeitenden Daten dadurch modifiziert.

Eingabebehandlung Die Maus- und Tastatureingaben, die durch den Anwender initiiert werden, müssen entgegen genommen und verarbeitet werden.

Renderer Das zu bearbeitende Dokument muss dem Anwender formatiert dargestellt werden.

Werkzeugleiste Über die Werkzeugleiste hat der Anwender die Möglichkeit, die verschiedenen Formatierungen vorzunehmen und alle den Editor betreffenden Funktionen aufzurufen.

WOM3Tools Zur Kapselung der für die Handhabung der im WOM3 Format vorliegenden Daten benötigten Funktionen bietet es sich an, diese in eine eigene Komponente auszulagern. Die Daten, die zur Bearbeitung im Editor vom REST-API der Wiki-Software abgerufen wurden und mithilfe des Sweble Parsers ins Wiki Object Model überführt wurden, liegen zunächst serialisiert im JSON-Format vor.

Ein REST-API ist eine Schnittstelle, die über das HTTP Protokoll einen Webservice zur Verfügung stellt, der über eine eindeutige URL erreichbar ist und auf dem verschiedene Methoden zum Abfragen oder Speichern von Daten definiert sein können.

Die Aufgabe der WOM3Tools Komponente besteht darin, die Daten in ein internes Format und im Anschluss in eine, für die Benutzerschnittstelle verständliche Repräsentation zu überführen. Nach der Bearbeitung muss dieser Vorgang in umgekehrter Richtung wiederholt werden.

WOM3Parser Zuständig für die Konvertierung des im JSON Format vorliegenden WOM3 Baumes in die interne Repräsentation.

WOM3Serializer Zuständig für die Konvertierung des in der internen Repräsentation vorliegenden WOM3 Baumes in das JSON Format.

WOM3ToHTMLConverter Zuständig für die Konvertierung der internen Repräsentation des WOM3 Baumes in eine HTML Repräsentation.

HTMLToWOM3Converter Zuständig für die Konvertierung des durch HTML Elemente repräsentierten WOM3 Baumes in die interne Darstellung.

Laufzeitsicht

Nachdem die einzelnen Komponenten bekannt sind, werde ich nun die Kommunikation der einzelnen Teile untereinander für einen typischen Ablauf einer Bearbeitung aufzeigen.

1. Der Benutzer ruft mit seinem Webbrowser eine Webseite auf, die den Editor wahlweise in eine bestehende Anwendung integriert oder als eigenständige Webapplikation ausführen soll. Der in der HTML Seite verlinkte JavaScript Code, sowie weitere benötigte Ressourcen, wie Bilder und CSS-Dateien, werden von einem Webserver im Internet durch den Webbrowser nachgeladen und anschließend lokal zur Ausführung gebracht.

2. Sobald der initiale Ladevorgang abgeschlossen ist, wird der zu bearbeitende Artikel an der REST-API angefragt. Daraufhin wird der aktuelle Inhalt des Artikels über die MediaWiki API heruntergeladen und lokal zwischengespeichert. Der Sweble Parser überführt den Wikitext nun in seine WOM3 JSON Darstellung, welche der WOM3Parser einliest, um sie in die in JavaScript geschriebene WOM3 Implementierung abzubilden, woraus der WOM3ToHTMLConverter im nächsten Schritt ein HTML Dokument erstellt, das abschließend verwendet werden kann, um die Anfrage des Clients zu beantworten.

   Zur Veranschaulichung zeigt Abbildung 2.7 ein UML Sequenzdiagramm für diesen Ablauf und die Interaktion der einzelnen Komponenten untereinander auf.

3. Nach erfolgter Bearbeitung wird das daraus hervorgegangene HTML Dokument in eine HTTP-Post Anfrage eingebettet und zurück an die REST-API geschickt. Dort versucht der HTMLToWOM3Converter aus dem durch das HTML Dokument entstandenen DOM-Baum einen neuen WOM3 Baum zu erzeugen. Im letzten Schritt serialisiert der WOM3Serializer die interne WOM3 Darstellung zurück in seine JSON Repräsentation.

Verteilungssicht

Abschließend zeigt Abbildung 2.8 die Verteilung der beteiligten Komponenten auf.

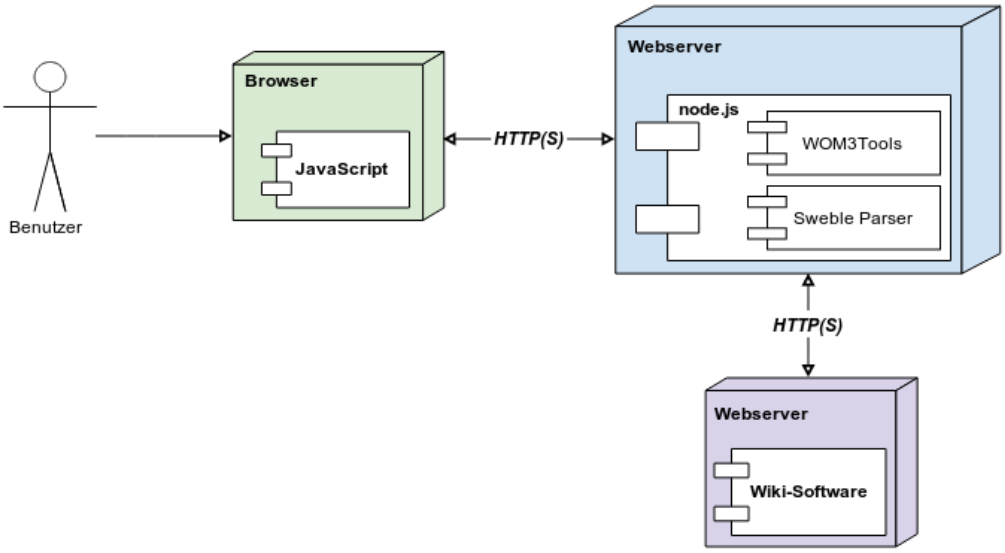

Abbildung 2.8: Verteilungssicht

Der Browser des Benutzers wird lokal auf dessen Computer ausgeführt, der wiederum die Ausführungsumgebung für die JavaScript Dateien der Benutzerschnittstelle darstellt.

Die WOM3Tools, sowie eine ausführbare JAR Datei des Sweble Parsers befinden sich in der aktuellen Version auf Serverseite, auf der eine *node.js* Anwendung als Webserver fungiert und sich um die Beantwortung von API-Anfragen kümmert, indem dieser auf die WOM3Tools Komponenten zurückgreift.

Daneben existiert noch eine MediaWiki Instanz als Datenquelle, deren API für den *node.js* Server erreichbar sein muss, um den zu bearbeitenden Artikel als Wikitext abrufen zu können.

2.4.5 Basistechnologien

JavaScript Framework

Zur Fragestellung Mit *ECMAScript* gibt es einen anerkannten Standard für JavaScript. Nichtsdestotrotz unterscheiden sich die JavaScript Implementierungen der verschiedenen Browserhersteller in einigen Details, wie der Eventbehandlung oder das unter anderem für *AJAX* verwendete *XMLHttpRequest*. (van Kesteren, Aubourg, Song & Steen, 2014)

Mit dem *XMLHttpRequest* steht eine Schnittstelle zur Datenübertragung zwischen Client und Server zur Verfügung. Diese wurde ursprünglich von Microsoft entwickelt und fand daher zuerst Verwendung im Internet Explorer. Die anderen Browserhersteller übernahmen dieses Konzept jedoch bald, allerdings nicht mit einer dazu kompatiblen Schnittstelle, weshalb bei Verwendung stets, je nach ausführender Browserumgebung, der passende Aufruf ausgewählt werden muss.

Zur einfacheren und weniger fehleranfälligen Implementierung bietet sich deshalb der Einsatz eines JavaScript Frameworks an, das diese und ähnliche Differenzen hinter einer konsistenten Schnittstelle verbirgt und nebenbei auch hilfreiche Zusatzfunktionen beinhaltet. Da mittlerweile zahlreiche Frameworks zur Verfügung stehen, ist zu entscheiden, auf Basis welchen Frameworks die Implementierung aufbauen soll.

Entscheidung Als JavaScript Framework wird die Google Closure Library eingesetzt.

Betrachtete Alternativen

jQuery Die 2006 veröffentlichte jQuery Bibliothek ist das mittlerweile am weitesten verbreitete JavaScript Framework[17]. , veröffentlicht unter der MIT-Lizenz[18]. Neben der Core Bibliothek[19] gibt es mit jQuery UI[20] und jQuery Mobile[21] zwei Bibliotheken zur Erstellung von Benutzerschnittstellen, sowohl für Desktop-Computer als auch mobile Geräte. Außerdem existiert mittlerweile eine Vielzahl verschiedener Plugins[22].

Google Closure Library Die Closure Library wurde 2009[23] von Google unter der Apache License 2.0 veröffentlicht[24].

Ich habe mich für den Einsatz dieses Frameworks entschieden, da es insbesondere mit der Verwendung des Closure Compilers ein Alleinstellungsmerkmal

[17] `http://w3techs.com/technologies/overview/javascript_library/all`
[18] `https://github.com/jquery/jquery/blob/master/MIT-LICENSE.txt`
[19] `http://learn.jquery.com/using-jquery-core/`
[20] `http://jqueryui.com/`
[21] `http://jquerymobile.com/`
[22] `https://plugins.jquery.com/`
[23] `http://googlecode.blogspot.de/2009/11/introducing-closure-tools.html`
[24] `https://code.google.com/p/closure-library/`

bietet. Der durchgängige Einsatz von Namensräumen, dem Klassenkonzept und den vom Compiler verwendeten JSDoc Annotationen, wirkt sich positiv auf die Struktur und Dokumentation der Implementierung aus. Neben dem Aufzeigen von (potenziellen) Fehlern zur Übersetzungszeit bietet der *Advanced Mode* unter anderem auch das Inlining von Funktionen und Dead Code Eliminiation an[25].

Benutzerschnittstelle

Zur Fragestellung Der Editor soll möglichst einfach in bestehende Systeme integriert werden können und zugleich eine komfortable und intuitive Benutzerschnittstelle bereit stellen. Es ist zu klären, ob die Software auf Basis einer bestehenden Technologie aufgebaut werden soll oder ob die Ziele mit einer vollständigen Eigenentwicklung zufriedenstellender verwirklicht werden können.

Entscheidung Als Benutzerschnittstelle wird der VisualEditor der Wikimedia Foundation verwendet.

Betrachtete Alternativen

Eigenentwicklung Eine vollständige Eigenentwicklung kann grundsätzlich auf zweierlei Arten realisiert werden:

contentEditable Mit dem *contentEditable* Attribut steht in HTML5 erstmals eine native Möglichkeit zur direkten Bearbeitung von HTML Elementen durch den Benutzer bereit. Die sich unterscheidenden Implementierungsdetails in den verschiedenen Browsern erzwingen jedoch eine gesonderte Behandlung dieser Inkonsistenzen und erfordern ein Abfangen aller Eingabebefehle, um ein über alle Browser hinweg konsistentes Ergebnis zu erhalten. Daneben ist zu berücksichtigen, dass die Bearbeitung von Wiki-Artikeln auf Basis eines WOM-Baumes nicht die beliebige Manipulierung aller im HTML Standard verfügbaren Elemente erlaubt, unter anderem, wenn es keine eins-zu-eins Entsprechung der WOM-Knoten zu HTML Elementen gibt, wie es beispielsweise bei den Image-Knoten der Fall ist, die auch eine Bildunterschrift beinhalten können und somit fester Bestandteil des umgebenden Blocks sind, sondern sind auf eine Teilmenge davon beschränkt. Auch diese Beschränkungen machen eine gesonderte Behandlung notwendig.

[25] `https://developers.google.com/closure/compiler/docs/compilation_levels`

JavaScript Ein weiterer Ansatz, um die zuvor genannten Inkonsistenzen zu umgehen, ist die reine JavaScript Implementierung, d.h. eine Art Emulierung der *contentEditable* Funktionen mittels JavaScript. Eine erfolgreiche Implementierung dieses Konzepts ist unter anderem bei Google Docs zu sehen.

Folgende, nicht vollständige Übersicht zeigt die grundlegenden Teilbereiche, die für eine Umsetzung zu berücksichtigen sind, wobei die Konzepte der Implementierung dabei nur auszugsweise angeführt werden, um einen Eindruck der Funktionsweise zu vermitteln:

Layoutengine Das gesamte Layout muss manuell durchgeführt werden, indem die zur korrekten Darstellung im Browser benötigten Elemente an die richtigen Stellen im DOM-Baum eingefügt werden. Elemente, die Text- oder Inlineelemente beinhalten, werden dabei innerhalb eines Zeilencontainers platziert, dessen Aufgabe unter anderem das Zeilenlayout ist.

Zeilen Um die Position von Zeilenumbrüchen herausfinden zu können, muss jedweder textueller Inhalt am Ende seines zur Verfügung stehenden Platzes manuell umgebrochen werden, wofür Zeilen in Form von `div`-Elementen Verwendung finden.

Eingabebehandlung Jegliche Eingaben über die Tastatur, innerhalb des virtuellen Dokumentes, müssen abgefangen und an der aktuellen Cursorposition eingefügt werden. Die Position des Textcursors wird dabei durch ein absolut positioniertes, blinkendes `div`-Element für den Benutzer ersichtlich.

Bei Mausklicks und der, sofern oberhalb von textuellem Inhalt vollzogenen, einhergehenden Änderung der Cursorposition muss zuerst der für das darunter liegende HTML Element zuständige WOM3-Knoten bestimmt werden, die Cursorposition berechnet und damit das zur Darstellung der Cursorposition verwendete `div`-Element neu positioniert werden.

Textselektion kann zum Beispiel über semi-transparente, oberhalb der als markiert zu kennzeichnenden HTML Elemente schwebende, `div`-Elemente angezeigt werden.

Da all dies jedoch viele Freiheiten bietet und auch die direkte Manipulation des zugrunde liegenden WOM3-Baums ermöglichen würde, hatte ich diesen Ansatz zuerst verfolgt, bin jedoch bei der Implementierung eines einfachen Prototyps zu der Erkenntnis gelangt, dass dies durch eine Vielzahl an Implementierungsdetails und der Behandlung von Sonderfällen erkauft werden muss, um ein für Anwender vertraut wirkendes Benutzungserlebnis erzeugen zu können.

Insbesondere im Hinblick der Erweiterbarkeit und Akzeptanz habe ich diesen Ansatz deshalb wieder fallen lassen.

Eclipse Orion Die Eclipse IDE zählt zu den verbreitetsten ihrer Art und unterstützt neben primär Java auch eine große Anzahl anderer Programmiersprachen, deren Unterstützung durch eine Vielzahl von Plugins ermöglicht werden.

Mit dem Orion Projekt hat die Eclipse Foundation sich zum Ziel gesetzt, eine einheitliche, durch ein umfangreiches Pluginsystem erweiterbare Plattform zur Webentwicklung zu schaffen, also ein Pendant zu den in der klassischen Softwareentwicklung eingesetzten IDEs, die an die Bedürfnisse des Webs ausgerichtet ist.[26]

```
<!DOCTYPE html>
<html>
<head>
<meta name="copyright" content="Copyright (c) IBM Corporation and others 2010,
<meta http-equiv="PRAGMA" content="NO-CACHE"/>
<meta http-equiv="Expires" content="-1"/>
<meta http-equiv='X-UA-Compatible' content='IE=EmulateIE7,IE=edge'/>
<title>Orion TextView Demo</title>
<link rel="stylesheet" type="text/css" href="demo.css" />
<!-- Note if running this standalone you will need to copy requirejs from bundl
<script src="../../requirejs/require.js"></script>
<script type="text/javascript">
    /*globals require*/
    require({
      baseUrl: '../..',
      paths: {
          text: 'requirejs/text',
          i18n: 'requirejs/i18n',
```

Abbildung 2.9: Eclipse Orion Editor

Die in Abbildung 2.9 dargestellte Editorkomponte basiert auf dem bereits zuvor angeführten *contentEditable* Feature. Syntaxhighlighting für gängige Sprachen im Webumfeld, sowie Zeilennummern und unter anderem Vim-Keybindings, erzeugen ein konsistentes Bild. Allerdings steht dabei klar die Bearbeitung von reinen Textdokumenten im Fokus der Entwicklung, was bedeutet, dass nativ unter anderem keine Auszeichnungen von Überschriften möglich sind und erst recht kein Einbinden von komplexeren Strukturen, wie Tabellen oder Bildern.

Closure Editor Die 2009 von Google veröffentlichten Closure Tools[27], bestehen, unter anderem, aus einem JavaScript Compiler sowie einer JavaScript Library, die Google selbst in einigen seiner Produkte verwendet.[28]

Letztgenannte Bibliothek beinhaltet dabei auch einen Texteditor auf Basis der *contentEditable* Technologie, der bereits alle grundlegenden Funktionen beinhaltet. In Abbildung 2.10 sind UI-Komponenten, wie Schaltfläche, Dialoge und eine Werkzeugleiste zu sehen, die eine für den Anwender konsistente Oberfläche erzeugen.

[26] https://wiki.eclipse.org/Orion/Project_mission_statement

[27] https://developers.google.com/closure/

[28] https://developers.google.com/closure/faq#gwt

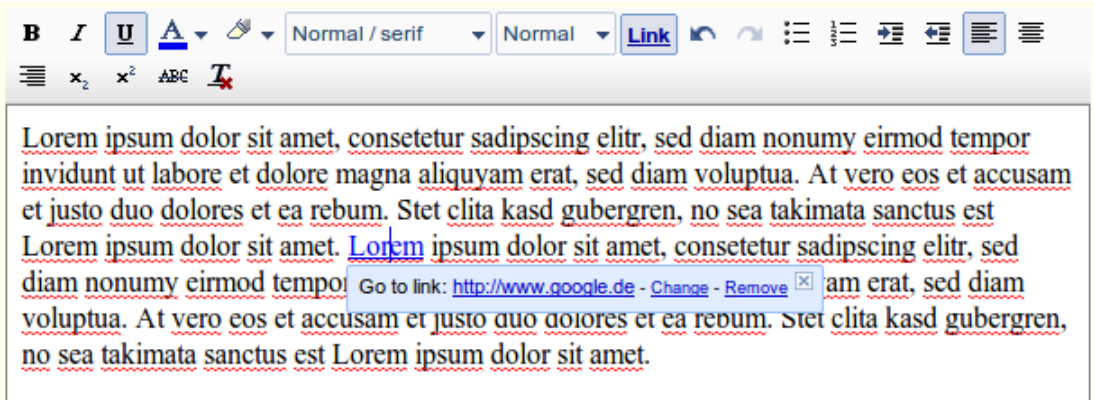

Abbildung 2.10: Google Closure Editor

Dabei werden auch die meisten Differenzen der unterschiedlichen Browserimplementierungen hinter einer konsistenten Schnittstelle verborgen. Durch die optionalen JSDoc Annotationen, der sich durch die an das Java angelehnte Klassenkonzept ergebende Codestruktur und dem Closure Compiler, der unter anderem Typkonsistenz und korrekte Syntax sicherstellen kann und nebenbei die Codegröße optimiert, können zahlreiche potenzielle Fehlerquellen bereits zur Übersetzungszeit gefunden werden, die ansonsten möglicherweise nur durch langwieriges Debuggen in Erscheinung getreten wären.

Die eben genannten Features bieten zwar aus Entwicklersicht einige Vorteile, allerdings wird die Editorkomponente nicht mehr aktiv weiterentwickelt oder es werden zumindest keine neue Komponenten als Open Source veröffentlicht, auch wenn der Editor noch in Google Produkten Verwendung findet. Dazu kommt, dass zwar die Browserdifferenzen weitestgehend verborgen werden, der Fokus aber auf der Bearbeitung von unstrukturiertem Text liegt und nur eine neue Schnittstelle für *contentEditable* bereit gestellt wird. Der im Folgenden vorgestellte Editor wurde dagegen von Anfang an mit dem Ziel entwickelt, strukturierte Dokumente zu bearbeiten, was den Implementierungsaufwand für einen Editor für das Wiki Object Model reduziert und aufgrund der aktiven Entwicklung auch zukunftsträchtiger ist.

Wikimedia VisualEditor Mit dem VisualEditor hat die Wikimedia Foundation im Jahr 2011 damit begonnen, einen visuellen Editor zur Bearbeitung von Artikeln der Wikipedia zu entwickeln. Dieser befindet sich bereits als Opt-in im Testbetrieb auf den Live-Seiten der Wikipedia.

Neben der MediaWiki Erweiterung[29], zur Bearbeitung von Artikeln in der MediaWiki-Software, steht auch eine, unter der MIT Lizenz veröffentlichte, eigenständige Version des Editors zur Verfügung, die gleichzeitig als Grundlage für die zuvor genannte Erweiterung dient.

[29] `http://www.mediawiki.org/wiki/Extension:VisualEditor`

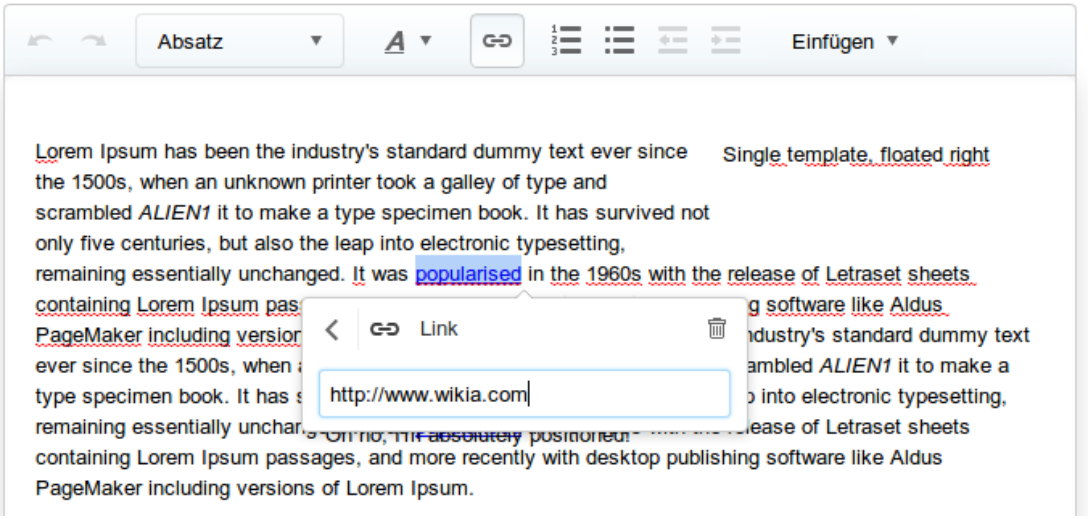

Abbildung 2.11: Wikimedia VisualEditor

Auch hier kommt die *contentEditable* Technologie zum Einsatz. Dabei sind die Entwickler jedoch unter anderem auch auf die zuvor aufgeführten Probleme gestoßen, haben diese jedoch inzwischen weitestgehend gelöst. So synchronisieren sich unter anderem das zugrunde liegende Datenmodell des Artikels und die HTML Repräsentation jeweils gegenseitig und Cursorbewegungen werden standardmäßig erlaubt, mit der erwarteten Position im Datenmodell verglichen und, wenn nötig, entsprechend korrigiert, um etwa komplexere Strukturen, deren Bearbeitung nicht erlaubt ist, zu überspringen. Daneben ist auch eine eigene UI Bibliothek Teil des Editors.

Für die zukünftige Entwicklung sind auch Features wie kollaboratives Editieren eines Artikels durch mehrere Benutzer geplant.

Da sich das Produkt aktuell in Entwicklung befindet und noch nicht für den produktiven Einsatz empfohlen wird, besteht allerdings die Gefahr, dass Schnittstellen oder gar der Aufbau des Gesamtsystems noch von größeren Änderungen betroffen sind und dadurch Anpassungen nötig werden.

Der VisualEditor bietet in der aktuellen Version noch keine Unterstützung für den Internet Explorer an. Dies soll sich nach Aussage der Entwickler aber in Zukunft noch ändern[30], was als potenzieller Standard-Editor der Wikipedia glaubwürdig erscheint, sofern diese nicht rund zehn Prozent ihrer Besucher[31] die Möglichkeit des Bearbeitens vorenthalten wollen.

Nichtsdestotrotz bietet der VisualEditor eine solide Grundlage und sollte eine Verwirklichung der gesteckten Qualitätsziele, innerhalb der untersuchten Kandidaten zur Auswahl der Benutzerschnittstelle, am besten unterstützen. Durch den angestrebten Einsatz als Standardeditor in der Wikipedia dürfte auch eine stetige Weiterentwicklung und Fehlerbehebung in der Zukunft gesichert sein.

[30] `http://www.mediawiki.org/wiki/VisualEditor/Target_browser_matrix`
[31] `http://stats.wikimedia.org/archive/squid_reports/2014-04/SquidReportClients.htm`

Wohingegen der VisualEditor als, ebenfalls unter der MIT Lizenz veröffentlichtes, MediaWiki Plugin bereits zu Beginn dieser Arbeit zur Verfügung stand, wurde die eingenständige Version erst viel später der Öffentlichkeit zur Verfügung gestellt und bei GitHub veröffentlicht. Da diese bereits die Bearbeitung aller grundlegenden Elemente innerhalb von HTML-Dokumenten unterstützt, habe ich mich schließlich für diese Alternative entschieden.

2.4.6 Implementierung

Nun, da die Softwarearchitektur und die zu verwendenden Technologien vorgestellt wurden, werde ich meine Implementierung erörtern, aufgeteilt nach den im Kapitel 2.4.4 vorgestellten Komponenten.

Da ich den Editor zu Beginn in der ersten im Abschnitt 2.4.5 vorgestellten Variante, einer vollständigen Eigenentwicklung, umsetzen wollte und erst nach der Implementierung eines Prototyps zu der Einsicht gekommen bin, dass dies im vorgegebenen Zeitrahmen nicht zu einem zufriedenstellenden Ergebnis führen würde, hatte ich bereits die Grundlagen für die *WOM3Tools* Komponente auf Basis der Closure Library geschaffen. Damit basiert die Implementierung dieser Komponente auch in der Endversion auf dem eben genannten Framework, wobei die Benutzerschnittstelle auf dem Wikimedia VisualEditor und der dort eingesetzten jQuery Bibliothek aufbaut.

Dies hat aber im Hinblick auf den Grundsatz der Separation of Concerns (Dijkstra, 1982) den Vorteil, dass die beiden Hauptkomponenten damit einer strikten Trennung unterliegen und auch der Overhead, der durch den Einsatz eines zweiten Frameworks für gewöhnlich entstehen würde, durch den Einsatz des Closure Compilers vernachlässigbar klein ist, da alle sich nicht in Verwendung befindlichen Teile der Bibliothek ohnehin nicht in der kompilierten Version der JavaScript Datei vorhanden sind.

Denkbar wäre es auch, auf die Konvertierung des im JSON Format vorliegenden WOM3-Baumes hin zu einer ihm entsprechenden HTML Repräsentation, gänzlich zu verzichten und letztere direkt auf Serverseite, durch Erweiterung des Sweble Parsers zu realisieren. Die aktuelle Version der WOM Implementierung in JavaScript hat den Vorteil, dass diese sowohl auf Client- als auch Serverseite, als Komponente eines *node.js* Servers, eingesetzt werden kann. Mit der Möglichkeit, die gesamte Logik auf Clientseite zu versammeln, ist es zukünftig auch problemlos möglich, einen universellen WOM Editor zu erstellen, der ohne größere Anpassungen, direkt mit mehreren unterschiedlichen, WOM-Dokumente bereitstellenden Schnittstellen kommunizieren kann, ohne weitere Anpassungen auf Serverseite durchführen zu müssen

In der aktuellen Version des Prototyps habe ich mich jedoch dazu entschieden, die Konvertierung zunächst auf Serverseite zu belassen, da der Sweble Parser zum Zeitpunkt der Implementierung nur in einer als lokal ausführbaren Version zur Verfügung stand und nicht direkt eine Webschnittstelle zum Abruf von Dokumenten anbot.

WOM3Tools

Für die interne Handhabung der WOM3 Datenstrukturen habe ich zunächst die für Java vorhandene WOM3-Schnittstellenbeschreibung auf JavaScript portiert und anschließend auf Basis der zur Verfügung gestellten Java Implementierung eine, alle für den aktuellen Anwendungsfall erforderlichen Funktionen beinhaltende, WOM3-Implementierung erstellt.

WOM3Parser Die Aufgabe des WOM3Parsers besteht darin, die im JSON Format serialisierten Daten des WOM3 Baumes zu parsen und daraus entsprechende Objekte der WOM3-Implementierung zu instanziieren.

Der Sweble Parser stellt aktuell zwei unterschiedliche Versionen der JSON Repräsentation bereit. Die Darstellung im *pretty* Format ist in erster Linie für die menschliche Interaktion gedacht, wie in Listing 2.4 zu sehen. Mit *compact* kann dagegen die Größe der zur Darstellung benötigten Zeichen reduziert werden und damit auch die Dauer der Übertragung, was jedoch auch auf die Übersichtlichkeit zutrifft.

```
{ "!type": "article",
  "@xmlns": "http://sweble.org/schema/wom30",
  "@version": "3.0",
  "@title": "Test.wikitext",
  "@xmlns:mww": "http://sweble.org/schema/mww30",
  "!children": [{
      "!type": "body",
      "!children": [{
          "!type": "p",
          "@topgap": "0",
          "@bottomgap": "0",
          "!children": [{
              "!type": "b",
              "!children": [{
                  "!type": "rtd",
                  "!children": [{
                      "!type": "#text",
                      "!value": "'''"
                  }]
                }, {
                  "!type": "text",
                  "!children": [{
                      "!type": "#text",
                      "!value": "Bold"
                  }]
                }, {
                  "!type": "rtd",
                  "!children": [{
                      "!type": "#text",
                      "!value": "'''"
}]}]}]}]}]}]}
```

Listing 2.4: WOM3 Pretty Serialisierung

Der WOM3Parser entscheidet auf Basis der seiner *parse*-Methode übergebenen Daten selbst, um welches der beiden Formate es sich handelt und führt anschließend das Parsen durch die für das Format ermittelte, konkrete Implementierung durch. Für die Umsetzung bietet sich das Strategie Entwurfsmuster (Gamma, Helm, Johnson & Vlissides, 1995) an.

WOM3Serializer Der WOM3Serializer hat die dazu diametrale Aufgabe, den als Instanzen der Klassen der WOM3-Implementierung vorliegenden Baum wieder in eine, demselben Baum entsprechende, JSON Repräsentation zu überführen.

WOM3ToHTMLConverter Der VisualEditor verlangt eine HTML Repräsentation des zu bearbeitenden Dokumentes. Dabei ist es wichtig, dass alle im WOM3-Baum enthaltenen Informationen auch weiterhin vorhanden sind, um eine verlustfreie Rücktransformation nach erfolgter Bearbeitung zu ermöglichen.

HTMLToWOM3Converter Nach der Bearbeitung durch den VisualEditor liegt ein gegebenenfalls modifiziertes HTML Dokument vor, das nun geparst und zurück in die interne WOM3 Repräsentation überführt werden muss.

Aufgrund der Trennung der WOM3Tools-Komponente von der Benutzerschnittstelle kann das Parsing nicht durch eine Traversierung des DOM-Baumes erfolgen und muss stattdessen mithilfe eines HTML Parsers durchgeführt werden.

Beispiel Zur Veranschaulichung der Funktionalität der einzelnen Komponenten zeige ich im folgenden einen beispielhaften Ablauf auf.

Das Wikitext Dokument in Listing 2.5 wird zunächst dem Sweble Parser übergeben, der daraus die in Listing 2.4 dargestellte JSON-Serialisierung des entsprechenden WOM-Dokumentes erzeugt.

```
1 '''Bold'''
```

Listing 2.5: Wikitext Dokument

Anschließend wird die JSON-Repräsentation vom WOM3Parser eingelesen. Das daraus instantiierte WOM-Dokument überführt der WOM3ToHTMLConverter in eine dazu äquivalente, im Listing 2.6 dargestellte, HTML Repräsentation.

```
<html>
  <head></head>
  <body data-wom=\"{"attributes"
     :{"version":"3.0","xmlns":"http://sweble.org/schema/wom30","
     title":"Test.wikitext","xmlns:mww":"http://sweble.org/schema
     /mww30"},"rtd":{"pre":null,"post":null},"type":"article"}\">
    <p data-wom=\"{"attributes":{"topgap":"0","bottomgap
       ":"0"},"rtd":{"pre":null,"post":null},"type":"p"}\">
      <b data-wom=\"{"attributes":{},"
         rtd":{"pre":{"!type":"rtd","!children":[{"!type":"#text"
         ,"!value":"'''"}]},"post":{"!type":"rtd","!children":[{"
         !type":"#text","!value":"'''"}]}},"type":"b"}\">Bold</b>
    </p>
  </body>
</html>
```

Listing 2.6: HTML Repräsentation

Die in den jeweiligen WOM-Knoten enthaltenen Informationen, wie Attribute und *RTD*-Elemente, werden im *data-wom* Attribut des zugehörigen HTML-Elementes, in Form eines serialisierten JSON-Objekts gespeichert. Listing 2.7 zeigt dazu den möglichen Inhalt eines Bold Elementes.

```
{
  "attributes": {},
  "rtd": {
    "pre": {
      "!type":"rtd",
      "!children": [{
        "!type": "#text",
        "!value": "'''"
      }]
    },
    "post": {
      "!type": "rtd",
      "!children": [{
        "!type": "#text",
        "!value": "'''"
      }]
    }
  },
  "type":"b"
}
```

Listing 2.7: Inhalt des *data-wom* Attributes

Für weitere Implementierungsdetails verweise ich auf Kapitel 3.1.

Benutzerschnittstelle

Der Wikimedia VisualEditor besitzt bereits die grundlegenden Funktionen, die ein ernst zu nehmender visueller Editor beinhalten sollte.

Für die Implementierung der WOM3 spezifischen Erweiterungen werde ich im Folgenden einen kurzen Überblick über die den für das Verständnis notwendigen, dem VisualEditor zugrunde liegenden Konzepte geben[32].

Die Architektur des Editors besteht im wesentlichen aus den folgenden drei Komponenten:

DataModel - dm Das dem Editor übergebene HTML Dokument wird zuerst in ein lineares Datenmodell überführt. Dazu wird der gesamte textuelle Inhalt des Dokumentes in einem einzelnen linearen Adressraum abgebildet, in dem jedes Zeichen eine eindeutige Adresse und, sofern Kind eines oder mehrerer Inline Elemente, Verweise auf entsprechende Annotationen besitzt.

ContentEditable - ce Aus dem linearen Datenmodell wird nun das für den Anwender sichtbare und bearbeitbare Dokument erzeugt, indem eine entsprechende DOM Struktur generiert und anschließend das durch den Browser bereitgestellte *contentEditable* Feature aktiviert wird.

Um die im Abschnitt 2.4.5 angesprochenenen Probleme, wie der Beschränkung der möglichen Bearbeitungsmöglichkeiten bei komplexeren Elementen, zu umgehen, wird nur eine eingeschränkte Menge der zur Verfügung stehenden Operationen zur Bearbeitung innerhalb des *contentEditable* Feldes erlaubt, bzw. im Nachhinein entsprechend korrigiert. So kann ein Bildelement mitsamt Bildunterschrift von der direkten Bearbeitung ausgenommen werden, sodass Modifikationen daran nur noch über ein gesondertes Dialogfenster vorgenommen werden können und, um beim Bewegen des Textcursors nicht in den Bereich der Bildunterschrift zu gelangen, wird der Textcursor sobald er auf ein Bild trifft, hinter dem Bildelement platziert.

UserInterface - ui Das UserInterface stellt die Komponenten, die der Anwender zur Interaktion mit dem Editor verwendet, bereit.

Toolbar Die Werkzeugleiste am oberen Bildschirmrand und die darin enthaltenen Schaltflächen und Menüs ermöglichen einen schnellen Zugriff auf die wichtigsten Funktionen, wie beispielsweise verschiedene Formatierungsmöglichkeiten (bold, italics, usw.), oder eine Schaltfläche, um das Dokument zu speichern.

Dialoge Mit Hilfe von Dialogen können Fenster über dem zu bearbeitenden Dokument eingeblendet werden, um zum Beispiel die für die Darstellung eines Bildes verfügbaren Parameter anpassen zu können.

[32] `http://www.mediawiki.org/wiki/VisualEditor/Software_design#Architecture`

Inspektors Mit Inspektors steht eine weitere Möglichkeit bereit, bestimmte Parameter eines Elementes zu verändern, ohne dafür einen eigenen Dialog erstellen zu müssen. Stattdessen werden diese direkt neben dem zugehörigen Element eingeblendet und überdecken nicht den gesamten Arbeitsbereich. Dies wird unter anderem standardmäßig zur Bearbeitung von Linkzielen eingesetzt.

Für weitere Details der Implementierung der Benutzerschnittstelle verweise ich auf Kapitel 3.3.

2.5 Ergebnisse

Nach der erfolgten Implementierung werde ich nun das Endergebnis vorstellen. Die beiden Komponenten werden dabei getrennt voneinander behandelt.

2.5.1 Server

Bei der aktuellen Serverimplementierung handelt es sich ausschließlich um einen Prototyp, der die für den Client erforderlichen Daten bereit stellt. So soll der Sweble Parser in Zukunft direkt über ein eigenes REST-Interface erreichbar sein, womit der im Moment bei jedem Aufruf durchgeführte Akt des Startens einer lokalen JVM, um den als JAR-Datei vorliegenden Parser auszuführen, entfallen könnte.

Durch den Einsatz von JavaScript auf Serverseite können die WOM3Tools in Zukunft auch komplett auf Clientseite eingesetzt werden, sodass der Browser des Benutzers direkt mit der REST-API des Sweble Parsers oder einer anderen Quelle für WOM-Dokumente interagieren kann.

Neben dem eigentlichen Abruf des zu bearbeitenden Artikels wird die MediaWiki-API auch für die Abfrage weiterer Informationen direkt angesprochen (Linkziele, Bildinformationen). Für eine Unterstützung unterschiedlicher Wiki-Softwares müsste dies noch an passender Stelle abstrahiert und im Idealfall durch die vom Sweble Parser bereitgestellte REST-API unterstützt werden.

2.5.2 Client

Die Benutzerschnittstelle besteht im wesentlichen aus zwei Bereichen, der sich am oberen Bildschirmrand befindlichen Werkzeugleiste zur einfachen Auswahl von Operationen und des eigentlichen Bearbeitungsbereichs, wie in Abbildung 2.12 zu sehen, der den Großteil des zur Verfügung stehenden Bildschirminhalts einnimmt. Die im Folgenden vorgestellten Komponenten waren bereits im VisualEditor oder der darauf basierenden MediaWiki Erweiterung enthalten und wurden von mir zur Verwendung im WOM Editor entsprechend angepasst.

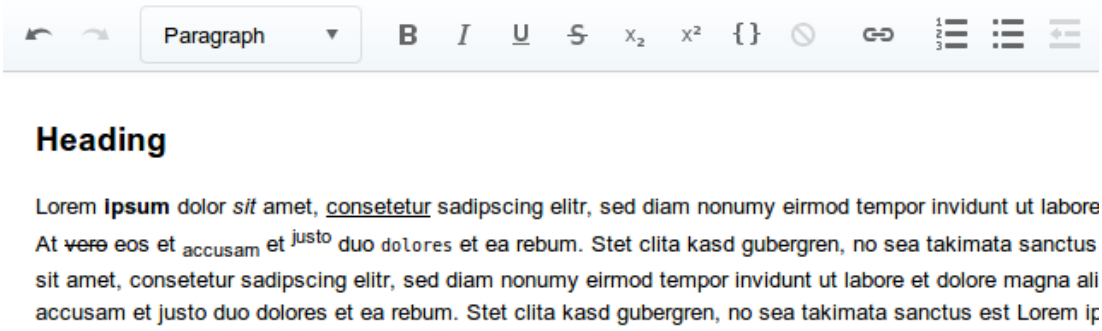

Abbildung 2.12: WOM3 VisualEditor Übersicht

Abbildung 2.13 zeigt die Möglichkeiten, textuellen Inhalt durch Formatierungen auszuzeichnen, was sowohl über die entsprechenden Schaltflächen in der Werkzeugleiste als auch über die gängigen Tastenkürzel vorgenommen werde. So können Listen beispielsweise mittels Tabulator eingerückt werden.

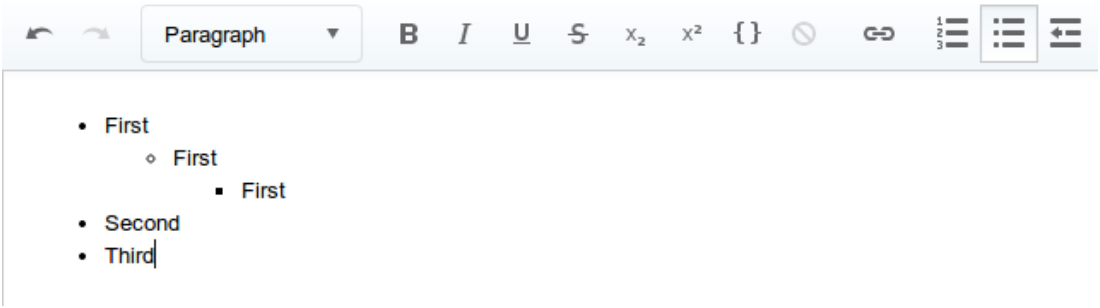

Abbildung 2.13: WOM3 VisualEditor Listen

Bei der Bearbeitung von Links aller Art erfolgt die Änderung des Titels direkt durch Platzierung des Cursors innerhalb des Elements. Um das Ziel zu modifizieren steht ein Dialogfenster bereit, das in Abbildung 2.14 abgebildet ist, und dem Benutzer während der Eingabe bereits Vorschläge für mögliche interne Linkziele vorschlägt.

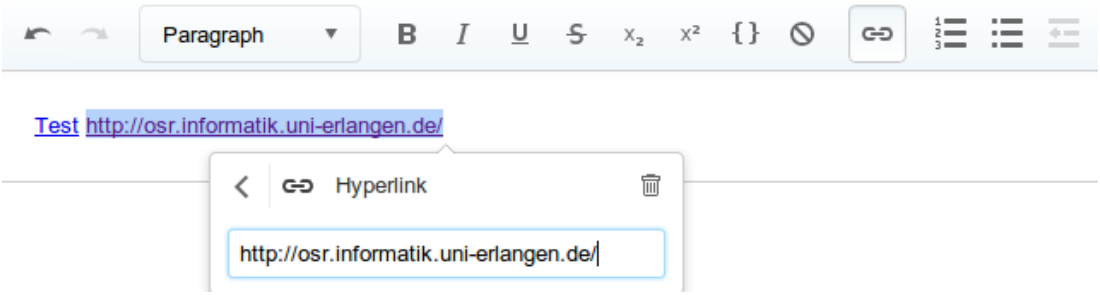

Abbildung 2.14: WOM3 VisualEditor Links

Abbildung 2.15 zeigt, wie Bilder direkt mithilfe der Maus in ihrer Größe verändert werden können. Für weitergehende Bearbeitungen, wie dem Hinzufügen einer Bildunterschrift, steht ein Dialogfenster bereit, das Zugriff auf alle Parameter bietet.

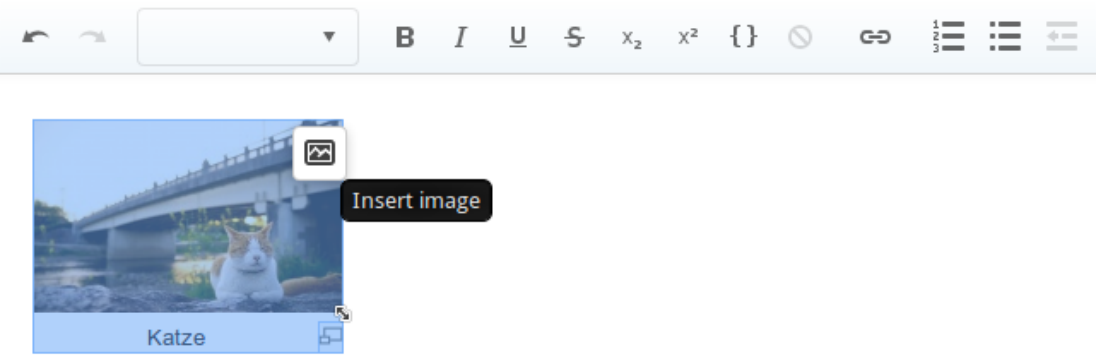

Abbildung 2.15: WOM3 VisualEditor Bilder

Im Rahmen dieser Bachelorarbeit konnten noch nicht alle in der WOM3 Spezifikation enthaltenen Elemente vollständig implementiert werden und so enthält der Editor aktuell nur einige grundlegende Bearbeitungsfunktionen, um das Konzept und die Softwarearchitektur zu veranschaulichen. Die Unterstützung von Tabellen und Templates fehlt zum Beispiel noch gänzlich.

2.6 Diskussion der Ergebnisse

Im Abschnitt 2.4.1 wurden verschiedene Qualitätsziele vorgestellt, die für die Umsetzung eines Editors von Bedeutung sind. Um zu klären, ob die in den letzten Kapiteln vorgestellte Softwarearchitektur und die daraus hervorgegangene Implementierung diesen Anforderungen gerecht wurden, werde ich im folgenden die Ziele noch einmal separat diskutieren und evaluieren.

Lokalität von Änderungen Die Bearbeitung von vollständig unterstützten WOM Dokumenten kann ohne Verlust von Informationen durchgeführt werden. Bei einfachen, nur aus einem einzelnen Tag bestehenden Elementen, können die RTD Informationen auch über eine Bearbeitung des Inhalts derer hinweg erhalten werden. Einzig die Bearbeitung komplexerer Elemente, wie etwa Bilder, die sich aus mehreren HTML Tags zusammensetzen, werden die *RTD* Informationen nach erfolgter Änderungen verworfen.

Da dies aber nur ebenjene Elemente betrifft, die ohnehin einer Modifikation des Benutzers unterlagen und alle anderen Informationen weiterhin erhalten bleiben, entspricht dies dem gesteckten Ziel.

Bedienbarkeit Die Editorkomponente orientiert sich sowohl beim Aufbau als auch bei der Bedienung an den Bedienkonzepten der gängigen Textverarbeitungsprogramme wie Open-/LibreOffice oder Word. Durch die Verwendung der *contentEditable* Technologie, auf die der VisualEditor aufbaut, können die zu erwartenden Resultate, die ein Benutzer von bestimmten, bekannten Aktionen annimmt, innerhalb der betrachteten Implementierungsalternativen, am besten erfüllt werden. Dies wäre zwar potenziell auch bei dem Ansatz des reinen JavaScript-Editors möglich

gewesen, allerdings nur unter einem nicht im Verhältnis zum Nutzen stehenden zeitlichen Aufwand, um überhaupt an die Funktionen der nativen Browserlösung heran zu reichen, geschweige denn diese zu übertreffen.

Die gängigsten Möglichkeiten zur Textauszeichnung sind direkt sowohl über die stets am oberen Bildschirmrand schwebende Werkzeugleiste als auch mittels Standard Tastenkombinationen erreichbar. Daneben ist die Bearbeitung von komplexeren Objekten wie etwa Bildern bequem über Dialoge möglich, die einen direkten Zugriff auf alle vorhandenen Optionen bieten.

Damit kann jeder Benutzer, der sich im Stande sieht ein modernes Textverarbeitungsprogramm zu bedienen, auch mit diesem Editor direkt umgehen können, ohne größeren Einarbeitungsaufwand.

Wartbarkeit Die WOM3Tools Komponente ist aufgrund der Verwendung des Closure Compilers durchgängig mit JSDoc Annotationen ausgestattet, was neben der Typüberprüfung und der damit einhergehenden Fehlerreduzierung auch die automatische Generierung einer API Dokumentation ermöglicht. Daneben geben die Compiler Warnungen gute Hinweise auf potenziell fehlerhafte Codestellen zurück, was insbesondere bei größeren JavaScript Projekten die für die Fehlersuche aufzuwendende Zeit stark verringern kann.

Auf Seiten der Benutzerschnittstelle zeigt sich der VisualEditor als durchgängig mittels JSDoc Annotationen bestückt und auch die zahlreichen Kommentare ermöglichen eine rasche Einarbeitung und anschließende Erweiterung. Neben der API Dokumentation als HTML Ausgabe finden sich die Ideen und zugrunde liegenden Konzepte auch als Dokumentation im Wiki des VisualEditor Projekts[33].

Allerdings ist der VisualEditor noch nicht in einer stabilen Version veröffentlicht worden und ist so entwicklungsbedingt noch teils größeren Änderungen unterworfen, was zumindest kleinere Anpassungen bei einem Update recht wahrscheinlich werden lässt.

Durch die geplante Verwendung des VisualEditors als Standard-Editor für die Wikipedia sollte eine stetige Weiterentwicklung jedoch auch in Zukunft gewährleistet sein.

Portierbarkeit Eines der Ziele des Wiki Object Models ist die weitestgehende Abstraktion von der zugrunde liegenden Wiki-Software. Dies hat zur Folge, dass der Editor im Idealfall überhaupt nicht angepasst werden muss, um eine neue Wiki-Software zu unterstützen. Einzig mögliche plattformspezifische Erweiterungen, wie es unter anderem bei den MediaWiki Transclusions oder den speziellen internen Links der Fall ist, müssen sowohl in den WOM3Tools, also der WOM3-JavaScript Implementierung, den verschiedenen HTML Convertern, dem Serialisierer und Parser implementiert werden, als auch direkt in der VisualEditor Komponente.

Der Editor besitzt aktuell zweierlei Komponenten, die direkt auf die API die der Daten zugrunde liegenden MediaWiki Instanz zugreift und zwar zum Abruf von Bildinformationen, also der Höhe, Breite und der eigentlichen Bilddateien

[33] `http://www.mediawiki.org/wiki/VisualEditor/Portal`

bzw. automatisch generierten Thumbnails davon und dem Abfragen von Artikelnamen. Diese müssten bei einem Wechsel entsprechend angepasst werden oder bei Bedarf weiter abstrahiert werden, um mehrere Wiki-Softwares parallel zu unterstützen.

2.7 Ausblick

Mit dem im Rahmen dieser Bachelorarbeit entwickelten und auf den vorhergehenden Seiten vorgestellten Editor zur Bearbeitung von Wikiartikeln auf Basis von WOM3-Dokumenten, existiert nun eine solide Grundlage, auf die in Zukunft aufgebaut werden kann.

Da nicht alle in der aktuellen Version der WOM3 Spezifikation enthaltenen Elemente vollständig implementiert werden konnten, bietet es sich als nächster Schritt an, dies für die noch fehlenden Elemente nachzuholen.

Insbesondere die Unterstützung von Tabellen fehlt noch gänzlich. Auch der VisualEditor selbst bietet aktuell noch keine Möglichkeit zur Bearbeitung von Tabellen an, auch wenn die Darstellung prinzipiell jetzt schon möglich ist. Dies wird sich aller Voraussicht nach aber in absehbarer Zukunft noch ändern.

Abschließend steht noch die Implementierung eines Interfaces für die bereits in der Einleitung dieser Arbeit erwähnten und in Dohrn und Riehle (2013) im Detail vorgestellten Refactoring Methoden auf WOM Dokumenten aus. Mittels der im VisualEditor enthaltenen UI-Bibliothek und ihrer Komponenten sollte dies jedoch auch für umfangreichere Methoden problemlos möglich sein.

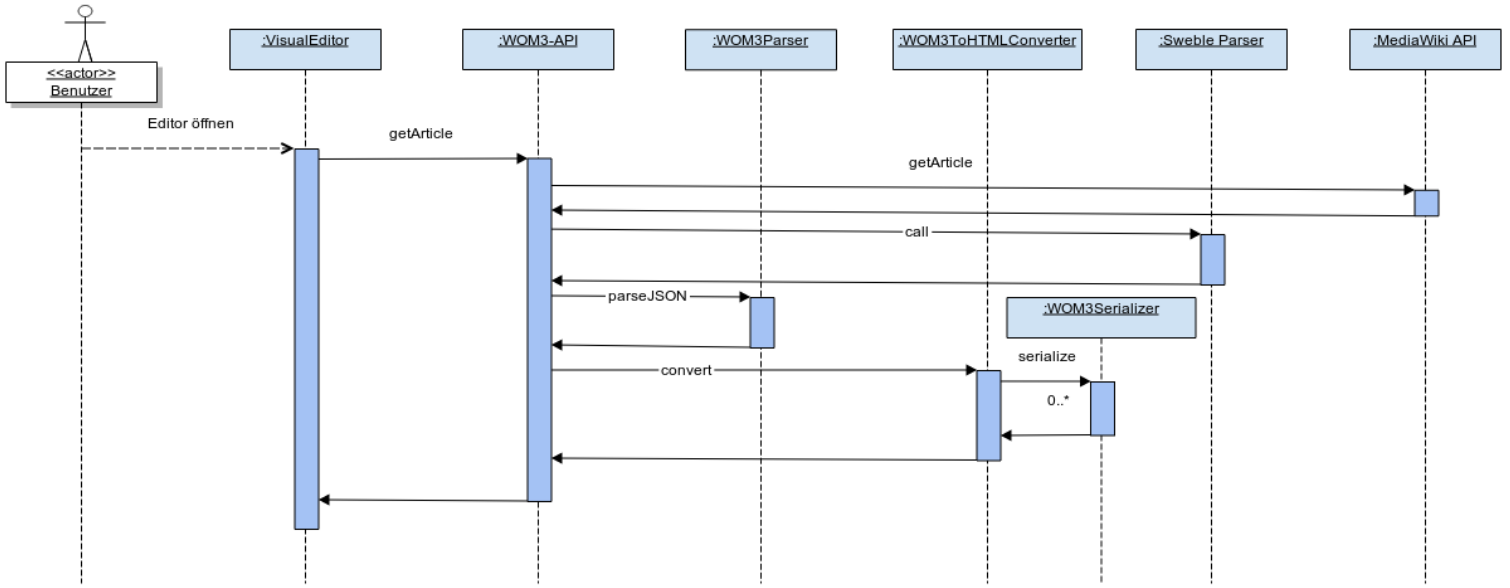

Abbildung 2.7: Sequenzdiagramm für den getArticle() Methodenaufruf

3 Ausarbeitung der Forschung

Der vorhergehende Teil dieser Arbeit hat dazu gedient, einen groben Überblick über das Gesamtsystem zu geben. Im nun folgenden Abschnitt werde ich auf weitere Details der Implementierung eingehen.

3.1 WOM3Tools

WOM3Parser Der WOM3Parser wählt anhand des Typs des aktuell zu bearbeitenden Knotens, wie in Listing 3.1 zu sehen, die passende Parserfunktion aus. Diese erzeugt ein neues WOM3 Domänenobjekt, fügt zunächst die Attribute hinzu und kümmert sich anschließend um die Erzeugung der Kindelemente.

Zur Bestimmung des Knotentyps wird das *!type* Attribut aus dem Objekt des aktuellen Knotens verwendet (Zeile 2).

```
1  womtools.core.WOM3PrettyParser
       .parseNode_ = function(doc, nodeData, callback) {
2      var nodeType = nodeData['!type'];
3      var parser;
4
5      if (!goog.isDef(nodeType)) {
6          return callback('No node type found', null);
7      } else if
           (womtools.core.WOM3PrettyParser.nodeParser_[nodeType]) {
8          parser = womtools
               .core.WOM3PrettyParser.nodeParser_[nodeType];
9
10         goog.bind(parser, womtools.core.Wom3PrettyParser
               , doc, nodeData, function(err, data) {
11             if (err) {
12                 callback(err, null);
13             } else {
14                 callback(null, data);
15             }
16         })();
17     } else {
18         callback
               ('Error: ' + nodeType + ' not implemented', null);
19     }
20 };
21 ...
```

```
22 womtools.core.WOM3PrettyParser.nodeParser_ = {
23     'abbr': womtools.core.WOM3PrettyParser.parseNodeAbbr_,
24     'big': womtools.core.WOM3PrettyParser.parseNodeBig_,
25     'bold': womtools.core.WOM3PrettyParser.parseNodeBold_,
26     ...
27 };
```

Listing 3.1: Auswahl der Parserfunktion

WOM3Serializer Die Serialisierungskomponente traversiert mit der in Listing 3.2 dargestellten Funktion, im Wesentlichen den gesamten Baum, beginnend beim Wurzelknoten in Tiefensuche, indem zuerst die Attribute in ein Objekt mit dem Attributnamen als Schlüssel und dem Attributwert als Wert, und anschließend die Kindelemente in einem Array gespeichert werden.

```
1  womtools.core.WOM3PrettySerializer
       .prototype.parseWomNode_ = function(womNode) {
2      var obj = {};
3      goog.object.add(obj, '!type', womNode.getWomName());
4
5      if (!womNode.getFirstAttribute_) console.dir(womNode);
6      this.parseAttributes_(obj, womNode.getFirstAttribute_());
7      this.parseChildren_(obj, /** @type {womtools.core.wom3
           .impl.BackboneWomElement} */ (womNode.getFirstChild()));
8
9      return obj;
10 };
```

Listing 3.2: Serialisierung

WOM3ToHTMLConverter Die Überführung in eine HTML Repräsentation des WOM-Dokumentes basiert im wesentlichen auf dem in der Closure Library enthaltenen Templatesystem[1], um das anschließend zur Bearbeitung verwendete HTML Dokument als einzelnen String aufzubauen.

Alternativ hätte die Konstruktion auch rein durch JavaScript Aufrufe durchgeführt werden können, was sich aber negativ auf die Übersichtlichkeit und Geschwindigkeit ausgewirkt hätte.

Für einfache WOM Elemente, also Elemente, die nur aus einem Tag, Attributen und Kindelementen bestehen (Bold, Paragraph, etc.), findet das Template in Listing 3.3 Verwendung.

```
1 {template .basicElement}
2 <{$data.tag} data-wom="{$data.wom|escapeHtml}">
3     {$data.children.body}
4 </{$data.tag}>
5 {/template}
```

Listing 3.3: BasicElement Template

[1] https://developers.google.com/closure/templates/

Der HTML5 Standard erlaubt erstmals die Speicherung zusätzlicher, benutzerdefinierter Attribute in HTML Elementen, die *data*-Attribute[2].

Innerhalb des *data-wom* Attributes werden die Attribute des WOM-Elementes, sowie potenziell vorhandene *RTD* Elemente serialisiert, sofern diese das erste und / oder letzte Kindelement darstellen. Wahlweise kann auch das gesamte WOM3 Element in serialisierter Form darin eingebettet werden, um eine spätere Rekonstruktion zu erleichtern.

Zur Serialisierung einzelner Elemente wird auf die WOM3Serializer Komponente zurückgegriffen.

Für komplexere Elemente, die sich nicht auf einzelne DOM Elemente abbilden lassen, wie etwa Bilder, existieren dedizierte Templates.

Das in Listing 3.4 aufgezeigte Template für Bildelemente basiert auf der von der MediaWiki Erweiterung verwendeten Struktur[3].

```
1  {template .image}
2  {if $isBlockElement}<figure{else}<span{/if} typeof
       ="{$imageType}" data-womtype="image" class="{$classes}">
3
4      {if $isLink}<a href="{$resource}">{else}<span>{/if}
5      <img resource="{$resource}" src="{$src}"
6          {if $width} width="{$width}"{/if}
7          {if $height} height="{$height}"{/if}
8      />
9      {if $isLink}</a>{else}</span>{/if}
10
11
12     {if $hasCaption}
13         {if $isBlockElement}<figcaption>{else}<span>{/if}
14             {$caption}
15         {if $isBlockElement}</figcaption>{else}</span>{/if}
16     {/if}
17
18 {if $isBlockElement}</figure>{else}</span>{/if}
19 {/template}
```

Listing 3.4: Template Image-Element

Je nachdem, ob es sich bei dem Bild um ein Block- oder Inlineelement handelt, wird ein `figure` oder `span` Element erzeugt (Zeile 2), in dem bei Bedarf auch eine Bildunterschrift angezeigt werden kann (Zeile 12-16).

HTMLToWOM3Converter Der HTMLToWOM3Converter erwartet eine durch den HTML Parser htmlparser2[4] erstellte DOM Repräsentation des editierten HTML Dokumentes.

Anhand des jeweiligen Tags wird die erforderliche Parser Methoden ausgewählt

[2] `https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/Using_data_attributes`
[3] `http://www.mediawiki.org/wiki/Parsoid/MediaWiki_DOM_spec#Images`
[4] `https://github.com/fb55/htmlparser2`

und, sofern eindeutig, direkt ein entsprechendes WOM3 Domänenobjekt erzeugt.

Überschriften markieren im Wiki Object Model stets den Beginn eines neuen Abschnitts und sind Teil eines Section-Elementes. Ein Section-Element besteht aus Heading- und Body-Element. Bei der Abbildung auf ein HTML-Dokument gilt es zu beachten, dass HTML keine eigenen Section-Elemente enthält.

Beim Auftreten von HTML Überschriften muss im WOM-Dokument ein neues Section-Element erstellt werden, bei dem alle nachfolgenden Elemente, die nicht Teil einer höherwertigen Section sind, innerhalb dieser Section als Kindelemente vorhanden sind, erscheinen. Da dies im DOM Baum aber nur als flache Hierarchie abgebildet werden kann, muss bei einer Überschrift zunächst der passende Elternknoten innerhalb des WOM3 Baumes gefunden werden.

Listing 3.5 zeigt den Mechanismus zum Finden des Elternelementes einer Überschrift. Die Hierarchie wird solange nach oben hin abgelaufen, bis ein Body-Element gefunden wird (Zeile 2). Ein Body-Element kann nur Kind eines Artikel-Elementes, was gleichzeitig die Wurzel des Baumes darstellt (Zeile 5), oder eines Section-Elementes sein. Im letzteren Fall wird der Level der Section mit dem Level der einzufügenden Überschrift verglichen und das entsprechende Elternelement ausgewählt (Zeile 9-16).

```
1  while (!goog.isNull(parent)) {
2      if (parent instanceof womtools.core.wom3.impl.BodyImpl) {
3          parentSection = parent.getParentNode();
4
5          if (parentSection
               instanceof womtools.core.wom3.impl.ArticleImpl) {
6              parentBody = parent;
7              break;
8          } else if (parentSection
               instanceof womtools.core.wom3.impl.SectionImpl) {
9              if (parentSection.getLevel() == level) {
10                 parentBody = parentSection.getParentNode();
11             } else if (parentSection.getLevel() < level) {
12                 parentBody = parentSection.getBody();
13                 break;
14             } else {
15                 // continue
16             }
17         } else {
18             console.log('this should not have happened..');
19             break;
20         }
21     }
22
23     parent = parent.getParentNode();
24 }
```

Listing 3.5: Behandlung von Überschriften

3.2 Server

Der Server baut auf der *node.js* Software auf und besteht im wesentlichen aus den Controllern in Abschnitt 3.2.1, die für die Bereitstellung der API Funktionalität über eine Web-Schnittstelle hin zuständig sind und den Services in Abschnitt 3.2.2, zur Kapselung von Funktionalität, die austauschbare Implementierungen hinter einer einheitlichen Schnittstelle verbergen (Fowler, 2003), um beispielsweise den Zugriff auf andere Datenquellen als die MediaWiki-API zu ermöglichen, ohne weitere Teile der Anwendung modifizieren zu müssen.

3.2.1 Controller

Es folgt eine Übersicht aller Controller, der erwarteten Aufrufparameter und daraus resultierenden Rückgabewerten.

getArticle Erwartet den Artikelnamen als GET-Parameter *title* und liefert die aus dem WOM-Dokument des Artikels generierte HTML Repräsentation zurück.

```
{
  "visualeditor": {
    "content": "<html><head></head><body data
       -wom=\"...\"><p data-wom=\"...">abc</p></body></html>"
  }
}
```

Listing 3.6: Beispielaufruf */getArticle?page=Test*

imageDimensions Erwartet den Namen des Bildes als GET-Parameter *titles* und liefert die Breite und Höhe des Bildes zurück.

```
{
  "pageids": [ "3" ],
  "pages": {
    "3": {
      "pageid": 3,
      "ns": 6,
      "title": "Datei:The flowers.jpg",
      "imagerepository": "local",
      "imageinfo": [{
        "size": 828724,
        "width": 1500,
        "height": 1000,
        "mediatype": "BITMAP"
      }]
    }
  }
}
```

Listing 3.7: Beispielaufruf */imageDimensions?titles=Datei:The flowers.jpg*

imageInfo Erwartet den Namen des Bildes als GET-Parameter *titles*, optional die gewünschten Dimensionen eines Thumbnails in den Parametern *urlwidth* bzw. *urlheight* und liefert alle für das Bild vorhandenen Informationen, sowie die URL zu einem Thumbnail mit den übergebenen Ausmaßen davon zurück.

```
{
  "normalized": [{
    "from": "File:Katze.jpg",
    "to": "Datei:Katze.jpg"
  }],
  "pages": {
    "4": {
      "pageid": 4,
      "ns": 6,
      "title": "Datei:Katze.jpg",
      "imagerepository": "local",
      "imageinfo": [{
        "thumburl": "...",
        "thumbwidth": 180,
        "thumbheight": 113,
        "url": "...",
        "descriptionurl": "..."
      }]
    }
  }
}
```

Listing 3.8: Beispielaufruf */imageInfo?titles=File:Katze.jpg&urlwidth=180*

imagesByPrefix Erwartet eine GET-Variable *search* als Prefix und liefert eine Liste aller Bilder zurück, deren Name mit diesem beginnt.

```
{
  "pages": {
    "3": {
      "pageid": 3,
      "ns": 6,
      "title": "Datei:The flowers.jpg",
      "imagerepository": "local",
      "imageinfo": [{
        ...
      }]
    }
  }
}
```

Listing 3.9: Beispielaufruf */imagesByPrefix?search=The*

openSearch Erwartet eine GET-Variable *search* als Prefix und liefert eine Liste der Namen aller Artikel zurück, die mit ebendiesem beginnen.

```
[
  "T",
  [ "Test" ]
]
```

Listing 3.10: Beispielaufruf */openSearch?search=T*

pagesByPrefix Erwartet eine GET-Variable *search* als Prefix und liefert eine Liste aller Artikel zurück, deren Name mit diesem beginnt.

```
{
  "query": {
    "pages": {
      "2": {
        "pageid": 2,
        "ns": 0,
        "title": "Test"
      }
    }
  }
}
```

Listing 3.11: Beispielaufruf */pagesByPrefix?search=T*

store Erwartet die POST-Variablen *name* für den Artikelnamen und *html* für die HTML Repräsentation des bearbeiteten Artikels, um diese wieder in das JSON Format zu überführen und lokal auf dem Server zu hinterlegen.

```
{
  "visualeditor": {
    "name": "Test",
    "result": "success"
  }
}
```

Listing 3.12: Beispielaufruf */store*

3.2.2 Services

MediaWiki Service Der MediaWiki Service kapselt alle für den Editor erforderlichen Methoden, für die ein Zugriff auf die MediaWiki API erforderlich ist.

- **getArticle** Liefert den Artikel mit dem übergebenen Namen zurück.
- **getImageDimensions** Liefert die Breite und Höhe des Bildes mit dem übergebenen Namen zurück.
- **getImageInfo** Liefert alle für das Bild mit dem übergebenen Namen vorhandenen Informationen zurück.

- **getImagesByPrefix** Liefert eine Liste aller Bilder zurück, die mit dem übergebenen Prefix beginnen.

- **getPagesByPrefix** Liefert eine Liste aller Artikel zurück, die mit dem übergebenen Prefix beginnen.

- **openSearch** Liefert eine Liste der Namen aller Artikel zurück, die mit dem übergebenen Prefix beginnen.

WOM3 Service Der WOM3 Service stellt eine Schnittstelle für den Zugriff auf die WOM3Tools und den Sweble Parser zur Verfügung.

- **parse** Ruft den Sweble Parser für den übergebenen Wikitext String auf und liefert das resultierende JSON Objekt zurück.

- **parse2HTML** Ruft den Sweble Parser für den übergebenen Wikitext String auf und liefert die resultierende HTML Repräsentation zurück.

- **parseDOM** Liefert die JSON Serialisierung des WOM Dokumentes für den vom Editor übergebenen HTML String zurück.

3.3 Client

Der Client besteht nur aus der von mir modifizierten VisualEditor Komponente, die den gesamten Editor darstellt.

3.3.1 VisualEditor

Da der VisualEditor abseits der Entwicklung durch die Wikimedia Foundation noch kaum Verbreitung hat, stehen auch keine anderen Dokumentationen und Code-Beispiele als die eigentliche Implementierung zur Verfügung. Aus diesem Grund habe ich die notwendigen Komponenten aus der, ebenfalls unter der MIT Lizenz veröffentlichten, MediaWiki VisualEditor Erweiterung[5] an meine Bedürfnisse angepasst und nicht von Grund auf neu geschrieben.

Für die Einbindung von Links werde ich nun exemplarisch die erforderlichen Teile erläutern. Der VisualEditor beinhaltet bereits generische Links, allerdings wird in der WOM3 Spezifikation zwischen internen und externen Links unterschieden. Dazu sollen die bestehenden Links erweitert werden.

[5] `http://www.mediawiki.org/wiki/Extension:VisualEditor`

DataModel Die *DataModel* Schicht enthält das lineare Modell des in Bearbeitung befindlichen Dokumentes[6], das aus dem eingelesenen HTML Dokument erzeugt wird. Im Anschluss an die Bearbeitung kann aus dem linearen Modell wieder ein HTML Dokument generiert werden. Um dies zu veranschaulichen, zeigt das folgende Beispiel den Aufbau des linearen Modells auf. Der folgende Ausschnitt eines HTML Dokumentes soll bearbeitet werden und wird dazu im VisualEditor geladen:

```
<p>
  Lorem <a href="http://example.com/">ipsum</a> dolor sit amet
</p>
```

Die *DataModel* Schicht erzeugt daraus die folgende Datenstruktur:

```
[
  { type: 'paragraph' },
  'L',
  'o',
  'r',
  'e',
  'm',
  ' ',
  [ 'i', {"type":"link","href":"http://example.com/"}],
  [ 'p', {"type":"link","href":"http://example.com/"}],
  [ 's', {"type":"link","href":"http://example.com/"}],
  [ 'u', {"type":"link","href":"http://example.com/"}],
  [ 'm', {"type":"link","href":"http://example.com/"}],
  ' ',
  ...
  { type: '/paragraph' }
]
```

Die Unterscheidung, ob ein HTML `a` Tag einen internen oder externen repräsentiert wird an anhand des `rel` Attributes getroffen, das mit den in den Elementen spezifizierten Typen übereinstimmen muss:

```
ve.dm
    .WOMInternalLinkAnnotation.static.name = 'link/womInternal';
ve.dm.WOMInternalLinkAnnotation
    .static.matchRdfaTypes = [ 'wom:IntLink' ];
```

Und im Vergleich dazu die Spezifikation für externe Links:

```
ve.dm
    .WOMExternalLinkAnnotation.static.name = 'link/womExternal';
ve.dm.WOMExternalLinkAnnotation
    .static.matchRdfaTypes = [ 'wom:ExtLink' ];
```

[6] http://www.mediawiki.org/wiki/VisualEditor/API/Data_Model/Surface

Die *toDataElement* Methode überführt das DOM Element in die interne Repräsentation für das lineare Modell:

```
ve.dm.LinkAnnotation
    .static.toDataElement = function ( domElements ) {
  return {
    'type': this.name,
    'attributes': {
      'href': domElements[0].getAttribute( 'href' )
    }
  };
};
```

Um im Anschluss an die Bearbeitung feststellen zu können, ob das Element manipuliert wurde, wird zusätzlich die DOM-Struktur zwischengespeichert:

```
ve.dm.WOMExternalLinkAnnotation
    .static.toDataElement = function(domElements) {
    var parentResult = ve.dm.LinkAnnotation
        .static.toDataElement.apply(this, arguments);

    parentResult
        .attributes.rel = domElements[0].getAttribute('rel');
    parentResult
        .attributes.originalDomElements = ve.copy(domElements);

    return parentResult;
};
```

Abschließend müssen die neuen Elemente noch als neue *DataModel* Objekte registriert werden, damit der VisualEditor beim Einlesen des zu bearbeitenden HTML Dokumentes von der Existenz der neuen Elemente weiß.

```
ve.dm.modelRegistry.register(ve.dm.WOMExternalLinkAnnotation);
```

ContentEditable In der ContentEditable Schicht wird aus dem linearen Modell wieder eine HTML Darstellung generiert.

Die Zuordnung zum passenden Element aus der DataModel Schicht erfolgt über den Vergleich des Klassenattributs *name*:

```
ve.ce.WOMExternalLinkAnnotation = function
     VeCeWOMExternalLinkAnnotation(model, parentNode, config) {
    ve.ce.LinkAnnotation.call(this, model, parentNode, config);

    this.$element.addClass('ve-ce-womExternalLinkAnnotation');
    this.$element.attr('title', model.getAttribute('href'));
};
...
ve.ce
    .WOMExternalLinkAnnotation.static.name = 'link/womExternal';
```

UserInterface Nun fehlt noch eine Möglichkeit zur Bearbeitung des Verweises. Dazu stehen im VisualEditor Inspectors zur Verfügung, die eine einfache Alternative zu den komplexen Dialogfenstern darstellen.

Auch hier muss festgelegt werden, für welche Elemente dieser Inspector aktiviert werden soll. Für die Bearbeitung von Link-Elementen, die beiden zuvor angelegten Annotationen:

```
ve.ui.WOMLinkInspector.static.name = 'link';

ve.ui.WOMLinkInspector.static.modelClasses = [
    ve.dm.WOMExternalLinkAnnotation,
    ve.dm.WOMInternalLinkAnnotation,
    ...
];

ve.ui.WOMLinkInspector.static
   .linkTargetInputWidget = ve.ui.WOMLinkTargetInputWidget;
```

Über das Klassenattribut *linkTargetInputWidget* wird festgelegt welches Widget zur Ausführung kommen soll, wenn der Inspector aktiviert wird.

Das hier definierte Widget schickt Anfragen an die, durch die zuvor vorgestellten Controller bereitgestellte API, um eine Liste der Artikel zu erhalten, die mit dem durch den Benutzer eingegebenen Titel beginnen. Wird bei der Eingabe eine gültige URL erkannt, so wird automatisch vorgeschlagen, ob ein externer Link erzeugt werden soll:

```
propsJqXhr = ve.init.wom3.Target.static.apiRequest({
  'action': 'pagesByPrefix',
    'search': title
});
```

Anhang A Einrichtung

Im nun folgenden Abschnitt erläutere ich die zum Einsatz der Software erforderlichen Schritte. Zu Demonstrationszwecken steht in Anhang B mit Docker auch eine einfachere Lösung zur Verfügung.

A.1 Client

Alle im Folgenden auszuführenden Befehle beziehen sich relativ zum Wurzelverzeichnis, das die Client-Dateien enthält.

API Dokumentation

Zur Generierung der API Dokumentation setzt der VisualEditor auf *JSDuck*[7].

Durch den folgenden Aufruf wird die HTML Version der API Dokumentation im Ordner *./docs* erstellt.

```
1  ./bin/generateDocs.sh
```

Grunt

Um die zahlreichen JavaScript Dateien des VisualEditors in einer einzelnen zusammen zu fassen, habe ich in *Grunt*[8] einen *UglifyJS*[9] Task angelegt, der im Order *target* die Datei *wom3editor.js* generiert, in der alle Quelldateien zusammen gefasst werden. Dies reduziert die Anzahl der durch den Browser anzufragenden Dateien und wirkt sich damit auch positiv auf die initiale Ladezeit aus.

```
1  npm install
2  grunt
```

A.2 WOM3Tools

Alle im Folgenden auszuführenden Befehle beziehen sich relativ zum Wurzelverzeichnis, das die WOM3Tools-Dateien enthält.

API Dokumentation

Zur Generierung der API Dokumentation setzt die *Closure Library* auf das *jsdoc-toolkit*[10].

[7] https://github.com/senchalabs/jsduck
[8] http://gruntjs.com/
[9] https://github.com/gruntjs/grunt-contrib-uglify
[10] https://code.google.com/p/jsdoc-toolkit/

Durch den folgenden Aufruf wird die HTML Version der API Dokumentation im Ordner *./docs* erstellt.

```
./bin/generateDocs.sh
```

Kompilierung

Der folgende Aufruf ruft den Closure Compiler zur Kompilierung der WOM3Tools Quelldateien in eine einzelne JavaScript Datei auf.

```
./bin/compile.sh
```

A.3 Server

Alle im Folgenden auszuführenden Befehle beziehen sich relativ zum Wurzelverzeichnis, das die Server-Dateien enthält.

node.js Server

Um die für den Betrieb erforderlichen Abhängigkeiten zu installieren, muss der folgende Befehl ausgeführt werden, der die benötigten Bibliotheken in der richtigen Version aus dem Internet herunterlädt.

```
npm install
```

Zum Starten des eigentlichen Servers, muss anschließend der folgende Befehl ausgeführt werden.

```
node app
```

Anhang B Docker

Mit dem noch jungen Projekt Docker[11] steht eine leichtgewichtige Virtualisierungslösung zur Virtualisierung von Anwendungscontainer bereit. Damit kann ein einzelner Container auf beliebiger, mit dem Linux Betriebssystem bestückten, Hardware laufen, sei es ein Laptop, auf einem dedizierten Server oder direkt in der Cloud.

Zur vereinfachten Demonstration des Editors stelle ich hier ein Dockerfile zur Erstellung eines Containers bereit, der alle erforderlichen Komponenten beinhaltet, die für einen Betrieb benötigt werden. Dies beinhaltet eine MediaWiki Instanz samt zugehöriger Datenbank und Webserver. Das MediaWiki wird hier zur Bereitstellung des im Editor zu bearbeitenden Dokumentes verwendet, sodass Änderungen im Wikitext Dokument, direkt im Editor angezeigt werden können und auch Bilddaten und Linkziele im Editor zur Verfügung stehen.

B.1 Dockerfile

```
1  FROM debian:wheezy
2  MAINTAINER Michael Haase <michael.haase@gernox.de>
3
4  ENV DEBIAN_FRONTEND noninteractive
5
6  # Configure language
7  RUN apt-get update
8  RUN echo 'Acquire
       ::Languages {"none";};' > /etc/apt/apt.conf.d/60language
9  RUN apt-get -y install locales apt-utils
10 RUN echo "de_DE.UTF-8 UTF-8" > /etc/locale.gen
11 RUN locale-gen
12 ENV LANGUAGE de_DE.UTF-8
13 ENV LANG de_DE.UTF-8
14 ENV LC_ALL de_DE.UTF-8
15
16 # Install tools
17 RUN apt-get update
18 RUN apt-get -y install apt-utils wget dialog net-tools curl
19
20 # Install MariaDB
21 RUN apt
       -get -y install python-software-properties pwgen inotify-tools
22 RUN apt-key adv --
       recv-keys --keyserver keyserver.ubuntu.com 0xcbcb082a1bb943db
23 RUN add-apt-repository 'deb http://
       mirror2.hs-esslingen.de/mariadb/repo/10.0/debian wheezy main'
24 RUN apt-get update
25 RUN apt-get -y install mariadb-server
26
27 # Configure MariaDB
28 RUN sed -i 's
       /^innodb_flush_method/#innodb_flush_method/' /etc/mysql/my.cnf
```

[11] https://www.docker.io/

```
29
30 # Install Nginx from repository
31 RUN wget -O - "http://www.dotdeb.org/dotdeb.gpg" | apt-key add -
32 RUN echo "deb http
      ://packages.dotdeb.org wheezy all \ndeb-src http://packages
      .dotdeb.org wheezy all" > /etc/apt/sources.list.d/dotdeb.list
33 RUN apt-get -y install nginx
34
35 # Install PHP5
36 RUN apt-get -y install php5-fpm php5
      -mysqlnd php-apc php5-imagick php5-imap php5-mcrypt php5-cli
37
38 # Configure Nginx and PHP5
39 RUN sed -i 's
      /^;daemonize = yes/daemonize = no/' /etc/php5/fpm/php-fpm.conf
40 RUN echo "cgi.fix_pathinfo = 0;" >> /etc/php5/fpm/php.ini
41 RUN echo "daemon off;" >> /etc/nginx/nginx.conf
42 RUN mkdir /var/www && chown -R www-data:www-data /var/www
43 ADD nginx_default /etc/nginx/sites-available/default
44
45 # Install supervisord
46 RUN apt-get -y install python-setuptools
47 RUN easy_install supervisor
48
49 # Configure supervisord
50 ADD supervisord.conf /etc/supervisord.conf
51
52 # Configure Mediawiki
53 ADD mediawiki /var/www/wiki
54 ADD content /content
55
56 # Install WOM Editor
57 ADD editor /var/www/editor
58
59 ADD start.sh /start.sh
60 RUN chmod +x /start.sh
61
62 # Install node.js
63 RUN echo "deb http://ftp.us.debian
      .org/debian wheezy-backports main" >> /etc/apt/sources.list
64 RUN apt-get update && apt-get -y install nodejs-legacy
65
66 EXPOSE 80
67
68 # Start
69 ENTRYPOINT ["/start.sh"]
```

Listing 3.13: Dockerfile

B.2 Installation

Ich gehe hier von der Annahme aus, dass Docker bereits auf dem System installiert wurde und der Benutzer die zur Ausführung erforderlichen Rechte besitzt.

Anderenfalls stehen unter folgender URL weitere Informationen bereit: `https://www.docker.io/gettingstarted/#h_installation`.

Container erstellen

Zur Ausführung muss aus dem Dockerfile zunächst ein Container generiert werden. Dazu muss in das Verzeichnis gewechselt werden, in dem sich das Dockerfile und die zusätzlich erforderlichen Dateien befinden. Anschließend kann der Container mit folgendem Befehl gebaut werden:

```
1  docker build -t wom3/editor .
```

Listing 3.14: Container erstellen

Ausführung

Um den eben erstellten Container zur Ausführung zu bringen genügt folgender Befehl

```
1  docker run -d -p 8080:80 wom3/editor
```

Listing 3.15: Container ausführen

Damit läuft der Container nun im Hintergrund und leitet den internen Port 80, an dem der integrierte Webserver auf Anfragen wartet, auf den Port 8080 des Hostsystems weiter.

Sobald die Installation der als Datenquelle dienenden MediaWiki Instanz abgeschlossen ist, steht der Editor unter `http://127.0.0.1:8080/editor/public` zur Verfügung stehen.

Es besteht die Möglichkeit, das dem im Editor zur Bearbeitung zugrunde liegende Wikitext-Dokument, direkt in der MediaWiki Instanz zu bearbeiten und zum Beispiel Bilder hochzuladen oder weitere Seiten als Linkziele zu erstellen.

Das Wiki ist unter der folgenden URL erreichbar: `http://127.0.0.1:8080/wiki/index.php?title=Test`

Literaturverzeichnis

Berjon, Robin et al. (2014). *HTML5* (Empfehlung). W3C. (http://www.w3.org/TR/2014/CR-html5-20140429/. Neueste Version verfügbar unter http://www.w3.org/TR/html5/)

Berners-Lee, T. & Connolly, D. (1995, November). *Hypertext Markup Language - 2.0* (Nr. 1866). RFC 1866 (Historic). IETF. Zugriff auf http://www.ietf.org/rfc/rfc1866.txt (Obsoleted by RFC 2854)

Cunningham, W. (2002, Juni). *What is Wiki.* Zugriff am 2014-05-05 auf http://www.wiki.org/wiki.cgi?WhatIsWiki

Dijkstra, E. W. (1982). On the role of scientific thought. In *Selected writings on computing: A personal perspective* (S. 60–66). Springer-Verlag.

Dohrn, H. & Riehle, D. (2011). Design and implementation of the sweble wikitext parser: Unlocking the structured data of wikipedia. In *Proceedings of the 7th international symposium on wikis and open collaboration* (S. 72–81). New York, NY, USA: ACM. Zugriff auf http://doi.acm.org/10.1145/2038558.2038571 doi: 10.1145/2038558.2038571

Dohrn, H. & Riehle, D. (2013). Design and implementation of wiki content transformations and refactorings. In *Proceedings of the 9th international symposium on open collaboration* (S. 2:1–2:10). New York, NY, USA: ACM. Zugriff auf http://doi.acm.org/10.1145/2491055.2491057 doi: 10.1145/2491055.2491057

ECMA International. (1999). *Standard ECMA-262*. Zugriff auf http://www.ecma-international.org/publications/standards/Ecma-262.htm

Fowler, M. (2003). *Patterns of enterprise application architecture*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.

Gamma, E., Helm, R., Johnson, R. & Vlissides, J. (1995). *Design patterns*. Reading, MA: Addison Wesley.

Kay, M. (2007). *XSL Transformations (XSLT) Version 2.0* (Empfehlung). W3C. (http://www.w3.org/TR/2007/REC-xslt20-20070123/. Neueste Version verfügbar unter http://www.w3.org/TR/xslt20/)

Lamport, L. (1988, April). Document Production: Visual or Logical? *TUGboat*, *9* (1), 8–10.

Le Hors, Arnaud et al. (2014). *Document Object Model (DOM) Level 3 Core Specification* (Empfehlung). W3C. (http://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407/. Neueste Version verfügbar unter http://www.w3.org/TR/DOM-Level-3-Core/)

Mell, P. & Grance, T. (2011, September). *The nist definition of cloud computing* (Bericht Nr. 800-145). Gaithersburg, MD: National Institute of Standards and Technology (NIST). Zugriff auf http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf

Posma, J. P. (2011, Juli). *In-line editing: a new approach to editing wikis.* (Bachelor thesis)

Sauer, C. (2006). What you see is wiki – questioning WYSIWYG in the Internet age. In *Proceedings of wikimania 2006.* Zugriff auf http://wikimania2006.wikimedia.org/wiki/Proceedings:CS1

Simonite, T. (2013, Oktober). *The decline of wikipedia.* MIT Technology Review. (Abgerufen von http://www.technologyreview.com/featuredstory/520446/the-decline-of-wikipedia/)

van Kesteren, A., Aubourg, J., Song, J. & Steen, H. R. (2014). *XMLHttpRequest Level 1* (Arbeitsentwurf). W3C. (http://www.w3.org/TR/2014/WD-XMLHttpRequest-20140130/. Neueste Version verfügbar unter http://www.w3.org/TR/XMLHttpRequest/)

Vora, P., Komura, N. & Team, S. U. (2010). The n00b wikipedia editing experience. In *Proceedings of the 6th international symposium on wikis and open collaboration* (S. 36:1–36:3). New York, NY, USA: ACM. Zugriff auf http://doi.acm.org/10.1145/1832772.1841393 doi: 10.1145/1832772.1841393

Wikimedia Foundation. (2011, April). *Wikipedia editors study* (Bericht). Zugriff auf http://upload.wikimedia.org/wikipedia/commons/7/76/Editor_Survey_Report_-_April_2011.pdf