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

OLIVER LUTZ
BACHELOR THESIS

TREE-BASED, SEMANTIC ARTICLE DIFFERENCE VISUALIZATION

Eingereicht am 18. April 2017

Betreuer: Dipl.-Inf. Hannes Dohrn
Supervisor: 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, 18. April 2017

License

Erlangen, 18. April 2017

Abstract

The platform, *Sweble Hub* allows to collaboratively edit Wiki-content using the Wiki Object Model. This is, however, not carried out on wikitext level as is normally the case, but rather by using the principle, 'what you see is what you get'.

The objective is to extend the existing project with a clearly displayed article difference visualization which provides a comprehensible overview of modifications made.

This bachelor thesis offers a solution as to how wikitext-file versions can be clearly compared. Based on tree-based algorithms and data structures, it allows to interpret the semantics of wikitext documents and any changes made within these documents. As a result it provides a more detailed depiction in comparison to text-based Diffs. The outcome is embedded into the Sweble Hub application.

Zusammenfassung

Die Plattform *Sweble Hub* ermöglicht es, Wiki-Inhalte auf Basis des *Wiki Object Model*s kollaborativ zu bearbeiten. Dies findet dadurch nicht, wie normalerweise üblich, auf Wikitext-Ebene statt, sondern nach dem Prinzip 'what you see is what you get'.

Ziel ist es, das existierende Projekt um eine übersichtliche Artikel Diff Visualisierung zu erweitern, die es ermöglicht, einen verständlichen Überblick über getätigte Änderungen zu erhalten.

Im Rahmen dieser Bachelorarbeit wird eine Lösung vorgestellt, wie man Wikitext-Datei-Versionen anschaulich vergleichen kann. Die Grundlage hierfür sind Baum-basierte Algorithmen und Datenstrukturen, die Wikitext-Dokumente und erfolgte Modifikationen semantisch interpretieren können. Dies erzeugt im Vergleich zu Text-basierten Diffs eine detailliertere Darstellung. Die erarbeitete Ausarbeitung soll in das *Sweble Hub* eingebettet werden.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einführung

Die Idee von Wikipedia ist „eine frei lizenzierte und hochwertige Enzyklopädie zu schaffen und damit lexikalisches Wissen zu verbreiten“[1]. Artikel werden durch *kollaboratives Schreiben* von freiwilligen Autoren hinzugefügt, bearbeitet und diskutiert. Hierbei handelt es sich um Autoren mit verschiedener Herkunft, Perspektive und Meinung. Deswegen ist es auch die Aufgabe der Software, dafür zu sorgen, dass bei der gemeinsamen Bearbeitung von Artikeln das bestmögliche Ergebnis einfach erarbeitet werden kann.

Dies betrifft auch die Versionskontrolle von Wikipedia-Artikeln, damit Änderungen von Mitautoren eindeutig und klar nachzuvollziehen sind. Vor allem ist es wichtig, möglichst einfach die Absicht des Mitautors zu verstehen oder potentiellem Spam schnell aus dem Weg zu gehen. Um verschiedene Wikipedia Artikel-Versionen miteinander zu vergleichen, verwendet *Wikipedia* zum jetzigen Zeitpunkt eine *Textbasierte Diff Visualisierung*. Hierbei werden die zwei Wikipedia Artikel-Versionen als Wikitext dargestellt und veränderte Wikitext-Zeilen farblich markiert.

Das Projekt *Sweble Hub*[2] hat sich zum Ziel gesetzt, Benutzern im Allgemeinen das Arbeiten mit Wikipedia zu vereinfachen. Dabei setzt die *Open Source Research Group*[3] auf das Prinzip *'what you see is what you get'*, um nicht nur den allgemeinen Umgang zu vereinfachen, sondern um auch Neueinsteigern einen leichten Start zu ermöglichen. Um dies zu bewerkstelligen, wurde der *Sweble Parser*[4] entwickelt, der aus reinem *Wikitext* ein *Wiki Object Model* (WOM) bildet. Dabei wird der *Wikitext* in einen *abstrakten Syntaxbaum* (AST) umgewandelt, dessen Knoten Informationen über die Semantik des Wikitextes enthält und syntaktischen *Markup* vom eigentlichen Text trennt. Zusätzlich wurde mit dem *HD-Diff Algorithmus*[5] eine Methode entwickelt, *Wiki Object Model*s miteinander zu vergleichen.

[1] “Interview mit Jimmy Wales: Wie geht es weiter mit Wikipedia?”, 2011
[2] http://sweble.org/
[3] https://osr.cs.fau.de/
[4] http://sweble.org/category/wikitext-parser/
[5] http://sweble.org/projects/hddiff/

Im Verlauf dieser Arbeit wird eine Visualisierung vorgestellt, die auf dem *Wiki Object Model* sowie dem *HD-Diff Algorithmus* basieren. Ziel ist es, Veränderungen von Wikipedia-Artikeln für Benutzer leicht einsehbar zu gestalten.

Im Folgenden werden die Rahmenbedingungen der entwickelten Arbeit genauer vorgestellt.

1.1 Ziele

Das Hauptziel dieser Arbeit ist es, eine verständliche Ansicht zweier Wikitext-Versionen zu realisieren, um dem Benutzer beim kollaborativen Arbeiten einen übersichtlichen Vergleich zu ermöglichen.

Grundlage hierbei sind auf Bäumen basierte Algorithmen und Datenstrukturen, die in Kapitel 2 genauer erläutert werden. Durch diese Gegebenheiten basieren Artikel-Diffs nicht nur auf dem reinen textuellen Unterschied, sondern veranschaulichen Änderungen je nach semantischem Kontext.

1.2 Anforderungen

Das Frontend für die beschriebene Software soll mit Hilfe des *React Frameworks*[6] entwickelt und nach Vollendung in das Projekt *Sweble Hub* eingebettet werden. Hierbei sollen folgende Anforderungen erfüllt werden, auf denen die Evaluation in Kapitel 5 basiert.

Wiki Object Model

Die Visualisierung von Wikitext soll in dieser Arbeit durch Verwendung des *Wiki Object Model*s umgesetzt werden. Dieses soll durch ein existentes Backend mit dem eingebettetem *Sweble Parser* angefordert werden.

Tree-based edit script

Des Weiteren sollen Änderungen von Wikitext-Versionen mit Hilfe des *edit script*s, welches das Backend zur Verfügung stellt, durchgeführt und auf dessen Basis vi-

[6] https://facebook.github.io/react/

suell hervorgehoben werden.

Zwei-Spalten-Ansicht

Ergebnis dieser Arbeit soll eine Zwei-Spalten-Ansicht sein, die in einem gängigen Browser angezeigt werden kann. Diese beinhaltet die beiden Artikel-Versionen in einer gegenüberstellenden Ansicht.

Visualisierung von Änderungen

Hierbei sollen die Änderungen je nach Art unterschiedlich markiert werden.

Der weitere Verlauf dieser Arbeit ist wie folgt strukturiert:

- In Kapitel 2 werden die Grundlagen erläutert, auf denen die Arbeit basiert. Im Zuge dessen werden die bereits genannte Datenstruktur *Wiki Object Model* und der HD-Diff Algorithmus inklusive des daraus resultierenden *edit script*s erklärt.
- Kapitel 3 enthält die Architektur und das Design der Visualisierungssoftware, dessen Implementierungsdetails im darauf folgenden Kapitel 4 dargestellt werden.
- Abschließend werden die soeben vorgestellten Anforderungen in Kapitel 5 evaluiert.

2 Grundlagen

Um den Wikitext zu einer formatierten HTML-Seite zu transformieren, verwendet die *MediaWiki* Software keine leicht einsehbare Repräsentation wie beispielsweise einen *abstrakten Syntaxbaum*. Im Folgenden werden die in dieser Arbeit verwendeten Datenstrukturen und die Grundlagen erläutert, wie Wikitext und deren Versionsvergleiche geschickt in eine Datenstruktur gespeichert werden. Dies ist die Grundlage, um die *Frontend App* benutzerdefiniert gestalten zu können.

2.1 Wikitext

Wikitext ist eine *Markup*-Sprache (Auszeichnungssprache), die verwendet wird, um Beiträge in Wikis zu erstellen. Es soll Benutzern den Umgang mit der Software erleichtern, die keine HTML Kenntnisse besitzen. Die Formatierungen erfolgen hierbei über spezielle vorgegebene Zeichenkombinationen. So wird beispielsweise aus der Eingabe "kursiver Text" ein *kursiver Text*, wobei dies durch ein doppeltes Hochkomma angezeigt wird. Dies ist nur ein kleines Beispiel von vielen, wo ein Text einer Formatvorlage zugewiesen wird, um somit den Wiki-Artikel gut gegliedert und lesbar zu gestalten. Mittels der *MediaWiki* Software wird aus dem Wikitext der HTML-Code generiert, um somit Inhalte im Browser anzeigen zu können.

2.2 Wiki Object Model (WOM)

Da das Arbeiten mit Wikitext für den Entwickler nicht optimal ist, wurde das Datenformat *Wiki Object Model* (WOM) kreiert, um Informationen zur Semantik von Wikitext zu verwalten und den Textinhalt von syntaktischem *Markup* zu trennen. Bei dieser Datenstruktur handelt es sich um einen abstrakten Syntaxbaum (AST). Hiermit wird die Lücke zwischen dem reinem Wikitext und reinem HTML-Code für Entwickler geschlossen.

Dabei wird mittels *Sweble Parser* aus dem Wikitext ein *Wiki Object Model* gebildet. Dieser Parser bereinigt bereits syntaktische Fehler im Wikitext und baut einen *abstrakten Syntaxbaum*, dessen Knoten Informationen über die Semantik und Syntax des Wikitextabschnittes beinhalten. Dabei gilt, dass der Wikitext äquivalent zum WOM ist und dementsprechend in beide Richtungen transformierbar ist.

Ein WOM-Baum besteht aus Text- und Elementknoten. In den Textknoten befindet sich der eigentliche Text, während Elementknoten die Formatierung und Semantik der Kinderknoten beschreibt. Zusätzlich werden die syntaktischen *Markup*-Zeichenketten in sogenannten *Round-Trip-Data* (RTD) tags separat vom eigentlichen Text gespeichert. So wird ein fett angezeigtes Wort wie in Abbildung 2.1 repräsentiert.

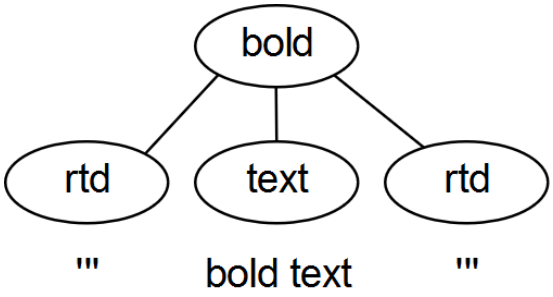


Abbildung 2.1: WOM-Knoten mit bold markiertem Text

Die Anzahl der Kinderknoten ist dabei allerdings nicht festgelegt und kann je nach Semantik variieren. Elementknoten können eine bestimmte Menge an Eigenschaften und Attribute enthalten, die weitere Informationen über den Wikitext der Kinderknoten zur Verfügung stellen.

In Abbildung 2.2 wird ein einfacher Artikel als WOM transformiert in seinem schematischen Aufbau dargestellt.

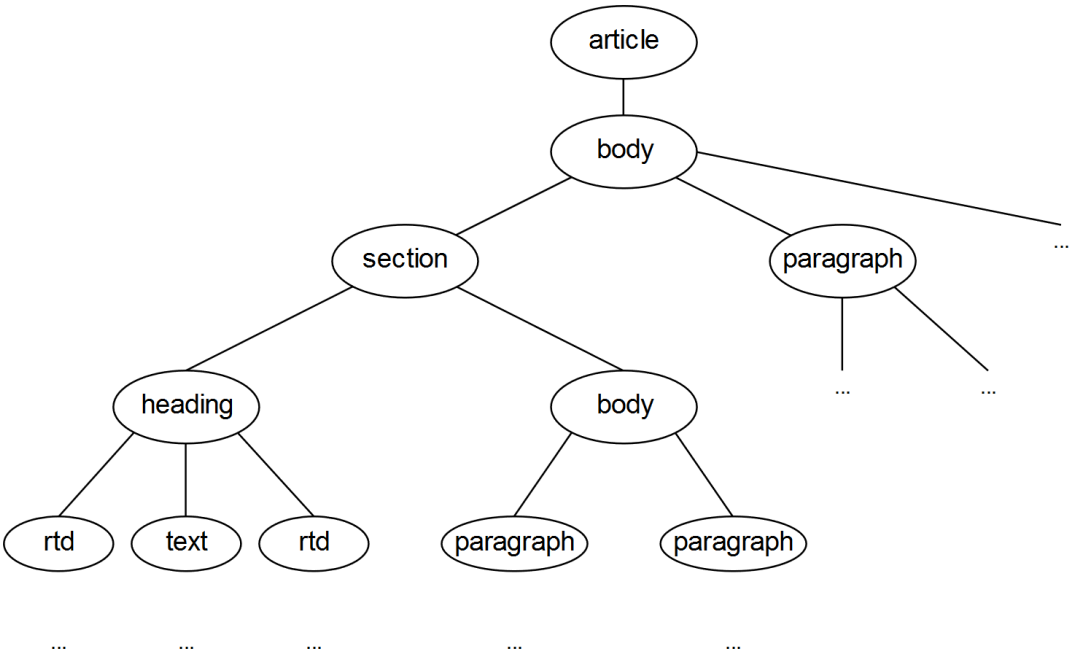


Abbildung 2.2: Schematischer Aufbau eines WOM-Artikels

Ein Wiki-Artikel als WOM-Baum enthält immer einen *article*-Wurzelknoten. Im darauf folgenden *body* sind typischerweise *paragraph*s oder *section*s zu finden. Ein *paragraph* kann variabel viele Kinderknoten enthalten, eine *section* wird durch den *head* und *body* definiert. Dadurch wird allerdings nur eine kleine Auswahl von vielen möglichen Knotentypen dargestellt.

Ein Wikipedia-Artikel setzt sich wieder zusammen, indem man von oben nach unten den Baum entlang die Metadaten interpretiert. Somit wird ein Artikel zusammen gestellt, der vom Inhalt her äquivalent zum Original ist. Um den reinen Wikitext wiederherzustellen, kann man auch alle Blattknoten aneinander reihen, da die RTD- bzw. Textknoten keine Kinderknoten besitzen.

2.3 HD-Diff Algorithmus

Um neben den Wikipedia-Artikeln auch dem Vergleich von verschiedenen Artikel-Versionen mehr Semantik zu verleihen, wurde der HD-Diff, ein Baum-basierter Algorithmus, entwickelt. Dieser erhält als Eingabe keine reine Buchstabensequenz, sondern die zu vergleichenden WOM-Dokument-Versionen und kann aufgrund des WOM-Aufbaus zwischen syntaktischem *Markup* und reinem Text unterscheiden. Das Resultat nach Anwendung des Algorithmus ist ein *edit script*, das ebenfalls baumartig aufgebaut ist (Kapitel 2.3.2) und dem hervorgeht, wie man mittels de-

finierter Operationen (Kapitel 2.3.1) aus dem ursprünglichen WOM-Dokument die neuere Version bilden kann.

2.3.1 Operationen des HD-Diff Algorithmus

Der *HD-Diff* verfügt neben den Operationen DELETE und INSERT die Methoden MOVE, SPLIT und UPDATE, wodurch aus einem ursprünglichen WOM-Baum die neue Version des WOM-Dokuments gebildet werden kann. Anhand zweier Bäume werden diese Operationen nun im Einzelnen vorgestellt und in Abbildung 2.3 visualisiert. Hierbei wird im Folgenden vom linken (= ursprünglicher Baum) und rechten Baum (= zu bearbeitendem Baum) die Rede sein:

- **DELETE**: Mittels der DELETE-Funktion wird angezeigt, dass ein Knoten im ursprünglichen Baum an einer spezifizierten Stelle gelöscht und somit Wikitext entfernt wurde.
- **INSERT**: Wurde Wikitext hinzugefügt, wird mit Hilfe der INSERT-Operation dem rechten Baum an der vorgegebenen Stelle ein Knoten angefügt.
- **MOVE**: Wenn Teile des Wikitextes verschoben wurden, wird der entsprechende Knoten im linken Baum entfernt und dem rechten Baum an der gewünschten Stelle platziert. Um dies technisch umzusetzen, wird der Befehl in MOVE_FROM und MOVE_TO aufgeteilt. Diese Befehle enthalten Informationen über die entsprechenden Positionen, an die der Knoten zu verschieben ist. Über eine *id* wird das Operationen-Paar MOVE_FROM und MOVE_TO eindeutig identifiziert. Dies ersetzt die aus den meisten Textbasierten Algorithmen verwendete Kombination aus DELETE und INSERT des gleichen Wikitext-Abschnittes. Mittels dieser Methode ist es möglich, dem Nutzer zu signalisieren, welche Textabschnitte wirklich gelöscht und welche nur verschoben wurden und hilft somit, die Veränderungen besser zu verstehen.
- **SPLIT**: Da dieser Algorithmus unformatierten Text als Textknoten und nicht als feingranulare Buchstabenfolge erfasst, wird die Funktion SPLIT verwendet, um Textknoten zu spalten. Dies ist zum Beispiel nötig, wenn in einem existierenden Textknoten aus drei Wörtern das mittlere Wort zu einem bold (fett markierten) Element wird. Dabei wird der Textknoten in drei Textknoten geteilt: einer für das veränderte Wort und jeweils einen für den Text vor dem Wort und nach dem Wort. Nun wird ein leerer Bold-Knoten eingefügt und das zu verändernde Wort mit der MOVE-Operation in diesen Knoten eingebettet. Nach außen hin wird das SPLIT nicht visualisiert und dient nur der technischen Umsetzung, denn der Text hat sich inhaltlich nicht verändert. Nur an dem einzeln veränderten Wort wird wegen der MOVE-Operation eine Modifikation angezeigt, welche die getätigte

Änderung ausreichend veranschaulicht. Die genaue Umsetzung ist den Abbildungen 2.5 und 2.6, zu entnehmen.

- **UPDATE**: Mittels dieser letzten Funktion werden Änderungen an einzelnen Knoten angezeigt, solange sich die Position des Knotens an sich nicht ändert. UPDATE umfasst sowohl Wikitext an sich als auch Attribute von Knoten. Diese Methode umfasst hierbei drei mögliche Modifikationen an Attributen, und zwar DELETE, INSERT und UPDATE. Auch diese Operation verfügt ähnlich wie die MOVE-Funktion über den Vorteil, dass sie im Gegensatz zu textuellen Algorithmen eine sinnvolle Bedeutung gibt, die nicht dem einfachen Löschen und Hinzufügen von Wörtern entspricht.

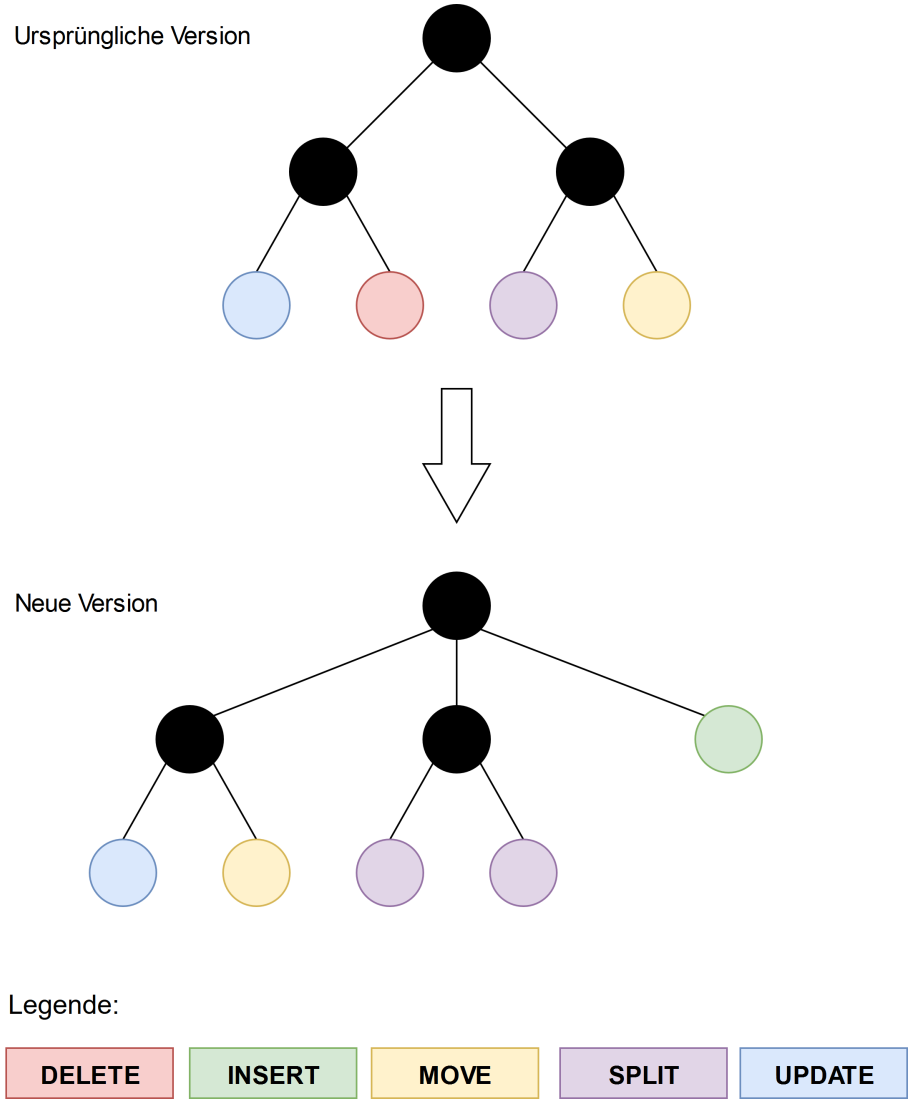


Abbildung 2.3: Beispiel für Operationen des HD-Diffs

2.3.2 Baum-basiertes edit script

Als Ausgabe des HD-Diff Algorithmus erhält man ein *edit script*, das eine Beschreibung beinhaltet, wie man aus der ursprünglichen Artikel-Version die neue Variante erhält. Dabei hat das *edit script* analog dem WOM-Domkument ebenfalls die Struktur eines abstrakten Syntaxbaums. Die Idee dahinter ist, während des Traversierens des WOM-Baums parallel das *edit script* zu durchlaufen und dadurch in der passenden Tiefe an der richtigen Position die aus Kapitel 2.3.1 bekannten Operationen auf den ursprünglichen Baum anzuwenden.

Dabei enthält jeder Knoten des *edit script*s die Information darüber, welche Operation angewendet wurde. Je nachdem, welche Änderung getätigt wurde, beinhalten die Knoten weitere Metadaten, die im Folgenden aufgezählt werden.

Ist an einer tiefer liegenden Stelle im Baum eine Änderung durchgeführt worden, so wird dies durch die Operation NODE angezeigt. Handelt es sich hierbei nicht um den Wurzelknoten, wird die Position des WOM-Kinderbaums angegeben, in dem Änderungen stattgefunden haben. Zuletzt werden die tiefer liegenden Abwandlungen in dem *edit script*-Kinderbaum spezifiziert.

Wenn die zwei Artikel-Versionen keine Änderungen beinhalten, so besteht der Baum aus einen einzigen Knoten mit der Operation NODE und keinen weiteren Attributen.

Die verschiedenen Operationen aus Kapitel 2.3.1 enthalten weitere zusätzliche Informationen, um die Änderungen genau und eindeutig, zu spezifizieren, insbesondere Angaben zur Position. Diese Positionsangabe bezieht sich immer auf den aktuellen Abarbeitungsstand und nicht auf die Ursprungsstelle im WOM-Baum. Wenn ein Textknoten beispielsweise zunächst geteilt oder ein Knoten vor einem anderen veränderten Knoten eingefügt wird, so verschiebt sich seine Position nach hinten und entspricht nicht mehr der Ursprungsposition. Weitere Spezifikationen werden im Folgenden genauer beschrieben:

- Die INSERT-Funktion enthält zusätzlich den vollständigen einzufügenden Knoten im Attribut *tree*.
- Ein UPDATE-Knoten kann neben dem neuen Wert, der im Attribut *newValue* festgelegt ist, auch Attribute, die einen WOM-Knoten betreffen, verändern. Dies ist im Attribut *attributeChanges* definiert. Hierbei wird mit einem '*'-Symbol vor der Attributbezeichnung signalisiert, dass ein Attributwert geändert wurde, während ein hinzugefügtes Attribut mit einem '+'-Symbol und ein zu entfernendes Attribut mit einem '-'-Symbol gekennzeichnet ist.
- Um einen Knoten-SPLIT auszuführen, bedarf es der genauen Position (*splitPos*), an der das im Knoten enthaltene Wort getrennt werden soll.

- Für den MOVE-Befehl ist eine eindeutige *id* nötig, um das Operationspaar MOVE_FROM und MOVE_TO eindeutig zu identifizieren. Somit bleibt bei der WOM-Baumumwandlung jede MOVE-Funktion eindeutig.
- Die DELETE-Operation benötigt neben der Position keine weiteren Informationen.

Die folgende Abbildung 2.4 veranschaulicht die zuvor beschriebenen Knoten, die in einem *edit script* auftauchen können.

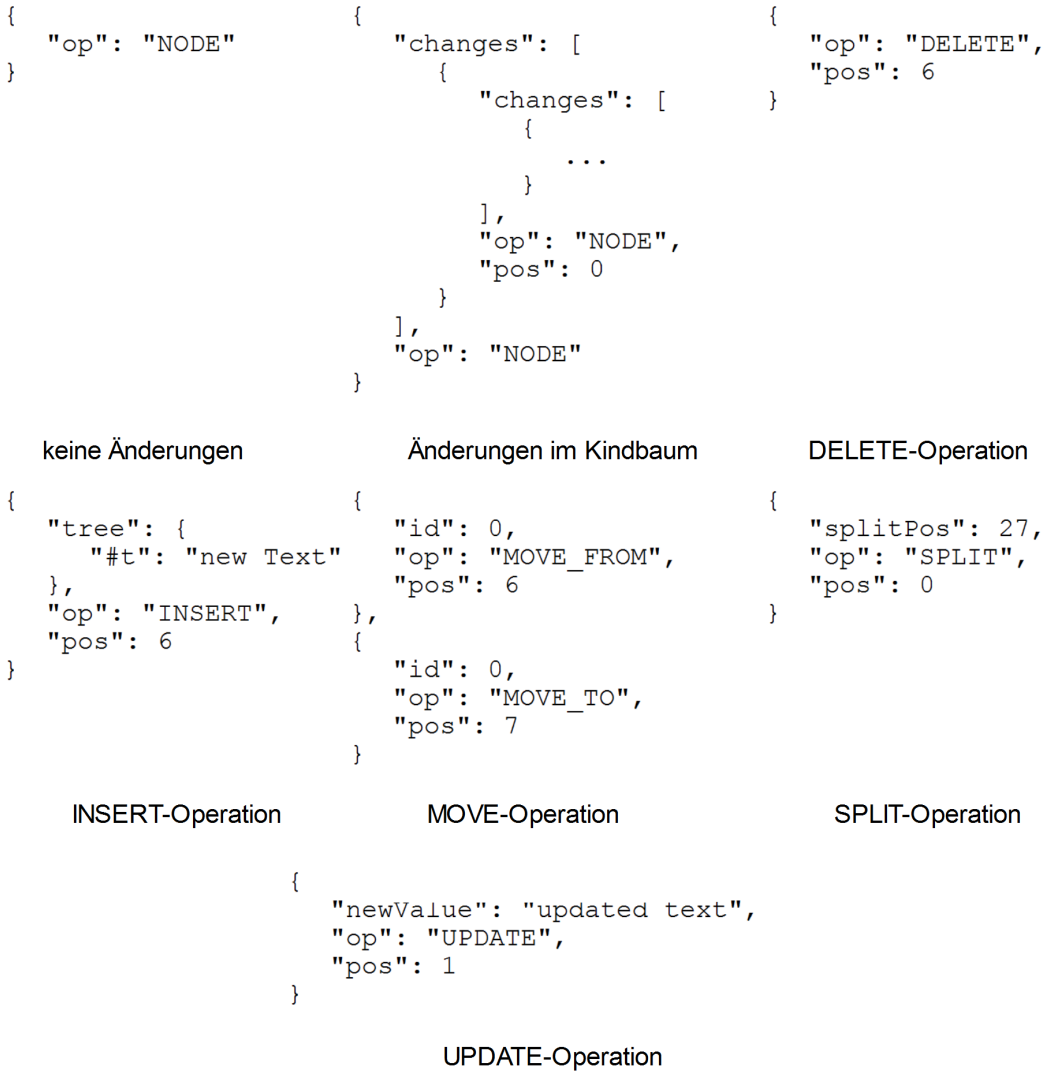


Abbildung 2.4: Beispielknoten aller Operationen des *edit scripts* als *JSON*

Die Baum-Transformationen für ein komplettes *edit script* wird anhand eines Beispiels in den Abbildungen 2.5 und 2.6 vorgestellt. Bei dem Beispiel handelt es sich um einen Satz, in dem ein Wort fett markiert wird. Der Einfachheit halber ist der Wurzelknoten ein *paragraph* und kein *article* Knoten.

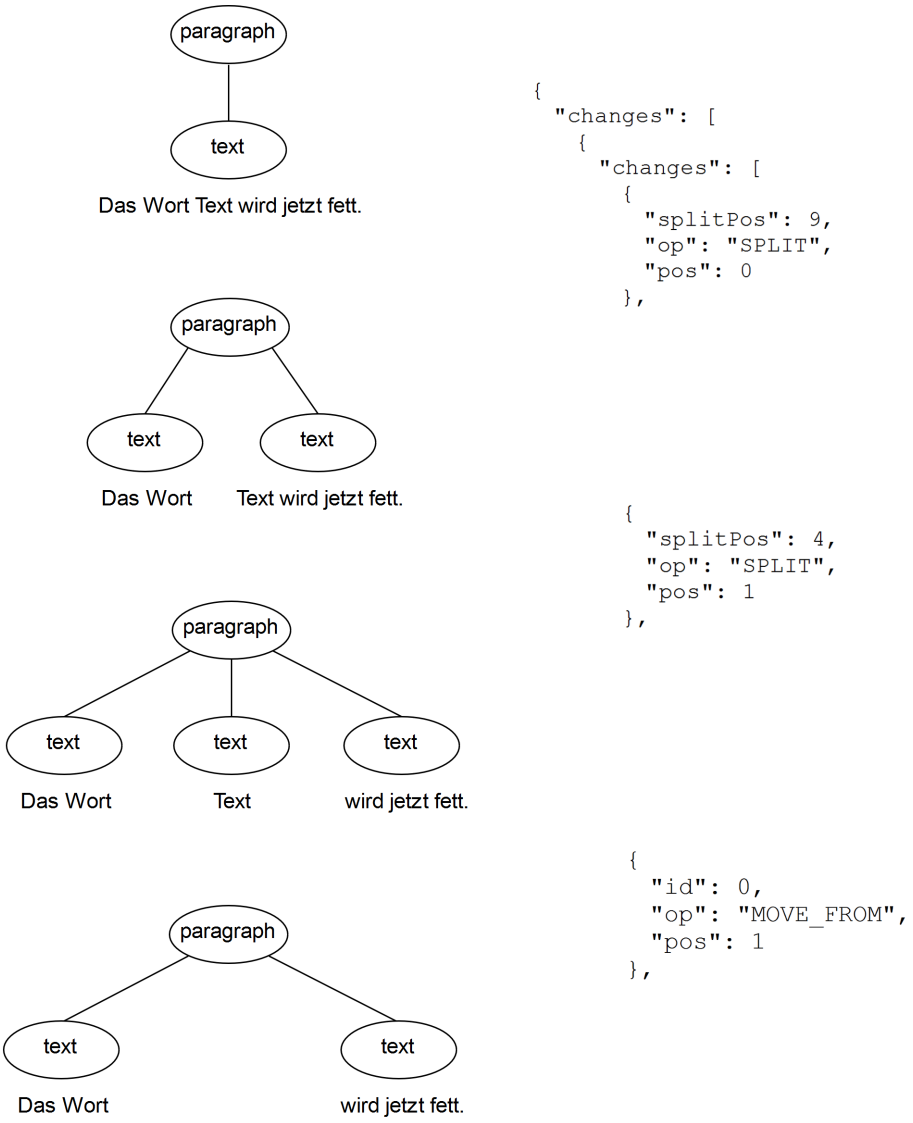
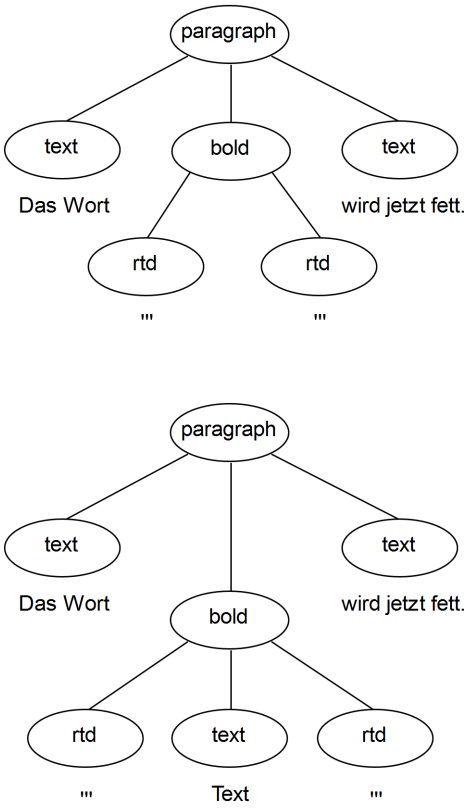


Abbildung 2.5: Transformationen an einem WOM-Baum Teil 1




```
{
  "tree": {
    "!b": [
      {
        "#r": "''''"
      },
      {
        "#r": "''''"
      }
    ]
  },
  "op": "INSERT",
  "pos": 1
},

{
  "changes": [
    {
      "id": 0,
      "op": "MOVE_TO",
      "pos": 1
    }
  ],
  "op": "NODE",
  "pos": 1
}
],
"op": "NODE",
"pos": 0
}
],
"op": "NODE",

}
```


Abbildung 2.6: Transformationen an einem WOM-Baum Teil 2

Wie in dem Beispiel vereinfacht angedeutet, ist es mit Hilfe des vorgestellten *edit script*s möglich, WOM-Dokumente und somit Wikitext-Versionen zu transformieren und damit Aussagen über getätigte Veränderungen zu machen.

3 Architektur und Design

In diesem Kapitel wird das Konzept und die Softwarearchitektur der im Rahmen dieser Arbeit entwickelten Lösung zur Visualisierung von *article diff*s vorgestellt.

3.1 Grundidee

Im Folgenden wird erläutert, wie die Software bestehend aus Frontend und Backend grundsätzlich arbeitet.

3.1.1 Backend

Das zur Verfügung gestellte Backend enthält eine *REST-API*, um die aus Kapitel 2 vorgestellten WOM-Dokumente sowie das *edit script* zu liefern. WOM-Objekte werden mit Hilfe des *Sweble Parser*s aus Wikitext-Dateien erzeugt, die in einem lokalen *Git Repository* gespeichert sind. Es ist möglich, die verschiedenen Wikitext-Versionen vom Server zu erhalten. Das *edit script*, das die Änderungen des Wikitextes beinhaltet, wird mittels HD-Diff Algorithmus erstellt.

3.1.2 Frontend

Über das Frontend kann der Nutzer Wiki-Artikel ansehen und verschiedene Versionen vergleichen, die er zunächst auswählt. Das Frontend steht in Interaktion mit dem Backend, um die entsprechenden Daten zu erhalten. Um die Änderungen visualisieren zu können, benötigt man vom Server die ältere Wikitext-Version als WOM-Dokument und das *edit script*, um einen neuen WOM zu bauen, der die getätigten Änderungen beinhaltet. Hierbei werden am WOM Vorkehrungen getroffen, damit diese entsprechend kenntlich und anzeigbar sind.

3.2 Workflow der Diff Visualisierung

Sobald sich der Benutzer für zwei Versionen entschieden hat und die Artikel Diff Visualisierung anfordert, arbeitet das Frontend grundsätzlich in 4 Schritten:

1. Das Frontend fordert vom Backend die vom Benutzer gewünschte ältere Wiki Artikel Version und das *edit script*, das die Veränderungen der Artikelversionen beinhaltet.
2. Um die neuere Version des Artikels anzuzeigen, muss diese mit Hilfe des Ursprungsartikels und dem *edit script* gebildet werden. Hierbei werden an veränderten Knoten im Artikel-Baum beider Versionen entsprechende Informationen hinzugefügt. Diese dienen als Hilfe, um aus dem Objekt direkt den HTML-Code mit entsprechenden Hervorhebungen genieren zu können und gegebenenfalls Veränderungen zu visualisieren.
3. Nachdem das *edit script* abgearbeitet und beide Artikel Versionen modifiziert wurden, werden diese in eine neue Datenstruktur überführt. Diese Vorgehensweise garantiert eine übersichtliche Gegenüberstellung.
4. Zuletzt wird aus der erstellten Datenstruktur der HTML-Code generiert.

Die genaue Umsetzung des Workflows ist dem Kapitel 4 zu entnehmen.

3.3 Basisarchitektur

Der Benutzer interagiert mit dem System über eine graphische Oberfläche in einer Browserumgebung. Hierbei kann er sich verschiedene Artikel ansehen und auswählen, welche Wikitext-Versionen er vergleichen möchte. Das System steht dabei in Verbindung mit dem Backend und liefert die benötigten Artikel-Versionen sowie das geforderte *edit script* an das System, um die Daten aufzuarbeiten und dem Benutzer anzuzeigen.

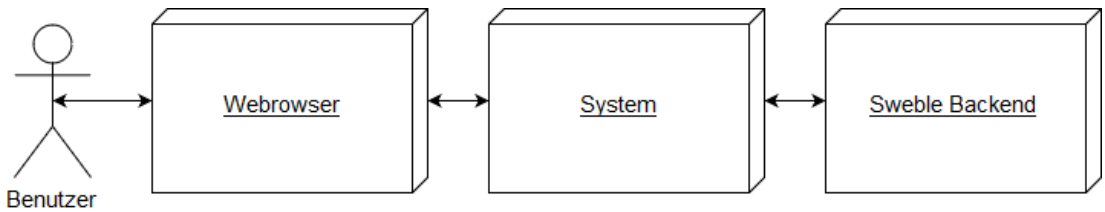


Abbildung 3.1: Basisarchitektur

3.4 Datenfluss im Backend

Die Aufgabe des Backend ist es, aus gespeicherten Wikitext-Dateien WOM-Dokumente bilden und verschiedene Versionen unterscheiden zu können. Für die Speicherung wird ein lokales *Git-Repository* verwendet, das die Wikitext-Versionen verwaltet. Aus diesen Wikitext-Dateien kann mit Hilfe des *Sweble Parser*s ein WOM-Baum generiert werden. Dieses Format wiederum bildet die Grundlage für die Gewinnung des *edit script*s, das der HD-Diff Algorithmus ausgibt. Das *edit script* enthält eine genaue Beschreibung, wie aus der ursprünglichen Artikel-Version die neuere gewonnen werden kann.

Für die Visualisierung des Artikel Diffs ist in Abbildung 3.2 der grundsätzliche Datenfluss grafisch dargestellt, als dessen Ergebnis das *edit script* und die ursprüngliche WOM-Version hervorgehen, welche die Grundlage für das Frontend bilden.

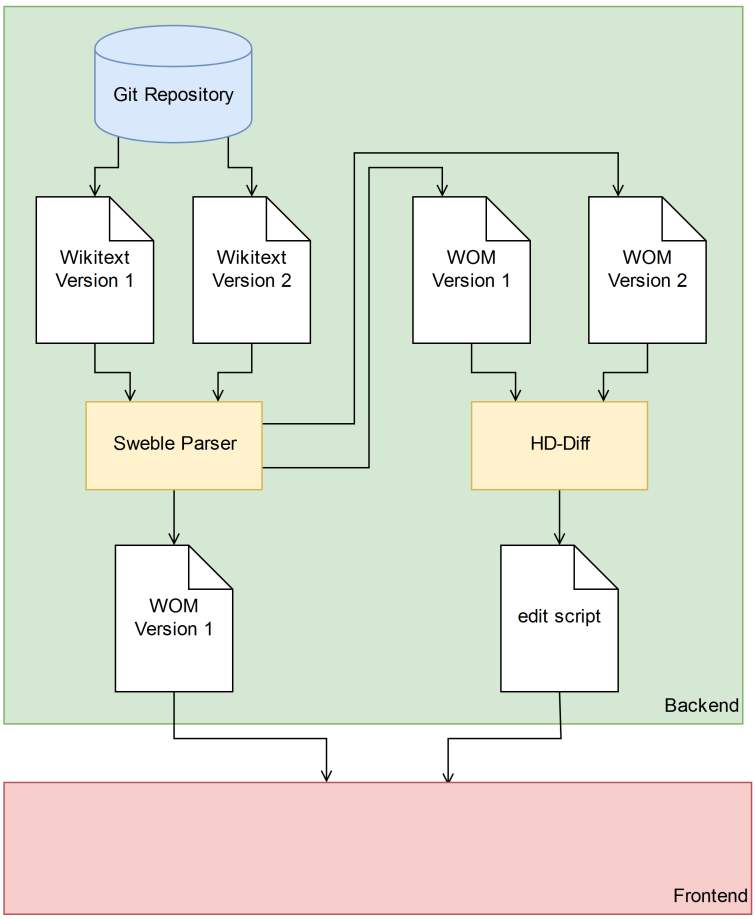


Abbildung 3.2: Datenfluss im Backend für die Diff-Visualisierung

3.5 Aufbau des Frontends

Das Frontend bietet die Möglichkeit, Artikel zu betrachten und Versionen zu vergleichen. Im Folgenden wird grundsätzlich beschrieben, wie das Frontend-System aufgebaut ist und die Artikel-Diff Visualisierungen realisiert.

3.5.1 Komponentenaufbau des Frontends

Um den in Kapitel 3.2 beschriebenen Workflow umzusetzen, werden im Frontend hauptsächlich die Komponenten aus Abbildung 3.3 verwendet, deren Funktionalitäten im weiteren Verlauf kurz erläutert werden. Die genauen Umsetzungsdetails sind dem folgenden Kapitel 4 zu entnehmen.

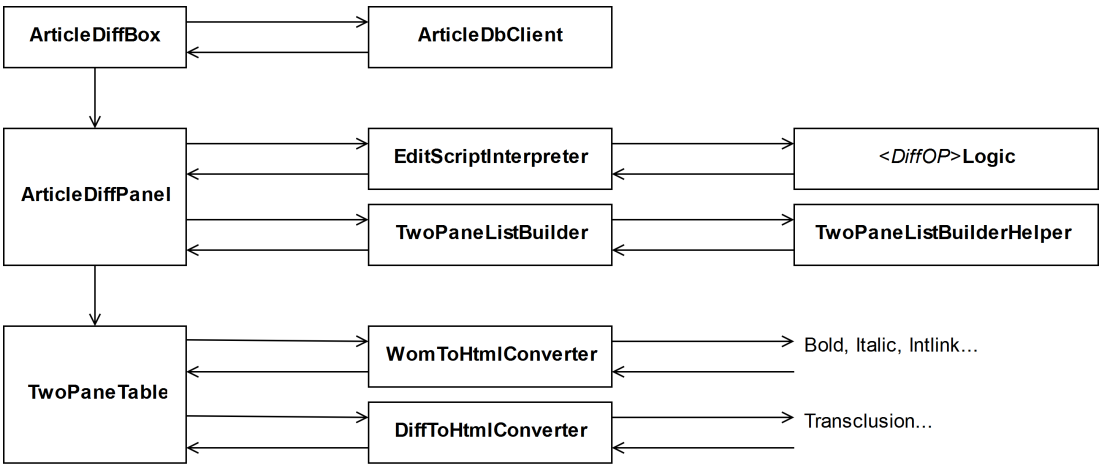


Abbildung 3.3: Zusammenhänge der zentralen Frontend-Komponenten

ArticleDiffBox

Die *ArticleDiffBox* ist dafür zuständig, vom Backend das WOM-Dokument der ursprünglichen Artikel-Version und das *edit script* zu laden und diese an das *ArticleDiffPanel* weiterzuleiten, das die weiteren nötigen Schritte durchführt. Die Daten werden mit Hilfe des *ArticleDbClient*s gewonnen.

ArticleDbClient

Über diese Klasse werden die GET-Anfragen an den Server realisiert. Mittels spezifizierter Eingabeparameter kann somit auf das Daten- und Funktionsangebot des Backends zugegriffen werden.

ArticleDiffPanel

Im *ArticleDiffPanel* wird zum einen die Konstruktion des zweiten WOM-Baums und zum anderen das Speichern der gegenüber zu stellenden Textabschnitte in eine Liste angestoßen. Diese Liste dient dann als Grundlage, um in der *TwoPaneTable* den entsprechenden HTML-Code zu rendern.

EditScriptInterpreter

In dieser Klasse wird der WOM-Baum gebildet, der aus dem ursprünglichen WOM-Dokument und den im *edit script* beschriebenen Funktionen entsteht. Durch paralleles Traversieren des WOM-Baums und des *edit script*s, werden Änderungen je nach Operationstyp in einer *<DiffOp>Logic* in den WOM-Baum übernommen und entsprechend markiert. Das Ergebnis entspricht nach Ausführung der aktuelleren Artikel-Version und steht dem *ArticleDiffPanel* zur Verfügung.

<DiffOp>Logic

Es können folgende Änderungen vom Typ DELETE, INSERT, MOVE, SPLIT und UPDATE am Baum vorgenommen werden, für welche es jeweils eine Logik gibt, die genau dies am übergebenen Baum umsetzt. Es handelt sich dabei um die *DeleteLogic*, *InsertLogic*, *MoveLogic*, *SplitLogic* und *UpdateLogic*, die von dem *edit script* entsprechend angesteuert werden. Hierbei werden veränderte Knoten je nach Modifikation gekennzeichnet.

TwoPaneListBuilder

Um aus zwei WOM-Versionen eine strukturierte Gegenüberstellung anzuzeigen, wird in dieser Klasse der WOM-Baum in Teilbäume aufgeteilt und in eine Liste gespeichert. Ein Listeneintrag enthält einen linken und rechten Teilbaum, sowie Informationen, welche Änderungen im jeweiligen Textabschnitt vollzogen worden sind. Hierbei werden beide WOM-Bäume parallel traversiert und die zuvor spezifizierten Anmerkungen über die Veränderungen ausgewertet.

TwoPaneListBuilderHelper

Diese Klasse unterstützt den zuvor definierten *TwoPaneListBuilder*, indem hier die Kinderbäume, die als nächstes der Liste hinzugefügt werden sollen, gebildet

und zwischengespeichert werden. So wird unter anderem sichergestellt, dass vorangegangene Elternknoten auch beim weiteren Traversieren gespeichert werden, sodass bei der Kreation der Unterbäume keine Information verloren gehen.

TwoPaneTable

Durch die in *TwoPaneListBuilder* entstandene Liste mit gegenüber zu stellenden WOM-Bäumen wird eine Ansicht gebildet und ausgegeben. Dies geschieht je nach Benutzereingabe und Knotentyp in den Klassen *WomToHtmlConverter* und *DiffToHtmlConverter*. Diese Klasse reagiert auch auf Benutzerinteraktionen und verändert dementsprechend den gerenderten HTML-Code.

WomToHtmlConverter

Diese Klasse ist dafür zuständig einen WOM-Baum, zu traversieren und den entsprechenden HTML-Code auszugeben. Dabei werden die Formatierungsdetails in Knotentyp-Funktionen wie beispielsweise *Bold*, *Italic*, *Intlink* usw., die vom *WomToHtmlConverter* angesteuert werden, umgesetzt. Sollten diese Knoten noch weitere Kinderknoten haben, wird diese Klasse erneut mit dem Unterbaum aufgerufen, um den Baum weiter in die Tiefe zu durchlaufen. Diese Klasse erzeugt für jeden Knotentyp eine Ausgabe. Wenn keine Knotenformatierung festgelegt ist, so wird der Wikitext inklusive des syntaktischen *Markup*s ausgegeben.

DiffToHtmlConverter

Analog zum *WomToTHtmlConverter* wird diese Klasse angesteuert, um je nach Knotentyp eine Anzeige zu erzeugen. Hierbei erhält man beide WOM-Versionen, in denen die Änderungen noch ausführlicher dargestellt werden. Diese unterstützt nur spezielle Knotentypen wie beispielsweise eine *transclusion*.

3.5.2 Datenfluss im Frontend

Im Kapitel 3.4 wurde die Datengrundlage für das Frontend beschrieben. In der Abbildung 3.4 wird nun grob dargestellt, wie aus dieser Eingabe der HTML-Code entsteht, der dem Benutzer am Ende angezeigt wird. Hierbei sind die einzelnen Schritte aus dem Workflow aus Kapitel 3.2 gut nachzuvollziehen.

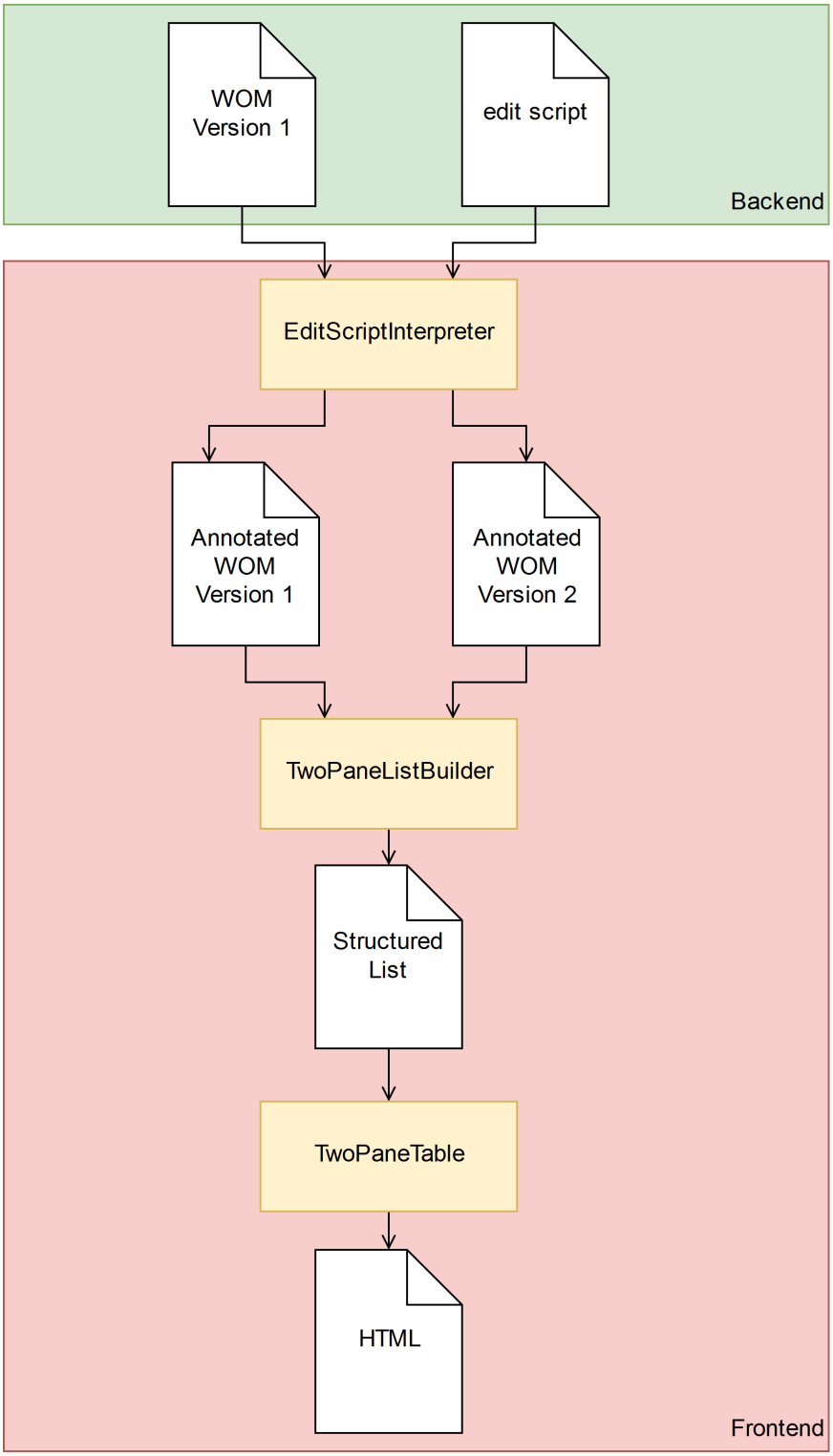


Abbildung 3.4: Datenfluss im Frontend

4 Implementierung

Im vorherigen Kapitel wurde die Architektur der entwickelten Lösung der Artikel Diff Visualisierung vorgestellt. Im Folgenden wird auf genauere Implementierungsdetails eingegangen.

4.1 Kommunikation mit dem Backend

Das auf JAVA basierende Backend stellt eine REST-API bereit, welche im Folgenden genauer vorgestellt wird. Über die Frontend-Klasse *ArticleDbClient* wird mit der Serverkomponente kommuniziert. Diese liefert sämtliche Informationen über Artikeltitel und Artikel-Historie, die sich im Backend *Git Repository* befinden. Wird ein Wiki-Artikel angefordert, so wird dieser als vom *Sweble Parser* gebildeten WOM im JSON-Format überliefert. Änderungen werden mittels HD-Diff Algorithmus in ein *edit script*, wie in Kapitel 2.3.2 dargelegt, geschrieben.

Für die Darstellung einer Diff-Visualisierung benötigt man Zugriff auf verschiedene Versionen eines Wiki-Artikels als WOM-Baum, die nötigen Revision-Identifikatoren der Artikel und das *edit script*, das die getätigten Änderungen definiert. Die genaue Interaktion mit dem Server wird auf der folgenden Seite dargestellt.

```
1   //get article history by parameter articleTitle
2   //returns history specifications of requested article with
     RevisionID, author, time etc. as JSON
3   getArticleHistory(articleTitle) {
4     return request
5       .get(`${this.backendUrl}history?articleTitle=${articleTitle
     }`)
6       .accept('json');
7   }
8
9   //get version of article by parameters articleTitle and
     revisionID
10  //returns requested article (WOM) as JSON
11  getArticleByRev(articleTitle, revSpec) {
12    return request
13      .get(`${this.backendUrl}article?title=${articleTitle}&
     revSpec=${revSpec}&formt=json`)
14      .accept('json');
15  }
16
17  //get article difference by parameters articleTitle,
     RevisionID1, RevisionID2
18  //returns edit script containing version changes as JSON
19  getArticleDifference(articleTitle, version1, version2) {
20    return request
21      .get(`${this.backendUrl}treeDiff?title=${articleTitle}
22      &fromVersionRevSpec=${version1}&toVersionRevSpec=${version2
     }`)
23      .accept('json');
24  }
```

4.2 Konstruktion des WOM-Baumes

In diesem Abschnitt wird die Umsetzung dargestellt, wie mittels eines ursprünglichen WOM-Dokuments und eines *edit script*s, das die am Artikel getätigten Änderungen beschreibt, der neue WOM-Baum gebildet wird.

4.2.1 Parallele Baumtraversierung

Das Absuchen des *edit script*s und das gleichzeitige Traversieren des WOM-Dokuments geschieht in der Klasse *EditScriptInterpreter*. Diese enthält als Eingabe den linken und den rechten Baum, was den zwei verschiedenen Versionen

entspricht, sowie das *edit script*. Beim rechten Baum handelt es sich zu Beginn nur um eine Kopie des linken Baumes, auf dem die Änderungen ausgeführt werden. Ansonsten beinhaltet dieser den aktuellen Abarbeitungsfortschritt. Alle WOM-Knoten haben bereits vor der Eingabe eine *id* erhalten, die deren chronologische Ablaufreihenfolge widerspiegelt (siehe Abbildung 4.1).

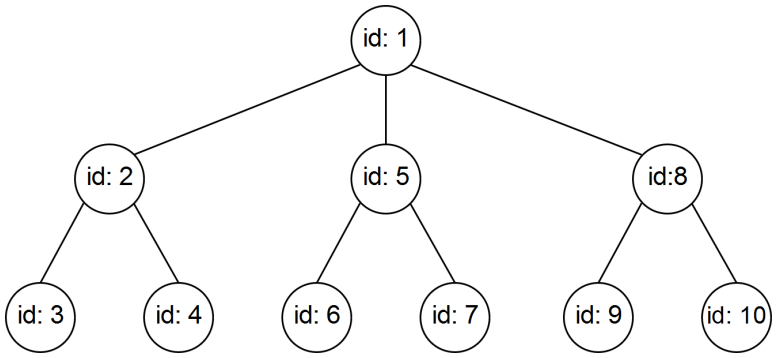


Abbildung 4.1: Anordnung und Reihenfolge der *id*s im Baum

Zusätzlich enthält der linke Baum einen Indikator, ob in einem Knoten ein Zeilensprung stattgefunden hat. Dies ist bereits Vorbereitung für die Strukturierung der Bäume und wird in Kapitel 4.3 genauer erläutert.

Von außen wird nun eine parallele Traversierung angestoßen. Zunächst wird der Wurzelknoten des *edit script*s dahingehend untersucht, ob überhaupt Änderungen getätigt wurden. Ist dies der Fall, werden der Reihe nach die Kinderknoten im Feld *changes* abgearbeitet. Je nach Art von Operation wird die entsprechende Klasse *DeleteLogic*, *InsertLogic*, *MoveLogic*, *SplitLogic* oder *UpdateLogic* angesteuert, welche jeweils die spezifizierten Operationen auf den WOM-Bäumen durchführt. Bei einer NODE-Operation wird der entsprechende Kinderbaum als nächstes betrachtet.

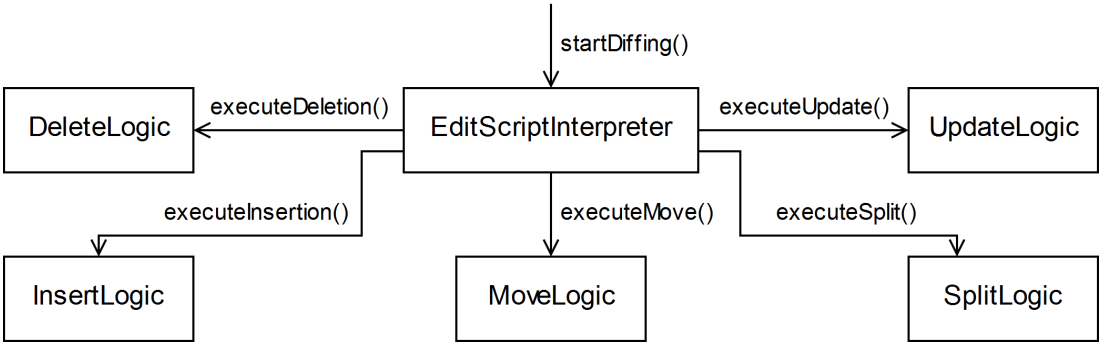


Abbildung 4.2: Aufrufhierarchie des *EditScriptInterpreter*s

In folgendem Pseudocode wird nun das grobe Verfahren erläutert.

```
1  //womv2 is a copy of womv1
2  class EditScriptInterpreter(editScript, womv1, womv2) {
3
4    startDiffing() {
5      if editScript.hasOwnProperty('changes') {
6        for i=0 to editScript.changes.length {
7          getDiffOperationType(editScript.changes[i], womv1,
   womv2)
8        }
9      }
10   }
11
12   getDiffOperationType(editScript, womv1, womv2) {
13     if editScript.op == NODE
14       pos = editScript.pos
15       continueDiffing(editScript, womv1[pos], womv2[pos])
16     if editScript.op == DELETE
17       call DeleteLogic with womv1[pos], womv2[pos]
18     if editScript.op == INSERT
19       call InsertLogic with womv1[pos], womv2[pos]
20     if editScript.op == MOVE_FROM || MOVE_TO
21       call MoveLogic with womv1[pos], womv2[pos]
22     if editScript.op == SPLIT
23       call SplitLogic with womv1[pos], womv2[pos]
24     if editScript.op == UPDATE
25       call UpdateLogic with womv1[pos], womv2[pos]
26   }
27
28   continueDiffing(editScript, womv1, womv2) {
29     curWomv1 = womv1.nodeType //returns child tree of current
   node
30     curWomv2 = womv2.nodeType
31     for i=0 to editScript.changes.length {
32       getDiffOperationType(editScript.changes[i], curWomv1,
   curWomv2)
33     }
34   }
35
36 }
```

Durch das Verwenden von Referenzen werden die von Operations-Logiken angewandten Änderungen auch in die Klasse *EditScriptInterpreter* übernommen, wodurch immer auf dem aktuellen Stand gearbeitet wird.

4.2.2 Umsetzung der Diff-Operationen

Die Aufgaben der im vorangegangenen Unterkapitel 4.2.1 angesprochenen Diff-Operationslogiken werden in diesem Abschnitt im Detail vorgestellt. Jede dieser Logiken erhält als Eingabe die Referenzen auf die WOM-Teilbäume, welche den ursprünglichen Baum und den zu bearbeiteten Baum, der am Ende die zweite Version repräsentiert, entsprechen. Dabei muss man die Höhe im Baum nicht mehr verändern, da dies im *EditScriptInterpreter* (siehe Kapitel 4.2.1) geschieht.

Ziel ist es, am Ende beide Versionen vollständig als WOM-Baum zu erhalten. Außerdem sollen die veränderten Knoten ein zusätzliches Attribut erhalten, das einen festgelegten Zahlenwert für die jeweilige Operation beinhaltet. Dieses Attribut dient zur Visualisierung von Veränderungen, um diese im generierten HTML-Code entsprechend zu markieren.

DELETE-Operation

In der Klasse *DeleteLogic* wird der Knoten inklusive möglichem Kinderknoten, aus dem rechten Baum entfernt, wie es in dem Eingabeparameter *position* spezifiziert wird. Der entsprechende Knoten im linken WOM-Baum erhält mittels *id* den Attributwert, der den Löschvorgang signalisiert.

INSERT-Operation

Wurde Wikitext hinzugefügt, so kümmert sich die *InsertLogic* um das Einfügen des neuen Knotens an der Position, an welcher der Knoten im rechten Baum angefügt werden soll. Wichtig ist hierbei, dass der neue Kinderknoten zwar das entsprechende Modifikationsattribut erhält, aber keine eigene *id*. Die Gründe hierfür werden in Kapitel 4.3 erläutert. Der linke Teilbaum bleibt in dieser Operation unangetastet.

MOVE-Operation

Die Klasse *MoveLogic* hat mehrere Funktionen, um das Verschieben von Wikitext umzusetzen. Zum einen ist diese Klasse in der Lage, Knoten einzufügen und zu löschen, zum anderen speichert sie wichtige Informationen in einer lokalen Liste ab, um das Umlagern des Knotens zu ermöglichen. Ein Listeneintrag enthält als Grundlage eine eindeutige *id* aus dem *edit script*, um das Befehlspaar MOVE_FROM und MOVE_TO zu identifizieren. Bei jedem Aufruf der *MoveLogic* wird zunächst diese Liste nach Einträgen mit der selben *id* durchsucht und

dementsprechend gehandelt. Es treten im Allgemeinen folgende zwei Szenarien auf:

- Die *MoveLogic* wird zunächst mit dem Befehl MOVE_FROM aufgerufen und erst danach mit MOVE_TO. Dabei wird eine Kopie des an der Position befindlichen Kinderbaums in die lokale Liste hinzugefügt und am rechten Baum wird das "Original" entfernt. Bei dem zweiten Aufruf mit MOVE_TO wird der entsprechende Eintrag anhand der *id* herausgesucht und der gespeicherte Knoten dem rechten Baum angefügt.
- Ist die Befehlsfolge genau anders herum, wird in die Liste eine Referenz des Kinderbaumes sowie die Position, an welche der Knoten eingefügt werden soll, hinzugefügt. Es wird im rechten Teilbaum zur Reservierung ein leerer Knoten an der spezifizierten Stelle platziert. Diese wird im späteren Verlauf durch den Knoten ersetzt, der durch den MOVE_FROM-Befehl definiert ist.

In beiden Fällen wird jeweils beim Löschen dem linken Knoten und beim Einfügen dem rechten Knoten das Modifikationsattribut angefügt.

SPLIT-Operation

Die SPLIT-Operation ist die einzige Methode, die gleichermaßen auf beiden Bäumen arbeitet und nur zur technischen Vereinfachung dient. Aufgabe ist es, einen Textknoten an einer in der Eingabe festgelegten Textposition zu teilen. Das Setzen des Attributes ist in diesem Fall nicht nötig, da keine Visualisierung erfolgt.

UPDATE-Operation

In der *UpdateLogic* werden einzelne Knoten mit dem im Parameter festgelegten neuen Wert zum Knoteninhalt aktualisiert. Dies kann auch WOM-Knoten-Attributwerte betreffen. Das zusätzliche Modifikationsattribut wird entsprechend dem linken und rechten Baum angefügt.

Wie die Modifikationsattribute gesetzt werden sollen, wird in folgender Abbildung 4.3 gezeigt. Hierbei handelt es sich um eine vereinfachte Baumtransformation, die veranschaulichen soll, wie Knoten abhängig von Operationen markiert werden. Man kann dem Beispiel auch entnehmen, wie wichtig die *id*s der Knoten für das Setzen der passenden Markierung ist, da sich die Knoten-Position ändern kann.

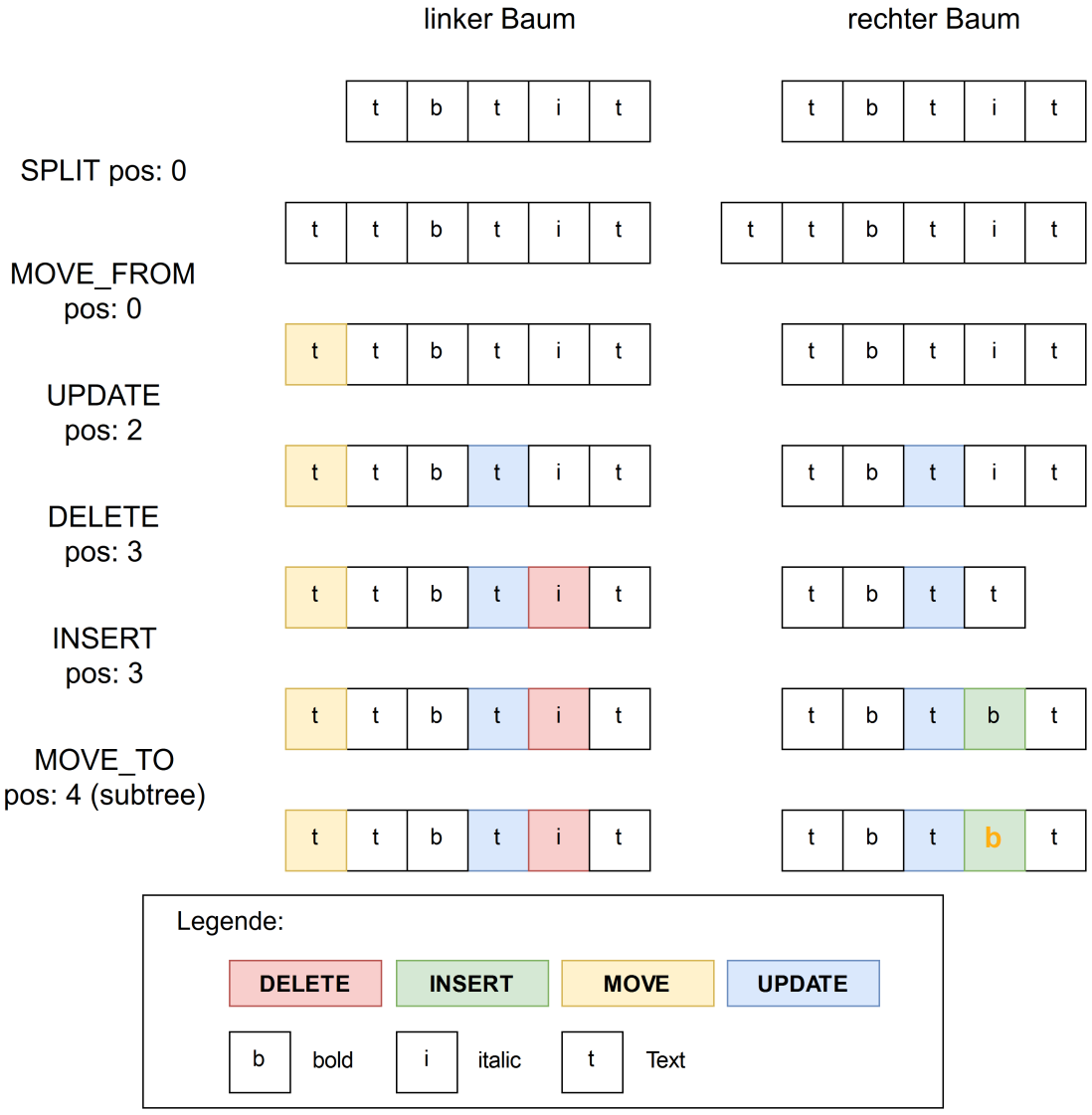


Abbildung 4.3: Beispiel zum Setzen der Modifikationsattribute

4.3 Gegenüberstellung der WOM-Versionen

Um nun die Veränderungen in einer strukturierten zwei-Spalten-Ansicht gegenüber zu stellen, wird der WOM-Baum in eine Liste umgewandelt. Dies dient vor allem der korrekten Ausrichtung (*alignment*), sodass Veränderungen möglichst in gegenüberliegenden Zeilen sichtbar sind und nicht nach oben oder unten verschoben werden.

Für die Umsetzung wird der Klasse *TwoPaneListBuilder* die beiden WOM-Baum-Versionen übergeben. Hierbei enthalten alle Knoten eine Information, ob Zeilensprünge im Knoten vorkommen. Während jedes WOM-Element des linken Baums eine *id* enthält, fehlt diese am rechten Teilbaum bei eingefügten oder verschobenen Knoten.

Nun werden beide WOM-Bäume parallel durchlaufen und passende Teile zusammen in eine lokale Liste eingefügt. Ein Listeneintrag soll die Attribute *changes*, *id*, *left* und *right* enthalten. *left* und *right* sind die linken und rechten Teilbäume, *changes* enthält alle Kennungsnummern von Modifikationen, die in den enthaltenen Teilbäumen aufgetaucht sind und die *id* dient zur Sortierung der Einträge. Alle in dem Abschnitt getätigten Änderungstypen werden deshalb dem Listeneintrag angehängt, um diese nach Bedarf ein- und ausblenden zu können. Näheres hierzu folgt in Kapitel 4.5.2.

Bei der parallelen Traversierung beider Bäume treten folgende Szenarien ein:

1. Einer der Bäume ist schon abgearbeitet, so wird nur der jeweils andere Teilbaum dem Listeneintrag angehängt.
2. Die gegenüberzustellenden Knoten haben dieselbe *id*, weshalb diese gemeinsam dem Listeneintrag hinzugefügt werden und das nächste Knotenpaar betrachtet wird.
3. Der Knoten des rechten Teilbaums hat keine *id* und wurde deshalb hinzugefügt oder verschoben. In diesem Fall wird nur der rechte Knoten dem Listeneintrag hinzugefügt und der linke Teilbaum wird mit dem nächsten rechten Teilbaum verglichen.
4. Die *id* des linken Teilbaums ist kleiner als die des rechten Teilbaums. Dies ist bei gelöschten Knoten der Fall. Hierbei wird nur der linke Teilbaum solange abgearbeitet und der Liste angefügt, bis die *id*-Werte übereinstimmen und Fall 2 erneut eintritt.

Einträge werden dann getrennt, wenn ein Zeilensprung festgestellt wird oder es sich um einen Knotentyp handelt, der eine spezielle Diff Ansichtsunterstützung besitzt. Listeneinträge werden für jede WOM-Version in der Klasse *TwoPaneListBuilderHelper* zwischengespeichert, sodass dafür gesorgt wird, dass die Infor-

mationen der Elternknoten nicht verloren gehen, sobald ein Eintrag der Liste hinzugefügt wird. Das heißt konkret, wenn ein Eintrag der Liste hinzugefügt wird, leert der *TwoPaneListBuilderHelper* alle Knoten außer den aktuellen Pfad bis zum zuletzt hinzugefügten Knoten, sodass der in der Traversion folgende Knoten direkt wieder in den Listeneintrag eingefügt werden kann.

Ein vereinfachtes Beispiel ohne Zeilensprünge ist der Abbildung 4.4 zu entnehmen.

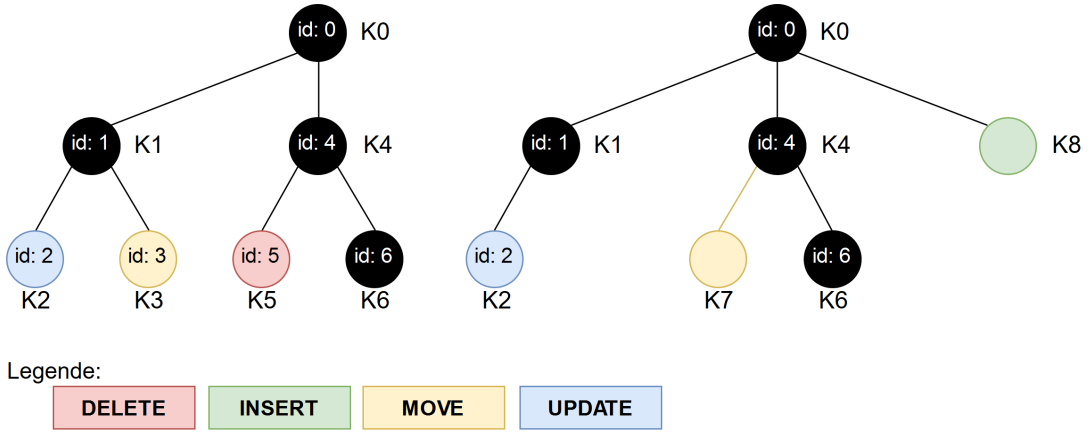


Abbildung 4.4: Einbettung der WOM-Bäume in eine strukturierte Liste

t	Aktueller linker Knoten	Aktueller rechter Knoten	Szenario	links hinzugefügt	rechts hinzugefügt
0	K0	K0	2	K0	K0
1	K1	K1	2	K1	K1
2	K2	K2	2	K2	K2
3	K3	-	1	K3	-
4	K4	K4	2	K4	K4
5	K5	K7	3	-	K7
6	K5	K6	4	K5	-
7	K6	K6	2	K6	K6
8	-	K8	1	-	K8

Die Tabelle zeigt, in welcher Reihenfolge die Knoten einem Listeneintrag hinzugefügt wird. Die Bäume untersuchen ihre Kinderknoten und handeln gemäß obiger Fallunterscheidung. Sollte ein Baum abgearbeitet sein oder existiert der jeweils andere Baum in dieser Tiefe nicht, so wird nur der aktuelle Baum abgearbeitet.

Die genannte Liste dient als Grundlage, um Änderungen direkt gegenüberstellen, zu können. Weiteres dazu folgt in Kapitel 4.4.

4.4 Aufbau der HTML-Seite

Die HTML-Seite wird durch *React*-Komponenten umgesetzt, welche im Folgenden vorgestellt werden. Die zentrale Komponente ist dabei das *ArticleDiffPanel*, welche das WOM-Dokument und das *edit script* entgegennimmt, die Konstruktion der neuen WOM-Version anstößt, sowie die Strukturierung aus Kapitel 4.3 in eine Liste von gegenüberzustellenden Knoten.

Diese Liste wird nun mittels der Komponente *TwoPaneTable* in reinen HTML-Code umgewandelt. Nun wird aus jedem Eintrag ein HTML Block gebildet, wobei sich die Formatierungen je nach Knotentyp unterscheiden können. Die Bäume werden mittels der *WomToHtmlConverter* traversiert und somit der ebenfalls baumartige HTML-Code generiert.

Dem *WomToHtmlConverter* wird der WOM-Baum übergeben. Dann wird überprüft, ob sich in dem Knoten ein Modifikationsattribut befindet. Falls dem so ist, wird je nach Operation eine Farbe als Hintergrund definiert. Grundsätzlich wird danach, oder wenn der Knoten im Allgemeinen unbearbeitet ist, die Knotentyp-Komponente angesteuert. Diese fügt dem Inhalt und Kinderbaum, falls dies nötig ist, eine sinnvolle Formatierung hinzu. So wird beispielsweise der Text eines *bold*-Knotens als fett formatierter Text ausgegeben. Enthält der Knoten noch weitere Kinderelemente, wird erneut die *WomToHtmlConverter*-Klasse aufgerufen, um je nach Knotentyp weiter zu arbeiten. Des Weiteren kann mittels eines Eingabeparameters festgelegt werden, ob eine formatierte Ansicht gezeigt werden soll oder ob man nur reinen Wikitext vergleichen will. Näheres dazu in Kapitel 4.5.

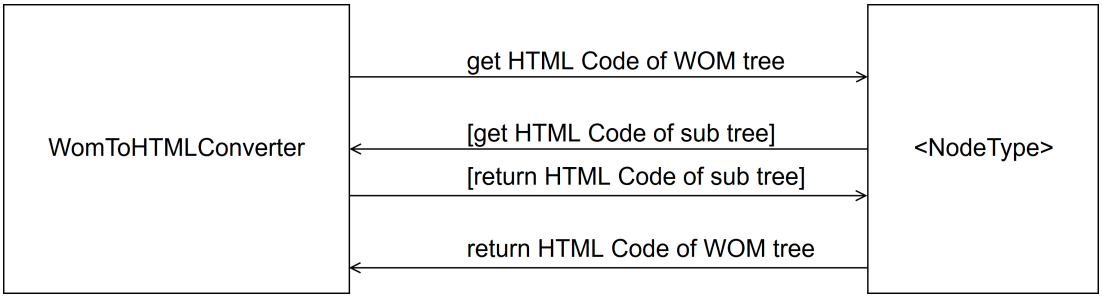


Abbildung 4.5: Arbeitsweise des WomToHtmlConverters

Über den *WomToHtmlConverter* lassen sich allerdings die ursprüngliche und neue Version nur unabhängig von einander konstruieren. Dies ist bei reinem Text auch nicht problematisch, doch um noch aussagekräftigere Diff-Visualisierungen, zu erschaffen, wird die *DiffToHtmlConverter* angesteuert.

Während der *WomToHtmlConverter* nur einen WOM-Baum abarbeitet, erhält der *DiffToHtmlConverter* beide WOM-Dokument-Versionen. Nun wird, falls für

die Knotentypen eine Unterstützung eingebaut wurde, diese *DiffView*-Komponente angesteuert. So wird beispielsweise bei einer *transclusion* die *Transclusion-Diff* aufgerufen, die eine übersichtlichere Darstellung der Änderungen bietet als die reine textuelle Gegenüberstellung. Eine *transclusion* kann zum Beispiel neben einem Namen Attribut-Wert-Paare, wie zum Beispiel "Geburtstag = 24. August 1994"beinhalten. Eine Wikipedia-Infobox wird im WOM-Aufbau als *transclusion* mit beliebig viele Attributen interpretiert. Mit *TransclusionDiff* wird eine Tabelle erzeugt, welche die Attributwerte gegenüberstellt, wodurch die Werteänderungen direkt einsehbar sind.

Die Abbildung 4.6 zeigt den grundsätzlichen Aufbau einer fertigen HTML-Seite.

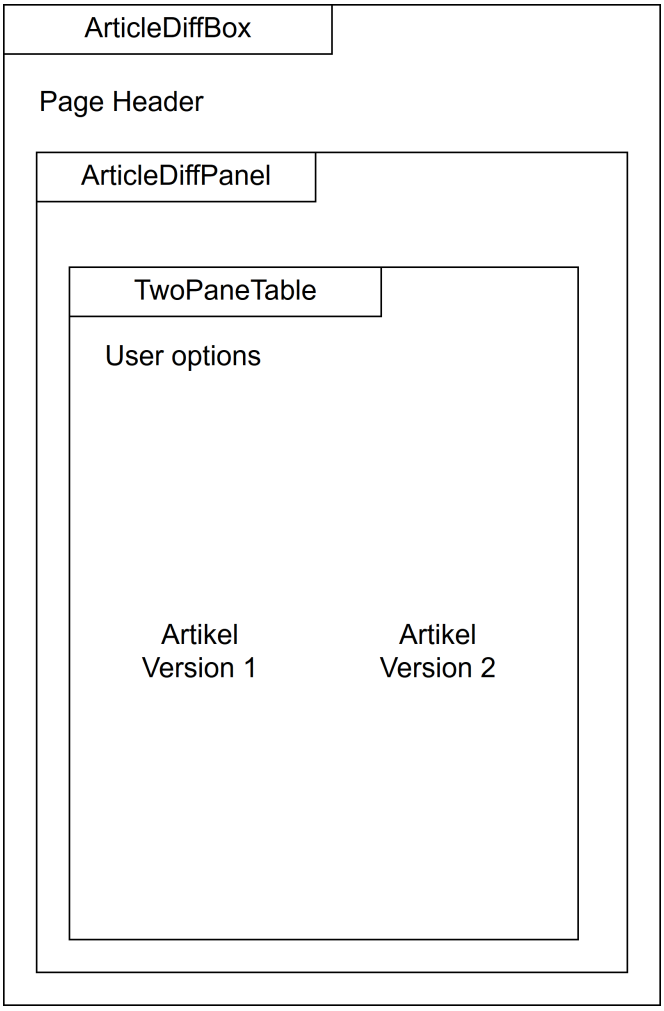


Abbildung 4.6: Grundsätzlicher HTML Seitenaufbau aus React.Components

4.5 Benutzerinteraktionen

Nachdem die in den vorherigen Kapiteln beschriebenen Aktionen durchgeführt wurden, ist der HTML-Code gebildet und man kann das Ergebnis nun in einem Browser öffnen. Hierbei hat der Benutzer mehrere Möglichkeiten, um je nach Wunsch die Versionen, zu vergleichen. Diese sind der Abbildung 4.7 zu entnehmen.

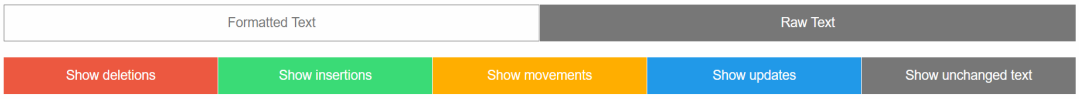


Abbildung 4.7: Ansichtsauswahl für eine Diff-Visualisierung

4.5.1 Wahl der Wikitext-Formatierung

Zunächst kann sich der Benutzer entscheiden, welche Formatierung er für die Anzeige des Wikitextes sehen will. Standardmäßig wird hierbei eine Ansicht präsentiert, die den Wikitext anhand der WOM-Knoten formatiert rendert. Allerdings besteht für erfahrene Benutzer auch weiterhin die Möglichkeit, reine Wikitext-Versionen inklusive der *Markup* Symbole, die farblich hervorgehoben sind, zu vergleichen. Die Abbildungen 4.8 und 4.9 zeigen hierbei jeweils eine exemplarische Artikel-Diff-Visualisierung.

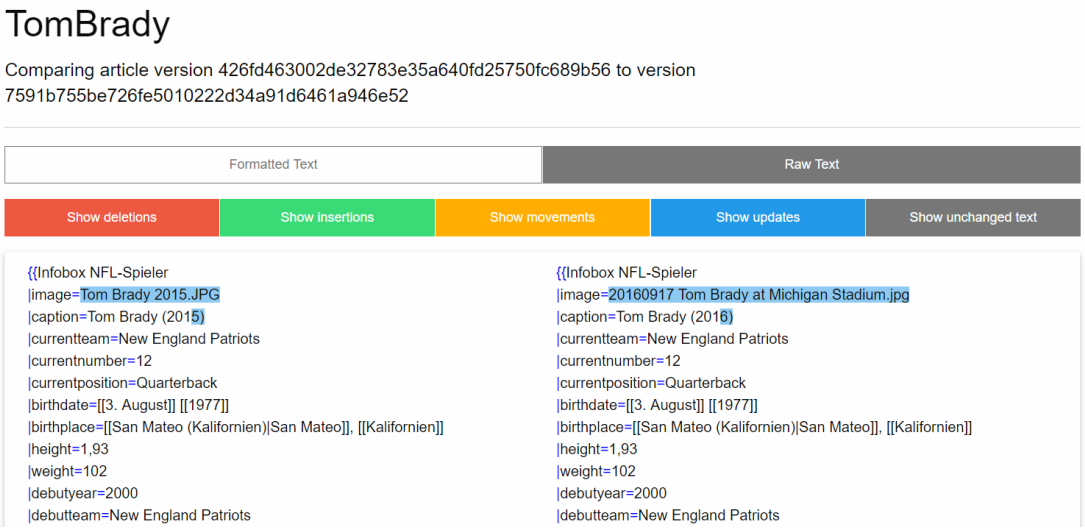


Abbildung 4.8: Diff-Visualisierung des reinen Wikitextes

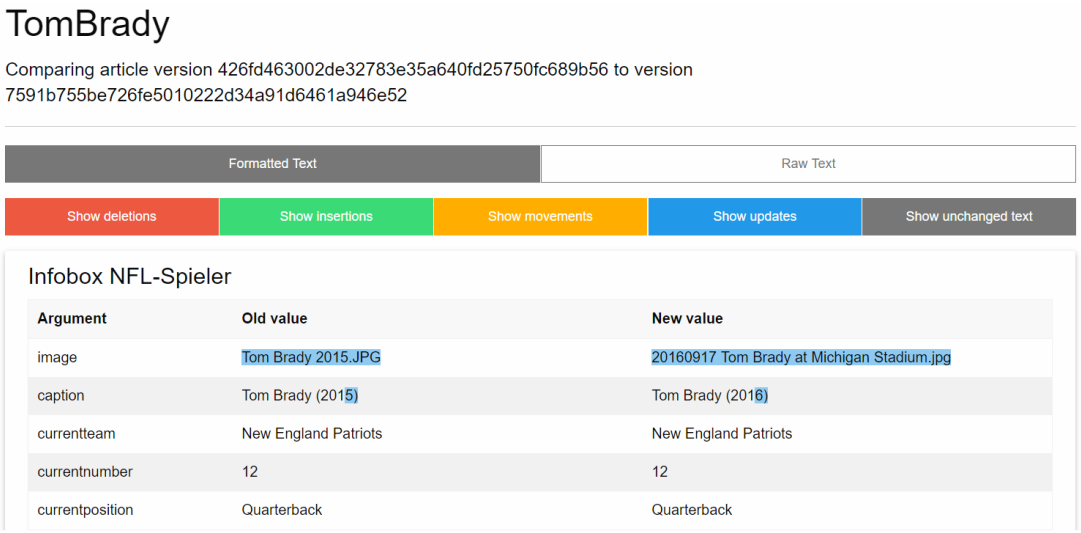


Abbildung 4.9: Formatierte Zwei-Spalten-Ansicht auf zwei WOM-Versionen

4.5.2 Sichtbarkeit je nach Änderungstyp

Neben den rein optischen Möglichkeiten, die Ansicht zu verändern, wird dem Benutzer auch angeboten, nur bestimmte Teile des Artikel Diffs anzuzeigen. So kann dieser wählen, ob er unveränderte, eingefügte, gelöschte, veränderte oder verschobene Textabschnitte in beliebiger Kombination sehen will, um sich einen Überblick über getätigte Änderungen zu verschaffen.

Diese Funktionalität wird mit der im Kapitel 4.4 vorgestellten *TwoPaneTable*-Klasse umgesetzt. Eine Kopie der wie im Kapitel 4.3 beschriebenen Liste wird zu Beginn in *state* der *React.Component* gespeichert und enthält fortan alle anzuzeigenden Textabschnitte. Mittels *handleInputChange(event)* wird eine Änderung dieser Liste angestoßen, sobald der Benutzer eine Schaltfläche aus Abbildung 4.7 anklickt. Hierbei steht zu jeder Zeit die ursprüngliche Liste zur Verfügung. Sobald sich der *state* der *TwoPaneTable*-Komponente ändert, wird auch automatisch der HTML-Code angepasst.

Sollten in einem Textabschnitt mehrere Veränderungstypen enthalten sein, wird dieser solange angezeigt, bis der Benutzer alle enthaltenen Modifikationsarten abgewählt hat. Außerdem werden weiterhin alle Änderungen markiert, unabhängig davon, ob der Benutzer diesen Typ ausgewählt hat, um weiterhin bestmöglich zu identifizieren, wie sich die ursprüngliche Version abgewandelt hat. So kann es passieren, dass beispielsweise ein verändertes Wort nur mit einem darauf eingefügtem Wort einen Sinn ergibt.

4.6 Ergebnis

Das Ergebnis der nun vorgestellten Schritte ist eine strukturierte zwei-Spalten-Ansicht, mit deren Hilfe man verschiedene Wikitext-Artikel-Versionen im direkten Vergleich ansehen kann. Hierbei sind unterschiedliche Ansichten möglich, die der Benutzer frei wählen kann. Durch die detaillierten Markierungen, die das *Wiki Object Model* und das *edit script* ermöglichen, werden Änderungen im Vergleich zum rein textuellen Diff besser sichtbar.

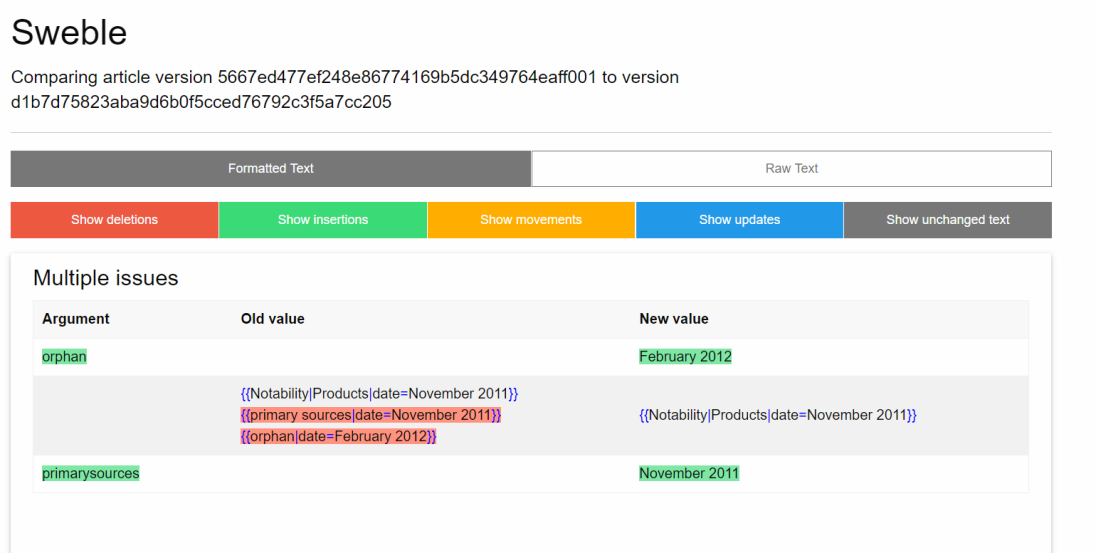


Abbildung 4.10: Screenshot der in dieser Arbeit erstellten Software

5 Evaluation

In diesem Kapitel wird die entwickelte Arbeit gemäß den aus Kapitel 1.2 definierten Anforderungen evaluiert. Hierzu diente als Basis ein zuvor entwickeltes Backend, das WOM-Dokumente und dazugehörige *edit script*s liefern kann.

Wiki Object Model

Die Grundlage für die Anzeige von Wikitext ist in dieser Arbeit ausschließlich aus dem WOM-Baum des Backends gewonnen worden. Die Umsetzung bietet Support für jeden Knotentyp und rendert mindestens den Wikitext. Weitere Formatierungen von Knotentypen können einfach in die vorliegende Implementierung eingearbeitet werden, wenn dies erwünscht ist. Auch weitere spezielle Diff-Anzeigen können in das vorgestellte Projekt eingebettet werden.

Tree-base edit script

Mit Hilfe des Baum-basierten *edit script*s wird das aktuellere WOM-Dokument erzeugt und dient als Grundlage, um Änderungen entsprechend zu markieren. Hierbei sind die Operationen DELETE, INSERT, MOVE, SPLIT und UPDATE umgesetzt worden.

Zwei-Spalten-Ansicht

Als Ergebnis erhält man eine strukturierte Zwei-Spalten-Ansicht, welche die beiden Wikitext-Versionen nebeneinander darstellt. Hierbei werden gleiche Textabschnitte so ausgerichtet, dass sie sich direkt gegenüber stehen, um dem Nutzer somit einen übersichtlichen Vergleich zu ermöglichen. Diese Ansicht kann in gebräuchlichen Browsern angezeigt werden.

Visualisierung von Änderungen

Änderungen werden dem Benutzer je nach Typ unterschiedlich gekennzeichnet und farblich markiert. Hierbei sind unter anderem mit Hilfe der MOVE- und UPDATE-Operation Änderungen deutlicher markiert als in gebräuchlichen Text-basierten Vergleichen, die vornehmlich mit INSERT und DELETE arbeiten. Des Weiteren werden Änderungen von speziellen Typen übersichtlicher dargestellt.

Literaturverzeichnis

Dohrn, H. & Riehle, D. (2011a). Design and implementation of the sweble wikitext parser: unlocking the structured data of wikipedia. In *Proceedings of the 2014 ACM symposium on Documen engineering (DocEng'14)* (S. 72–81).

Dohrn, H. & Riehle, D. (2011b Juli). *WOM: An object model for Wikitext*. University of Erlangen, Dept. of Computer Science. Technical Reports, CS-2011-05.

Dohrn, H. & Riehle, D. (2013). Design and implementation of Wiki Content Transformations and Refactorings. In *Proceedings of the 9th International Symposium on Open Collaboration (WikiSym + OpenSym 2013, Hongkong)*. Article No. 2.

Dohrn, H. & Riehle, D. (2014). Fine-grained Change Detection in Structured Text Documents. In *Proceedings of the 2014 ACM symposium on Document engineering (DocEng'14)* (S. 87–96).

Interview mit Jimmy Wales: Wie geht es weiter mit Wikipedia? (2011). (abgerufen am 13. April 2017). Zugriff unter https://de.wikinews.org/w/index.php?oldid=577184

Hilfe:Versionsvergleich. (2015). (abgerufen am 13. April 2017). Zugriff unter https://de.wikipedia.org/wiki/Hilfe:Versionsvergleich

Help:Wiki markup. (2017). (abgerufen am 13. April 2017). Zugriff unter https://en.wikipedia.org/wiki/Help:Wiki_markup

Hilfe:Versionen. (2017). (abgerufen am 13. April 2017). Zugriff unter https://de.wikipedia.org/wiki/Hilfe:Versionen