Kriterien für interaktive Visualisierbarkeit von Querbeziehungen in XML-Daten und halbautomatische Klassifikation für ausgewählte Dokumentenformate

Masterarbeit

Hochschule Merseburg
Fachbereich Wirtschaftswissenschaften und Informationswissenschaften

Studiengang Informationsdesign und Medienmanagement
Wintersemester 2016/17

Vorgelegt von:
Sandra Zitzmann
Weißenfelser Straße 32
04229 Leipzig

Eingereicht bei:
Dr. rer. nat. Thomas Meinike (Hochschule Merseburg)
Dipl.-Phys. Gerrit Imsieke (le-tex publishing services GmbH Leipzig)

Merseburg, Februar 2017

INHALTSVERZEICHNIS

ABKÜRZUNGSVERZEICHNIS

Ajax	Asynchronus JavaScript And XML
API	Application Programming Interface
CDN	Content Delivery Network
CSS	Cascading Style Sheets
D3.js	Data-Driven Documents JavaScript-Bibliothek
DITA	Darwin Information Typing Architecture
DITA-OT	DITA-Open Toolkit
DOM	Document Object Model
DTD	Document Type Definition
ECMA	European Computer Manufacturers Association
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IETF	Internet Engineeirng Task Force
JSON	JavaScript Object Notation
le-tex	le-tex publishing services GmbH
npm	node package manager
OASIS	Organization for the Advancement of Structured Information Standards
PDF	Portable Document Format
S.	Seite
SGML	Standard Generalized Markup Language
SVG	Scalable Vector Graphics

ABBILDUNGSVERZEICHNIS

TABELLENVERZEICHNIS

QUELLCODEVERZEICHNIS

1 Einleitung

1.1 Unternehmen [1]

Die 1999 in Leipzig gegründete le-tex publishing services GmbH (i. F. kurz: le-tex) ist ein Content Engineering Dienstleister im Bereich des Verlagswesen. Sie hat sich auf die Unterstützung von Verlagen und Organisationen bei der Produktion von Büchern, Zeitschriften und elektronischen Publikationen spezialisiert. Im Sinne der Mission Mitdenken und der Vision Vorausdenken hat le-tex gemeinsam mit zwei weiteren führenden XML-Verlagsdienstleistern das Label "XML made in Germany" mit gegründeten. Hinter dem Label steht eine Initiative mit dem Ziel "die Zukunft des Publizierens aktiv mitzugestalten".[2] Neben einem regelmäßigen Strategieaustausch zählen auch offene Expertengespräche über aktuelle Fragestellungen zum Publishing der Zukunft mit zum Portfolio der Initiative.

Als Verlagsdienstleister ist le-tex jährlich auf der Frankfurter Buchmesse mit einem eigenen Messestand vertreten. Hierbei wird u. a. die Website der le-tex auf einem Touchscreen präsentiert, der Besucher einladen soll das Unternehmen kennen zu lernen. Für diese Präsentation wurde eine interaktive Navigation in die Website eingebunden. Die Verbesserung dieser Navigation wurde auf Basis von D3.js[3] im Rahmen einer durch le-tex betreuten Bachelorarbeit realisiert.

1.2 Problemstellung

Die Website von le-tex wird aus dem von OASIS[4] entwickelten XML[5]-Dokumentenformat DocBook[6] erstellt. Dementsprechend wurde die auf D3.js basierende interaktive Navigation (i.F. kurz Navigator) auf dieses DocBook-Dokument zugeschnitten und eng in das HTML-Rendering der Website eingebunden. Darauf basierend wurde, im Rahmen eines Praktikums der Autorin, der Navigator für DITA[7]-Dokumente angepasst. Hierbei wurde der Navigator modularisiert aufgebaut, indem bspw. die gesamte Steuerung des Navigators in ein XHTML-Dokument ausgelagert und die Erzeugung der Navigator-Daten unabhängig vom HTML-Rendering erfolgte. Dennoch konnte die Modularisierung des Navigators dabei noch nicht soweit umgesetzt,

---

[1] Vgl. le-tex: Mission/Vision, XML made in Germany und Buchmesse; vgl. auch Jache: Interaktive Visualisierung von Querbeziehungen in XML-Daten mit D3.js, S. 6.
[2] le-tex: XML made in Germany.
[3] Data Driven Documents, JavaScript-Bibliothek.
[4] Organization for the Advancement of Structured Information Standards.
[5] Extensible Markup Language.
[6] DocBook Version 5.0, Dokumententyp Buch.
[7] Darwin Information Typing Architecture. Ein weiterer von OASIS veröffentlichter XML-Standard.

dass er als Anwendung auf GitHub gehostet und als externes Repository im Build-Prozess eingebunden werden kann.

Der Navigator soll in beiden Dokumentenformate Querbeziehungen visualisieren, welche durch eine rein hierarchische Struktur nicht abgebildet werden können. Des Weiteren soll er dem Besucher einer Website den Überblick über deren Inhalte erleichtern und ihn bei der Orientierung auf der Website unterstützen. Hierbei entsteht die Frage welche Kriterien eine XML-Datenquelle erfüllen muss, damit die interaktive Visualisierung der enthaltenen Querbeziehungen mit dem Navigator den angestrebten Mehrwert bringt.

1.3 Zielstellung

Ziel der Arbeit ist es ebendiese Kriterien zu definieren und die beiden Navigatoren in eine Navigator-Anwendung zusammenzuführen. Des Weiteren soll ein Navigator entwickelt werden, der auf bestehende HTML-Websites aufgesetzt werden kann. Das hierfür benötigte Spider-Modul soll ebenfalls im Rahmen dieser Arbeit aufgebaut werden. Als ein weiteres Modul der Navigator-Anwendung soll ein Prüftool entwickelt werden. Das Prüftool soll ermöglichen die interaktive Visualisierbarkeit ausgewählter Dokumentenformaten zu bewerten. Hierfür soll es in den Prozess zur Erstellung des Navigators eingebunden werden.

Dem Navigator liegt ein visuelles Konzept zugrunde, welches eine Klassifikation der Daten, die im Navigator dargestellt werden, erfordert. Ein weiterer Aspekt dieser Arbeit ist daher die Untersuchung der Möglichkeiten die Klassifikation halbautomatisch durchzuführen. Dabei soll der Fokus auf ausgewählte XML-Dokumentenformate gelegt werden.

1.4 Methodik

Die Arbeit gliedert sich ein zwei Teile. Der erste Teil widmet sich theoretischen Vorbetrachtungen, auf deren Grundlage anschließend die Navigator-Anwendung weiterentwickelt bzw. erstellt wird.

Zunächst stellt die Arbeit daher verschiedene XML-basierte Dokumentenformate und deren Entwicklung bis zur aktuellsten Version vor. Anschließend werden die Querbeziehungen in den Dokumentenformaten, die für die Einbindung in den Navigator ausgewählt wurden, mit dem Fokus auf die Relevanz für die Navigator-Anwendung untersucht. Im darauf folgenden Abschnitt wird zunächst das Konzept der Millerschen Zahl kritisch gewürdigt, bevor verschiedene Methoden zur Berechnung statistischer Lagemaße thematisiert werden. Darauf aufbauend erfolgt die Definition der Kriterien für die Visualisierbarkeit von Querbeziehungen in XML-Daten. Anschließend werden die Möglichkeiten zur halbautomatischen Klassifikation der Knoten des Navigators untersucht. Die letzten beiden Abschnitten beschließen den Teil der theoretischen Vorbetrachtungen mit der Vorstellung ausgewählter Techniken und Werkzeuge. Die Darstel-

lungen der Techniken und Werkzeuge sind nicht als abschließend zu betrachten, sondern werden entsprechend der Relevanz für die Navigator-Anwendung nur umrissen.
Die Umsetzung der angestellten theoretischen Vorbetrachtungen stellt der zweite Teil der Arbeit dar. Zunächst werden hierfür die Beispiele vorgestellt, anhand derer die Entwicklung der Navigator-Anwendung durchgeführt wird. Anschließend erfolgt die Formulierung der Anforderungen an die Navigator-Anwendung. In den folgenden Abschnitten wird der Entwicklungsstand der vorhandenen Navigatoren, sowie die Grundstruktur, die für alle Navigatoren zu berücksichtigen ist, dargestellt. Die notwendigen Anpassungen der Navigatoren werden daran anschließend erläutert. Bevor die Entwicklung des Spider-Moduls und des HTML-Navigators durchgeführt wird, thematisiert die Arbeit die Einbindung der halbautomatischen Klassifizierung für alle gewählten Dokumentenformate. Die Umsetzung des Prüftools wird im letzen Abschnitt zur Erweiterung der Anwendung dargestellt. Abschließend werden die verschiedenen Module als Navigator-Anwendung zusammengeführt.
Ein Fazit sowie ein kurzer Ausblick auf die Weiterentwicklungsmöglichkeiten der Kriterien und der Anwendung beschließen die Arbeit.

1.5 Begriffsklärung

Die Bezeichnung Navigator für die D3.js-basierte interaktive Navigation wurde während der Entwicklung des DocBook-Navigators gewählt. [8] Während der Anpassung des Navigators für das DITA-Dokumentenformat wurde dieser Begriff entsprechend angepasst. Im Rahmen dieser Arbeit werden der DocBook-, DITA-, und HTML-Navigator unterschieden. Die Navigatoren umfassen dabei jeweils die dokumentenformatspezifischen Pipelines. Der HTML-Navigator schließt außerdem ein Spider-Modul mit ein. Als zusammenfassende Bezeichnung wird der Begriff Navigator-Anwendung genutzt. Die Navigator-Anwendung umfasst neben den einzelnen Navigatoren auch ein Prüftool.
Bereits in der Namensgebung innerhalb des HTML-Navigators wird der Begriff 'spidern' genutzt. Im Rahmen dieser Arbeit wird spidern als Bezeichnung für das Auslesen und Speichern von Informationen einer Website.
Der Begriff Website wird in dieser Arbeit immer als Bezeichnung für vollständige Internetauftritte verwendet. In Abgrenzung hierzu wird mit dem Begriff (Web)Seite eine einzelne Seite einer Website bezeichnet.

[8] Vgl. Jache: Interaktive Visualisierung von Querbeziehungen in XML-Daten mit D3.js, S. 7.

2 Theoretische Vorbetrachtungen

2.1 Auswahl der Dokumentenformate

Die Navigator-Anwendung dient zur Visualisierung von Querbeziehungen in XML-Daten, daher sollen ausschließlich XML-basierte Dokumentenformate ausgewählt werden.

2.1.1 XML

XML (Abkürzung für Extensible Markup Language) ist eine vom W3C entwickelte erweiterbare Auszeichnungssprache. Das textbasierte Format wurde auf Grundlage des SGML[9]-Standards für den Austausch strukturierter Informationen konzipiert. Der XML-Standard definiert keine feste Anzahl von Elementen. Die Bezeichnung, Anzahl und Verschachtelung von Elementen eines XML-Dokuments werden erst über eine DTD[10] oder eine XSD[11] bestimmt. Eine XSD ist dabei selbst ein XML-Dokument. Anhand der erstellten DTD oder XSD können XML-Dokument validiert werden. Ein Textdokument wird jedoch erst durch die Einhaltung der vom XML-Standard vorgegebenen Syntaxregeln (Wohlgeformtheit) zu einem XML-Dokument. [12]

Der XML-Standard sieht strenge Syntaxregeln vor: [13]

- Es gibt exakt ein Wurzelelement.
- Alle Elemente müssen mit Anfangs- und Endtags ausgezeichnet werden. Die verkürzte Schreibweise für leere Elemente ist zulässig.
- Die Groß- und Kleinschreibung der Elementtags muss eingehalten werden.
- Alle Elemente müssen korrekt ineinander verschachtelt sein.
- Alle Attributwerte müssen in Anführungszeichen gesetzt werden.
- Kritische Zeichen (bspw. < und &) müssen als Entität umschrieben werden.

Die Bezeichnungen von XML-Elementen und -Attributen sind grundsätzlich frei wählbar, daher enthält der XML-Standard neben den Syntaxregeln auch Beschränkungen zur Namensgebung:

- Die Groß- und Kleinschreibung wird unterschieden.
- Namen beginnen mit einem Buchstaben oder einem Unterstrich.
- Namen dürfen nicht mit der Zeichenkette 'xml' (unabhängig von der Groß- und Kleinschreibung) beginnen.
- Namen dürfen keine Leerzeichen enthalten.
- Namen können Buchstaben, Zahlen, Trennungsstriche, Unterstriche und Punkte enthalten.

---

[9] Standard Generalized Markup Language.
[10] Document Type Definition.
[11] XML Schema Definition.
[12] Vgl. XML Spezifikation, 2 Documents; vgl. auch SELFHTML (Hrsg.): XML, Regeln/ Wohlgeformtheit.
[13] Vgl. W3CSchools (Hrsg.): XML Tutorial, XML Syntax Rules.

Die Verwendung von Trennungsstrichen und Punkten sollte vermieden werden, weil sie von XML-Interpretern falsch verstanden werden könnten. [14]

Durch die frei wählbare Struktur des Dokuments, können Querbeziehungen nicht universell erkannt werden. Die Identifizierung der Querbeziehungen ist jedoch essentieller Bestandteil der Navigator-Anwendung. In der Navigator-Anwendung sollen XML-Dokumente, welche nicht durch einen weiteren Standard definiert sind, daher nicht integriert werden.

2.1.2 DocBook

DocBook ist ein Dokumentenformat, das insbesondere für Bücher und Dokumente über Computer Hardware und Software geeignet ist. Die Anwendung in anderen Bereichen wird dadurch nicht ausgeschlossen. Als Gemeinschaftsprojekt von HaL Computer und O'Reilly (später von der Davenport Group[15]) wurde DocBook seit 1991 in Form einer SGML[16]-DTD entwickelt. DocBook 3.0 ist die letzte von der Davenport Group veröffentlichte Version. Anschließend wurden die Standardisierungsarbeiten an OASIS übergeben. Die folgende Version, DocBook 3.1 wurde daher 1999 von OASIS als Standard veröffentlicht. Neben der SGML-DTD wurde seit der DocBook 4.1 auch eine DocBook-XML-DTD als Standard entwickelt. Mit der Version 5.0 veröffentlichte OASIS 2009 DocBook erstmals exklusiv als XML-basierten Standard. Weitere wesentliche Änderungen dieser Version sind die Definition des Standards mittels RELAX NG und Schematron sowie die Einführung eines Namensraums für DocBook. Die aktuellste Version, DocBook 5.1 wurde bereits vom Technical Committee genehmigt, aber noch nicht als Standard veröffentlicht. [17]

Der Aufbau von DocBook ist an die hierarchische Struktur von Büchern angelehnt. Als XML-basierter Standard gilt in DocBook, dass jedes Dokument exakt ein Wurzelelement enthalten muss. Die Bezeichnung des Wurzelelements ist abhängig von der Art des Dokuments. Von einzelnen Absätzen, über Abschnitte (bspw. Kapitel, Vorwort, Index oder Glossar), Kapitel und Bücher bis hin zu Buchreihen[18] kann nahezu jede hierarchischen Ebene als separates Dokument mit dem entsprechenden Wurzelelement erfasst werden. Enthält die gewählte Ebene weitere strukturell untergeordnete Ebenen, kann deren Inhalt direkt in dem aktuellen Dokument erfasst werden oder als separate Dokumente eingebunden werden. Die Liste der gültigen

---

[14] Vgl. W3CSchools (Hrsg.): XML Tutorial, XML Elements.

[15] Die Davenport Group war ein Forum für Ersteller von Computer Dokumentationen, das von O´Reilly begründet wurde. 1994 wurde die Davenport Group zur offizielle Instanz für die Wartung von DocBook.

[16] Standard Generalized Markup Language.

[17] Vgl. Walsh: DocBook 5, 1.1. A Short DocBook History; vgl. auch Walsh: DocBook, Schemas/ V5.x und Namespaces/ Overview; vgl auch DocBook 5.0 Spezifikation, Abstract und Declared XML Namespace.

[18] Aufzählung nicht abschließend.

Wurzelelemente ist sehr umfangreich. [19] Daher werden im folgenden Abschnitt die Möglichkeiten zur Verbindung der einzelnen hierarchischen Ebenen nur exemplarisch anhand des DocBook-Dokumenttyps Buch dargestellt.

Ein Buch wird in dem Wurzelelement `<book>` erfasst. Vor dem Wurzelelement können im Prolog die XML-Deklaration mit Version und Zeichencodierung und Verarbeitungsanweisungen zur Einbindung von RELAX NG und Schematron angegeben werden. Diese Angaben sind optional, werden jedoch empfohlen. Außerdem kann im Prolog eine Dokumenttyp-Deklaration erfasst werden. Neben den Verweis auf DTDs[20], können hier individuelle Entitäten definiert werden. Die für das Dokument benötigten Namensräume werden als Attribute des Wurzelelements definiert. Hierbei können beliebig viele Namensräume, aber mindestens der DocBook-Namensraum bestimmt werden. Wie unten in Quellcode 1, Zeile 15 dargestellt, erfolgt die Angabe der genutzten DocBook-Version als Zahl im `@version`-Attribut des Wurzelelements. [21]

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <?xml-model href="http://docbook.org/xml/5.0/rng/docbook.rng"
3    schematypens="http://relaxng.org/ns/structure/1.0"?>
4  <?xml-model href="http://docbook.org/xml/5.0/rng/docbook.rng"
5    type="application/xml"
6    schematypens="http://purl.oclc.org/dsdl/schematron"?>
7
8  <!DOCTYPE book [
9    <!ENTITY name SYSTEM "filename.xml">
10 ]>
11
12 <book
13   xmlns="http://docbook.org/ns/docbook"
14   xmlns:xlink="http://www.w3.org/1999/xlink"
15   version="5.0">
16   <title>Mein Buch</title>
17   &name;
18 </book>
```

Quellcode 1: Aufbau eines DocBook-Dokuments

Das `<book>`-Element kann ein `<title>`- und ein `<info>`-Element enthalten. Das `<title>`-Element enthält den Titel des Buches als Zeichenkette. Es kann in jeder hierarchischen Ebene genutzt werden. Innerhalb des `<info>`-Blocks werden in diversen Elementen Metainformationen zum Buch erfasst, bspw. Autoren und andere Namensnennungen. Dieses Element kann ebenfalls in jeder hierarchischen Ebene genutzt werden. Auch einzelne Absätze können einen `<info>`-Block enthalten. Tabelle 1 beschreibt weitere Strukturelemente, die im `<book>`-

[19] Vgl. Walsh: DocBook 5, 2.4. Roots und 2.2 Physical Division.
[20] Die Einbindung der DocBook-DTDs ist nur dann notwendig, falls das genutzte Entwicklertool diese Angabe bspw. für Syntax-Highlighting benötigt. Ansonsten muss die DTD nicht eingebunden werden, da die Validierung des Dokuments mittels RELAX NG und Schematron erfolgt.
[21] Vgl. Walsh: DocBook 5, 2.1. Making an XML Document.

Element in beliebiger Anzahl vorkommen können. Hierbei wird der Fokus auf Elemente gelegt, die vom Navigator als Knoten dargestellt werden sollen. [22]

Element	Funktion	Verschachtelung mit den anderen Elementen
`part`	Optionale erste Unterebene / Grobe Untergliederung in mehrere Teile	Kann ausschließlich im `<book>`-Element vorkommen.
`preface`	Vorwort und Einleitung	Können im `<book>`- und im `<part>`-Element vorkommen.
`chapter`	Kapitel	
`appendix`	Anhang / Nachwort	

Tabelle 1: Beschreibung ausgewählter DocBook-Strukturelemente[23]

Diese Elemente können in externe Dokumente ausgelagert werden. Für die Einbindung in ein DocBook-Dokument bestehen hierfür verschiedene Möglichkeiten. Einerseits können Entitäten entsprechend dem Schema in Quellcode 1, Zeile 9 (oben) definiert werden, welche zu den gewünschten Dokumenten verweisen. Andererseits kann innerhalb des Wurzelelements das `<xi:include>`-Element genutzt werden. Zuvor muss der XInculde-Namensraum definiert werden. Der Verweis auf die externen Dokumente erfolgt über das `@href`-Attribut. Die Einbindung von externen Inhalten muss innerhalb eines DocBook-Dokuments einheitlich erfolgen. Das Wurzelelement darf nicht in externe Dokumente ausgelagert werden. [24]

Anpassungen der DocBook-Struktur an individuelle Bedürfnisse sind durch Erweiterungen oder Einschränkungen des zugrundeliegenden Schemas möglich. Jedoch darf das resultierende Dokument anschließend nicht mehr als DocBook-Dokument bezeichnet werden. [25]

Im Rahmen seiner Bachelorarbeit entwickelte A. Jache für die le-tex Website einen Navigator mit DocBook als Ausgangsdatenformat.[26] Der vorhandene Navigator muss noch dahingehend angepasst werden, dass er unabhängig von dem HTML-Rendering der le-tex Website erzeugt wird. Nach diesen Anpassungen soll das Dokumentenformat DocBook in die Navigator-Anwendung integriert werden.

2.1.3 DITA

Die Darwin Information Typing Architecture (kurz: DITA) ist eine Architektur, die OASIS erstmals im Mai 2005 als ein XML-Standard für die technische Dokumentation veröffentlichte. Seitdem hat OASIS diesen kontinuierlich weiterentwickelt und den steigenden Anforderungen durch neue Anwendergruppen angepasst. Die aktuellste Version, DITA 1.3 wurde im Dezember 2015 veröffentlicht. Der Standard ist seitdem in drei Editionen unterteilt und umfasst für alle

[22] Vgl. Walsh: DocBook 5, 2.5. Making a DocBook Book, Element Reference /book und /info.
[23] Vgl. Walsh: DocBook 5, Element Reference /part, /preface, /chapter und /appendix.
[24] Vgl. Walsh: DocBook 5, 2.2. Physical Divisions.
[25] Vgl. Walsh: DocBook 5, 5. Customizing DocBook.
[26] Vgl. Jache: Interaktive Visualisierung von Querbeziehungen in XML-Daten mit D3.js.

Editionen eine Spezifikation sowie Informationsmodelle in Form von RELAX NG, DTDs und Schemas. Die Validierung der DITA-Dokumente mit dem RELAX NG wird empfohlen. [27]

Bevor DITA als Standard veröffentlicht wurde, hat die Firma IBM seit den 1990er Jahren verschiedene Techniken entwickelt, die in der Definition von DITA als vereinfachtes XML für technische Dokumentationen mündeten. Ursprünglich wurden diese Konzepte nur firmenintern genutzt. Ziel von DITA war es die Praxis des Information Typing für Print- und Onlineaus-Onlineausgaben zu formalisieren, sowie eine Architektur zur Verfügung zu stellen, die durch die Spezialisierung der Basis-Topics erweiterbar ist. Mit dem Ziel diese Architektur weiter zu verbreiten wurde das Konzept an OASIS übergeben. [28]

Die allgemeine Einheit zur Erfassung von Inhalten ist ein Topic. DITA unterscheidet seit der ersten Version grundsätzlich drei Topictypen, die bereits Spezialisierungen des Ausgangstopic (`<topic>`) darstellen: `<task>`, `<concept>` und `<reference>`. Die einzelnen Topics beinhalten immer einen Titel und optional einen Textkörper. Weiterhin können den Topics Metadaten, eine kurze Beschreibung sowie ein Abschnitt mit Verweisen auf andere Topics angefügt werden. Auch untergeordnete Topics sind möglich. Abhängigkeiten direkt in einem Topic zu verankern sollte jedoch nach Möglichkeit vermieden werden, da die Wiederverwendbarkeit dadurch eingeschränkt werden kann. Die mit einem Inhaltsverzeichnis vergleichbare Strukturierung der Topics für Ausgabezwecke erfolgt durch eine DITA-Map. Hierbei werden mit Hilfe von `<topicref>`-Elementen die einzelnen Topics verlinkt und ein Ausgabedokument wird erzeugt. In einer DITA-Map können Topics auch ineinander verschachtelt werden. [29]

Der DocBook-Navigator wurde (im Rahmen eines Praktikums der Autorin bei le-tex) bereits an das Dokumentenformat DITA angepasst. Der daraus entstandene DITA-Navigator ist bereits dahingehend modularisiert, dass er unabhängig vom HTML-Rendering erzeugt wird. Dennoch sind auch für dieses Dokumentenformat weitere Anpassungen notwendig. Bspw. sind für den Navigator notwendige `click`-Events noch an das HTML-Rendering gebunden. [30] Dieses Dokumentenformat soll daher Bestandteil der Navigator-Anwendung sein.

2.1.4 HTML/XHTML

Die HyperText Markup Language (kurz HTML) ist eine Auszeichnungssprache zur Beschreibung der Struktur einer Website. Ursprünglich für semantische, wissenschaftliche Dokumente ent-

---

[27] Vgl. DITA 1.3 Overview, 1.1 About The DITA specification; vgl. auch Bergert (Hrsg.): Warum DITA sich als Standard durchsetzen wird, S. 1.
[28] Vgl. DITA XML.org (Hrsg.): History of DITA, The 1990's und The 2000's.
[29] Vgl. Bergert (Hrsg.): Warum DITA sich als Standard durchsetzen wird, S. 1-2; vgl. auch Day, Priestley, Schell: Introduction to the Darwin Information Typing Architecture, S. 6.
[30] Vgl. DITA-Navigator Documentation.

wickelt, hat das allgemeine Design von HTML dazu geführt, dass HTML für viele andere Dokumententypen genutzt wird und die Auszeichnungssprache des World Wide Web ist. [31]

Bis zur, im Dezember 1999 veröffentlichten, Version 4.01 war HTML eine SGML-basierte und vom W3C standardisierte Auszeichnungssprache. Anschließend konzentrierte sich das W3C auf die Entwicklung eines XML-basierten HTML-Standards. Mit dem im Januar 2000 veröffentlichten XHTML 1.0 wurde HTML 4.01 in die strengere XML-Syntax übersetzt und entsprechend weiterentwickelt. Dieser Standard sollte mit XHTML 2.0 weiter ausgebaut werden. Parallel hierzu entwickelte die WHATWG[32] den HTML 4.01-Standard unabhängig vom W3C weiter. Die WHATWG versteht HTML als lebenden Standard und verwendet daher keine Versionsnummern. Erst 2013 fanden das W3C und die WHATWG zusammen und der HTML 5.0-Standard wurde gemeinsam von der WHATWG und dem W3C veröffentlicht. Laufende Entwicklungen pflegt die WHATWG weiterhin in ihren lebenden HTML-Standard ein. Auch der W3C arbeitet weiterhin an dem Standard und veröffentlichte im November 2016 die aktuellste Version HTML 5.1. Obwohl HTML und XHTML seit der Version 5.0 einen gemeinsamen Standard teilen, können nicht alle Bestandteile in beiden Ausführungen verwendet werden. Beispielsweise erhalten XHTML-Dokumente einen Namensraum, aber nicht HTML-Dokumente. Hingegen kann das `<noscript>`-Element nur in HTML-Dokumenten, aber nicht in XHTML-Dokumenten genutzt werden. [33] In die Navigator-Anwendung soll daher ein Prozess integriert werden, der die gespiderten Dokumente (unabhängig von der HTML-Version) prüft und in XHTML5-Dokumente übersetzt, sodass diese mit XML-Technologien weiter verarbeitet werden können. Daher wird im Folgenden HTML als Oberbegriff für alle HTML-Versionen genutzt.

Das Wurzelelement `<html>` enthält grundsätzlich zwei Bereiche. Im `<head>` des Dokuments werden beliebig viele Meta-Information erfasst. Wobei mindestens das <title>-Element angegeben werden muss, es sei denn diese Information wird durch ein übergeordnetes Protokoll übergeben. Der `<body>` des Dokuments enthält beliebig viele Sektionen, bspw. ein Navigationscontainer (`<nav>`), Überschriften (`<h1>` bis `<h6>`) sowie Kopf-, Fuß- und Seitenbereiche (`<footer>`, `<header>`, `<aside>` und `<main>`). Alle Sektionen des `<body>` stellen den Inhalt des Dokuments dar. [34]

Der Navigator wurde bereits bei den genannten Dokumentenformaten in HTML-Dokumenten integriert. Daher ist die Erweiterung der Navigator-Anwendung um HTML-Formate nahelie-

---

[31] Vgl. W3C (Hrsg.): HTML & CSS, What is HTML?; vgl. auch

[32] Web Hypertext Application Technology Working Group: Interessenverbund mehrerer Browserentwickler. (Apple, Mozilla, Opera und Google)

[33] Vgl. HTML5.1 Spezifikation, 1.6 HTML vs XHTML; vgl. auch Wolf: HTML5 und CSS3, S. 41; vgl. auch Wenz: JavaScript, S. 383f.

[34] Vgl. HTML5 Spezifikation, 4.2 Document Metadata und 4.3 Sections.

gend. Die Umwandlung der HTML-Dokumente in XHTML ist dabei eine Grundvoraussetzung, damit das Dokumentenformat für den Navigator entsprechend automatisch verarbeitet werden kann. Entsprechende Techniken stehen zur Verfügung, sodass HTML-Dokumente als weiteres Format ausgewählt werden, für die der Navigator erzeugt werden soll. Dabei soll der Navigator auf bereits bestehende Websites aufgesetzt werden. Das hierfür notwendige Spider-Modul wird als weiterer Teil der Navigator-Anwendung entwickelt.

2.1.5 Querbeziehungen in den ausgewählten Dokumentenformaten

In den folgenden Kapiteln werden die Möglichkeiten für interne Querverweise in den zuvor ausgewählten Dokumentenformaten betrachtet. Externe Querverweise werden nicht berücksichtigt, da diese später in der Navigator-Anwendung nicht visualisiert werden sollen.

2.1.5.1 Querbeziehungen in DocBook

Neben den DocBook-spezifischen Verweisen können seit Version 5.0 auch XLinks eingesetzt werden. Querverweise können auf bibliografische Einträge, Fußnoten oder allgemein auf Elemente mit einem Anker zielen. Neben direkt für Querverweise konzipierten Elementen, erlaubt DocBook seit der Version 5.0 das `@linkend`-Attribut für jedes Element zu verwenden. Somit lässt sich jedes Element in einen Querverweis verwandeln. [35] Für die Anpassungen des Navigators soll der Fokus daher weiterhin auf die zwei Elemente `<xref>` und `<link>` gelegt werden, welche vom Standard als allgemeine Querverweise konzipiert sind.

Das `<link>`-Element umschließt den Text, welcher als Link gerendert werden soll. Im Gegensatz hierzu ist `<xref>` ein leeres Element, d. h. der Linktext wird erst beim Rendering erstellt. Beide Varianten können entweder mit dem `@linkend`-Attribut oder als XLink mit dem `@xlink:href`-Attribut IDs anvisieren. Während die ID im `@linkend`-Attribut als einfache Zeichenkette angegeben wird, muss die ID im `@xlink:href`-Attribut um den Fragmentbezeichner Doppelkreuz (`#`) erweitert werden. XLinks können auch URIs beinhalten. Diese müssen nicht durch einen Fragmentbezeichner eingeleitet werden. [36]

2.1.5.2 Querbeziehungen in DITA

Der DITA-Standard sieht grundsätzlich zwei Wege vor Querbeziehungen zu erstellen. Einerseits können sie innerhalb eines Topics mit dem `<xref>`-Element als Inline-Links oder außerhalb des Topicrumpfs als `<link>`-Element erstellt werden. Das `<link>`-Element kann dabei in dem Container-Element `<related-links>` verschieden tief verschachtelt vorkommen. Anderseits

---

[35] Vgl. Schraitle: DocBook-XML, Querverweise und Links /Vergleich verschiedener Verweiselemente und /Interne Links /Zusätzliche Möglichkeiten für DocBook 5.

[36] Vgl. Schraitle: DocBook-XML, Querverweise und Links /Interne Querverweise und /Mit XLinks verlinken; vgl. auch Walsh: DocBook 5, Element Reference /link und /xref.

können Querbeziehungen flexibler mit der Verlinkung über die DITA-Map initiiert werden. Hierfür werden beliebig viele `<topicref>`-Elemente ineinander verschachtelt und gruppiert, sodass eine Hierarchie abgebildet wird. Innerhalb der DITA-Map können dabei Links generiert werden, wenn entsprechende Metadaten verwendet werden. [37]

Auch für die Adressierung der Linkziele sieht der DITA-Standard zwei Varianten vor: direkt und indirekt. Die direkte Adressierung erfolgt über das `@href`-Attribut, dem als Wert eine feste URI zugeordnet wird. Seit DITA 1.2 besteht die Möglichkeit der indirekten Adressierung. Diese basiert auf der Verwendung von Schlüsseln. Diese sog. Keys sind frei wählbare Bezeichnungen für Ressourcen, zu denen Links führen sollen. Der Key wird mit dem `@keys`-Attribut definiert und innerhalb eines Topics über das `@keyref`-Attribut aufgerufen. Alternativ kann die Definition der Keys in der Map mit dem `<keydef>`-Element erfolgen. [38]

2.1.5.3 Querbeziehungen in HTML

Innerhalb von HTML-Dokumenten werden Querverweise ausschließlich über das `<a>`-Element erstellt. Hierfür wird das Linkziel im `@href`-Attribut definiert. Ihm wird eine absolute oder relative URL zugewiesen. Der Verweis auf eine ID muss mit dem einleitenden Fragmentbezeichner Doppelkreuz (`#`) gekennzeichnet werden. Enthält das Element kein `@href`-Attribut, wird es als Platzhalter interpretiert. [39]

2.2 Bestimmung der Kriterien für die interaktive Visualisierbarkeit der Querbeziehungen

2.2.1 Eingrenzung der auszuwertenden Daten

Der Navigator wird durch die Anzahl der Knoten und den Querverbindungen zwischen den Knoten charakterisiert. Dementsprechend sollen diese beiden Elemente zur Bewertung der interaktiven Visualisierbarkeit genutzt werden.

Neben seiner Bezeichnung (`@name`) enthält jeder Knoten Informationen zum Typ (`@type`), die URL (`@url`), Schlüsselwörter (`@keywords`) und eine individuelle ID (`@id`). Jeder Link enthält die Attribute `@source` und `@target`, wobei der Wert dieser Attribute die ID des entsprechenden Knotens enthält.

---

[37] Vgl. DITA 1.3 Spezifikation, 3.2.2.40 <xref>, 3.2.4.1 <link>, Content models for <link>, 2.2.2.3 DITA map elements und 3.3.1 Basic map elements; vgl. auch DITA XML.org (Hrsg.): What is a DITA map?.

[38] Vgl. DITA 1.3 Spezifikation, 3.2.2.40 <xref>, 3.2.4.1 <link>, 3.3.1.2 <topicref>, 2.3.4 Indirect key-based addressing, 2.3.4.1 Core concepts for working with keys und 3.3.2.2 <keydef>.

[39] Vgl. HTML5 Spezifikation, 4.5.1 The a element.

2.2.2 Die Millersche Zahl als Orientierung für die interaktive Visualisierbarkeit

2.2.2.1 Bedeutung der Millerschen Zahl

In der Lernpsychologie ist es unumstritten, dass das Arbeitsgedächtnis nur begrenzt Informationen verarbeiten kann.[40] Die Millersche Zahl ist ein Konzept, welches die Bestimmung dieser Grenze als Ziel hatte. Das Konzept besagt, dass ein Mensch gleichzeitig nur 7 ± 2 Informationseinheiten im Arbeitsgedächtnis speichern kann. Es wurde 1956 von dem US-amerikanischen Psychologen George A. Miller beschrieben. Miller ist dabei nicht der erste Philosoph, der im Zusammenhang mit dem Auffassungsvermögen von Erwachsenen die Zahl Sieben bestimmt. Bereits vor 300 Jahren beschrieb John Locke[41] diesen Wert. [42]

Als weitere Konzepte der Lernpsychologie, welche sich mit der Verarbeitung von Informationen im Arbeitsgedächtnis beschäftigen, sind die Cognitive-Load-Theory und die kognitive Theorie des multimedialen Lernens zu nennen. Beide Konzepte werden an dieser Stelle vernachlässigt, weil sie keine messbare Grenze bestimmen. [43]

Als Informationseinheiten verwendete Miller eindimensionale Reize. Ein Beispiel für eindimensionale Reize sind einzelne Töne. Mit den Versuchen bestimmte er sowohl die Grenze für die Genauigkeit, mit der Menschen die Größe verschiedener Aspekte eines Reizes bestimmen können, als auch die Grenze für die Kapazität des Arbeitsgedächtnisses. Als Bezeichnung für die Genauigkeit, mit der Menschen die Größe verschiedener Aspekte eines Reizes bestimmen können, schlägt Miller den Begriff Absolute Judgement vor. [44]

Die Abgrenzung zum Arbeitsgedächtnis erfolgt über den Umfang der Reize. Während das Absolute Judgement verschiedene Aspekte eines Reizes als einzelnes betrachtet, werden bei dem Arbeitsgedächtnis mehrere Reize mit ihren verschiedenen Aspekten einbezogen. [45]

Millers Versuche ergaben den Grenzwert Sieben sowohl für das Absolute Judgement als auch für das Arbeitsgedächtnis. Dass beide Grenzen identisch sind, ist Miller zufolge Zufall. Beide Werte bilden gemeinsam die Basis für die Fähigkeit eines Menschen Informationen zu verarbeiten. In der Präsentation seiner Forschungsergebnisse geht Miller außerdem darauf ein, dass es auch für die Wahrnehmung von vermischten Dimensionen eine Grenze geben wird. Er ver-

---

[40] Vgl. Hessel: Lernen mit Medien, S. 42; vgl. auch Miller: The Magical Number Seven, Plus or Minus Two, S. 9.
[41] Englischer Philosoph (1632 - 1704), http://www.philosophenlexikon.de/john-locke/.
[42] Vgl. Miller: The Magical Number Seven, Plus or Minus Two; vgl. auch Schoch: The Magical Number Seven, S. 1.
[43] Vgl. Hessel: Lernen mit Medien, S. 45 bis 53.
[44] Vgl. Miller: The Magical Number Seven, Plus or Minus Two, S. 1 und 8; vgl. auch Schoch: The Magical Number Seven, S. 1.
[45] Vgl. Miller: The Magical Number Seven, Plus or Minus Two, S. 9.

mutet diese im Bereich um den Wert Zehn. Dies ist aber nur eine Vermutung, zu der er keine Belege nachweisen kann. [46]

2.2.2.2 Grenzen der Millerschen Zahl

Die Fokussierung auf die Zahl Sieben entstand durch die gewählte Überschrift. Tatsächlich beschrieb Miller die Grenze für die Unterscheidung von eindimensionalen Reizen und die Fähigkeit, diese sofort wieder zu geben. Auf verschiedene Objekte übersetzt, umfasst diese Speicherkapazität sieben Zahlen, sechs Buchstaben oder fünf Wörter. [47] Des Weiteren enthält bereits die ursprüngliche Veröffentlichung zur Millerschen Zahl den Hinweis, dass der Wert Sieben nur begrenzt gültig ist. Die Speicherkapazität des Arbeitsgedächtnisses ist nicht mit der Wahrnehmung von Informationen gleichzusetzen. Die Spannweite der Wahrnehmung sollte demnach bei etwa zehn Einheiten liegen. [48]

Mit sieben Experimenten untersuchte Alan D. Baddeley[49] 1975 die Ergebnisse der Millerschen Zahl für einzelne Wörter. Er kam dabei zu der Erkenntnis, dass die Anzahl der Silben und die Lernstrategien die Anzahl der Wörter beeinflussten, welche sich die Probanden merken konnten. Ein weiterer Einflussfaktor war die Lesegeschwindigkeit der Probanden. Im Ergebnis setzte Baddeley der Kapazität des Arbeitsgedächtnis eine zeitliche Grenze: ein Mensch kann sich demnach so viele Informationseinheiten merken, wie er innerhalb von ein bis zwei Sekunden lesen kann. Weiterhin stellte Baddeley fest, dass zusammengehörige Informationseinheiten von den Probanden besser gemerkt wurden als solche, die nicht zusammenhängen oder verbunden werden können. [50] Mit dieser zeitlichen Grenze bestätigt Baddeley die Forschungen von Peterson und Peterson[51]. Die Ergebnisse wurden außerdem durch die Versuche von Marsh, Sebrechts, Hicks und Landau[52] bestätigt und dahingehend erweitert, dass der Inhalt des Arbeitsgedächtnisses innerhalb von zwei bis 20 Sekunden gelöscht wird. [53]

2.2.2.3 Anwendung auf die Festlegung der Kriterien für die interaktive Visualisierbarkeit

Als Argument gegen die Millersche Zahl als Basis für die Bewertung der optimalen durchschnittlichen Linkanzahl steht der Kontext, in dem diese Zahl gültig ist. In einer veröffentlichten

---

[46] Vgl. Miller: The Magical Number Seven, Plus or Minus Two, S. 8 und 10.
[47] Vgl. Schoch: The Magical Number Seven, S. 2; vgl. auch Halpern, Miller: Citation for your disclaimer.
[48] Vgl. Miller: The Magical Number Seven, Plus or Minus Two, S. 9.
[49] Englischer Psychologe, http://www.york.ac.uk/psychology/staff/academicstaff/ab50/.
[50] Vgl. Baddeley: Word Length and the Structure of Short-Term Memory, S. 12.
[51] Die US-amerikanischen Psychologen Lloyd R. Peterson und Margaret Jean Peterson veröffentlichten 1959 ihre Forschungen an der Indiana University.
[52] Die US-amerikanischen Psychologen Richard L. Marsh, Marc M. Sebrecht, Jason L. Hicks und Joshua D. Landau veröffentlichten 1997 ihre Forschungen an der University of Georgia.
[53] Vgl. Hessel: Lernen mit Medien, S. 43.

E-Mail von Miller[54] betonte dieser nochmals, dass sich die Millersche Zahl auf die Speicherung von Informationseinheiten im Arbeitsgedächtnis bezieht. Dabei sind Informationseinheiten in Form von eindimensionalen Reizen gemeint. [55] Das Verständnis von Texten lässt sich mit diesem Wert nicht messen, da ein Text nicht vollständig gespeichert werden muss und nicht als eindimensionaler Reiz klassifiziert werden kann.

Die von Baddeley entwickelte Spanne für das Arbeitsgedächtnis unterstreicht die begrenzte Gültigkeit der Millerschen Zahl. Des Weiteren wurde die zeitliche Spanne durch diverse Experimente anderer Forscher bestätigt. Unter diesem Blickwinkel ist die zeitliche Grenze als Kriterium für die durchschnittliche Linkanzahl eher geeignet als die Millersche Zahl. Für die Umsetzung in einen automatischen Prüfschritt ist diese zeitliche Grenze jedoch ungeeignet. Einerseits ist der Umfang der behaltenen Informationseinheiten abhängig von der Lesegeschwindigkeit, sodass sich für verschiedene Menschen sehr verschiedene optimale Linkanzahlen ergeben würden. Andererseits kann die Zeit, welche benötigt wird, um die einzelnen Knotenbeschriftungen zu lesen, nicht automatisch gemessen werden.

Ziel des Kriteriums 'Optimale Linkanzahl' ist es, die Qualität der Daten automatisch messbar zu machen. Die Millersche Zahl soll daher als Richtwert für die optimale Linkanzahl genutzt werden, weil dies im Gegensatz zur zeitlichen Grenze von zwei Sekunden ein Wert ist, der in einer automatischen Berechnung berücksichtigt werden kann.

Der Aspekt, dass es insbesondere in der Navigation nicht notwendig ist alle Informationseinheiten nach kurzer Zeit auswendig wiedergeben zu können, soll berücksichtigt werden, indem für die Wertung der Linkanzahl keine radikale Absenkung angesetzt wird, falls ein Knoten mehr als sieben Links beinhaltet. Die begrenzte Bildschirmgröße und damit einhergehende Überlagerung der Knotenbeschriftungen setzt der optimalen Linkanzahl an dieser Stelle eher eine Grenze als die Wahrnehmung von sieben Informationseinheiten. [56]

2.2.3 Statistische Lagemaße zur Ermittlung der durchschnittlichen Linkanzahl

Im Folgenden werden statistische Lagemaße zur Berechnung von Mittelwerten betrachtet, bevor ein Lagemaß für die Ermittlung des Kriteriums 'Durchschnittliche Linkanzahl' ausgewählt wird.

2.2.3.1 Median

Der Median ($\tilde{x}$) gibt den Wert einer Zahlenmenge an, der in der Mitte steht, nachdem die Zahlenwerte in aufsteigender Reihenfolge aufgelistet wurden. Er teilt somit die Zahlenmenge in

[54] Vgl. Halpern, Miller: Citation for your disclaimer.
[55] Vgl. Schoch: The Magical Number Seven, S. 2; vgl. auch Halpern, Miller: Citation for your disclaimer.
[56] Siehe dazu auch 2.2.4 Bewertung der Linkhäufigkeit, S. 26.

zwei gleichgroße Hälften. [57] "Mindestens 50% der [Zahlenwerte] sind kleiner oder gleich dem Median und mindestens 50% der [Zahlenwerte] sind größer oder gleich dem Median"[58].

Die Berechnung des Medians unterscheidet drei Fälle: [59]

- Berechnung bei Einzelwerten
- Berechnung bei unklassierter Häufigkeitsverteilung
- Berechnung bei klassierter Häufigkeitsverteilung

Die Berechnung des Medians bei unklassierter Häufigkeitsverteilung und die Berechnung bei klassierter Häufigkeitsverteilung wird vernachlässigt, da der Median als Möglichkeit zur Bestimmung der durchschnittlichen Linkanzahl genutzt werden soll. Es handelt sich somit ausschließlich um Einzelwerte.

Voraussetzung für die Berechnung ist eine der Größe nach aufsteigend sortierte Zahlenreihe. Jeder einzelne Zahlenwert (x) dieser Reihe bekommt einen Ordnungszahl zugewiesen, die der Position des Wertes in der Zahlenreihe entspricht. Demnach gilt für jeden Zahlenwert, dass jeder Vorgänger kleiner oder gleich dem aktuellen Zahlenwert und jeder Nachfolger größer oder gleich dem aktuellen Zahlenwert ist. Aus dieser geordneten Reihe ist erkennbar, ob es eine gerade oder ungerade Menge an Zahlenwerten ist. Sie bildet außerdem die Basis für den nächsten Schritt zur Berechnung des Medians. Dabei muss unterschieden werden, ob es eine gerade oder ungerade Zahlenmenge (n) ist.

Ungerade Zahlenmenge:

$$\tilde{x} = x_{\left(\frac{n+1}{2}\right)}$$

Gerade Zahlenmenge:

$$\tilde{x} = \frac{1}{2} \cdot \left(x_{\left(\frac{n}{2}\right)} + x_{\left(\frac{n}{2}+1\right)} \right)$$

Das Ergebnis ist der Wert des Medians. Dieser Wert ist unempfindlich gegenüber stark abweichender Einzelwerte, sog. Ausreißer. [60]

2.2.3.2 Arithmetisches Mittel

Das am häufigsten verwendete Lagemaß ist das arithmetische Mittel ($\bar{x}$). Es bildet die Beziehung der Summe aller Zahlenwerte zur Anzahl aller Zahlenwerte ab. [61]

Auch die Berechnung des arithmetischen Mittels unterscheidet die drei von der Berechnung des Medians bekannten Fälle.[62] Die Berechnung wird dementsprechenden wieder auf die Berechnung für Einzelwerte begrenzt.

---

[57] Vgl. Krämer: Statistik für alle, S. 127; vgl. auch Eckey, Kosfeld, Türck: Deskriptive Statistik, S. 60.
[58] Eckey, Kosfeld, Türck: Deskriptive Statistik, S. 60.
[59] Vgl. Eckey, Kosfeld, Türck: Deskriptive Statistik, S. 60, 62 und 63.
[60] Vgl. Eckey, Kosfeld, Türck: Deskriptive Statistik, S. 65f; vgl. auch Krämer: Statistik für alle, S. 128.
[61] Vgl. Krämer: Statistik für alle, S. 13; vgl. auch Eckey, Kosfeld, Türck: Deskriptive Statistik, S. 67f.

Im Gegensatz zum Median müssen die Zahlenwerte $(x_1, x_2, x_3, \dots, x_n)$ vor der Berechnung nicht sortiert werden. Auch eine Unterscheidung zwischen geraden und ungeraden Zahlenmengen entfällt.

Die Berechnung des arithmetischen Mittels erfolgt über den Quotient aus der Summe der Zahlenwerte und der Anzahl der Zahlenwerte (n):

$$\bar{x} = \frac{1}{n} \cdot \sum_{i=1}^{n} x_i = \frac{(x_1 + x_2 + x_3 + x_n)}{n}$$

Der Wert aus dieser Berechnung ist das ungewogene arithmetische Mittel. [63] Auf die Berechnung des gewogenen arithmetischen Mittels durch die Berechnung von unklassierten Häufigkeitsverteilungen[64] wird, wie am Anfang des Abschnitts festgelegt, nicht weiter eingegangen.

Das arithmetische Mittel "balanciert"[65] alle Zahlenwerte und bildet den exakten Durchschnittswert aller Zahlenwerte ab. Entsprechend ihrer Größe, können Ausreißer das arithmetische Mittel stark beeinflussen.

2.2.3.3 Geometrisches Mittel

In der Geometrie gibt es Fälle, in denen das arithmetische Mittel falsche Mittelwerte bestimmt. Regelmäßig ist dies bei der Flächenberechnung der Fall. Als alternative Mittelwertberechnung sollte daher das geometrische Mittel (g) genutzt werden. Das geometrische Mittel bestimmt den Mittelwert über das Produkt aller Zahlenwerte und die Wurzel aus diesem. Wichtigster Anwendungsbereich des geometrischen Mittels ist die Ermittlung von durchschnittlichen Wachstumsraten. [66] Die Berechnung der durchschnittlichen Wachstumsraten mit dem geometrischen Mittel wird nicht näher betrachtet, da der Mittelwert von Zahlenwerten, welche kein Wachstum abbilden, bestimmt werden soll.

Vor der Berechnung des geometrischen Mittels sollten zwei Bedingungen geprüft werden: Die Zahlenwerte (x) dürfen nicht negativ sein, weil sonst die Wurzel nicht berechnet werden kann. Des Weiteren sollte keiner der Zahlenwerte den Wert null besitzen, weil das gesamte Produkt und demzufolge das geometrische Mittel ebenfalls null ergeben würde. [67]

Der Wurzelexponent zur Berechnung des arithmetischen Mittels ergibt sich aus dem Anzahl der Zahlenwerte (n): [68]

---

[62] Vgl. Eckey, Kosfeld, Türck: Deskriptive Statistik, S. 68 - 70.
[63] Vgl. Eckey, Kosfeld, Türck: Deskriptive Statistik, S. 68; vgl. auch Krämer: Statistik für alle, S. 13f.
[64] Vgl. Eckey, Kosfeld, Türck: Deskriptive Statistik, S. 69.
[65] Krämer: Statistik für alle, S. 14.
[66] Vgl. Krämer: Statistik für alle, S. 75f; vgl. auch Eckey, Kosfeld, Türck: Deskriptive Statistik, S. 77f.
[67] Vgl. Krämer: Statistik für alle, S. 76.
[68] Vgl. Krämer: Statistik für alle, S. 76.

$$g = \sqrt[n]{x_1 \cdot x_2 \cdot \ldots \cdot x_n}$$

Das geometrische Mittel ist gegenüber Ausreißern wesentlich geringer anfällig als das arithmetische Mittel. Zur Verdeutlichung soll folgendes, stark überzeichnete Beispiel genutzt werden:

x_1	x_2	x_3	x_4	x_5	x_6
1	1	1	1	1	10.000

arithmetisches Mittel	1.667,50
geometrisches Mittel	4,64

Abb. 1: Vergleich arithmetisches und geometrisches Mittel

2.2.3.4 Harmonisches Mittel

Das harmonische Mittel (h) ist als Quotient aus der Anzahl der Zahlenwerte (n) und der Summe deren Kehrwerte definiert. Somit ist der Kehrwert des harmonischen Mittels gleichzeitig das arithmetische Mittel der Kehrwerte der einzelnen Zahlenwerte. [69]

Voraussetzung zur Berechnung des harmonischen Mittels ist ein konstanter Nenner, welcher sich aus der Anzahl der Zahlenwerte ergibt, deren Mittelwert gesucht wird. [70]

Theoretisch sind alle Zahlenwerte (x) für die Berechnung des harmonischen Mittels möglich. Dennoch sollte die Festlegung getroffen werden, dass das harmonische Mittel den Wert Null erhält, sollte einer der Zahlenwerte Null betragen. Diese Festlegung erleichtert die Berechnung, weil sonst der Grenzwert des harmonischen Mittels berücksichtigt werden müsste.

$$h = \frac{n}{\frac{1}{x_1} + \frac{1}{x_2} + \ldots + \frac{1}{x_n}}$$

Der ermittelte Wert des harmonischen Mittels wird immer niedriger sein, als die ermittelten Werte des arithmetischen und geometrischen Mittels. Typischerweise kommt das harmonische Mittel im Bereich der Musik zum Einsatz, aber auch bei der Berechnungen von durchschnittlichen Geschwindigkeiten.

2.2.3.5 Vergleich der statistischen Lagemaße

Der Vergleich der genannten statistischen Lagemaße erfolgt anhand von vier Beispielen. Ziel des Vergleichs ist die Auswahl eines Lagemaßes zur Bestimmung des Kriteriums durchschnittliche Linkanzahl.

---

[69] Vgl. Weisstein (Hrsg.): Wolfram MathWorld, Stichwort: Harmonic Mean; vgl. auch Krämer Statistik für alle, S. 84.

[70] Vgl. Eckey, Kosfeld, Türck: Deskriptive Statistik, S. 84.

Bsp. 1) Überzeichnetes Beispiel zur Visualisierung der Auswirkung von Ausreißern[71]

Die Werte für dieses Beispiel wurden frei gewählt. In Abb. 2 ist 'Eins' der häufigste Wert der Zahlenmenge. Dieser Wert wird willkürlich als Orientierung und erwarteter Mittelwert verwendet. Dabei bedeutet erwarteter Mittelwert, dass sich der tatsächliche Wert im Vergleich zur gesamten Zahlenmenge am nächsten in diesem Bereich befinden sollte.

x_1	x_2	x_3	x_4	x_5	x_6
1	1	1	1	1	10.000

Median	1,00
arithmetisches Mittel	1.667,50
geometrisches Mittel	4,64
harmonisches Mittel	1,20

Abb. 2: Überzeichnetes Beispiel zum Vergleich der Lagemaße

In diesem Beispiel ist erkennbar, dass der Median durch einen Ausreißer nicht beeinflusst wird. Die weiteren Lagemaße wurden unterschiedlich stark beeinflusst. Das harmonische Mittel ist dem Median und dem erwarteten Mittelwert am nächsten.

Bsp. 2) Garagenbeispiel des DITA-OpenToolkit (i. F. kurz: DITA-OT) [72]

Die Einzelwerte für dieses Beispiel sind dem Garagenbeispiel des DITA-OT entnommen. Als Basis wurde die hierarchisch strukturierte DITA-Map genutzt. Diese Zahlenmenge enthält die Linkanzahl 'Zwei' als häufigsten Wert. Dieser Wert wird wieder als Orientierung und erwarteter Mittelwert für die anderen Lagemaße genutzt.

x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}
1	1	1	2	2	2	2	2	2	2

x_{11}	x_{12}	x_{13}	x_{14}	x_{15}	x_{16}	x_{17}	x_{18}	x_{19}	x_{20}
2	2	2	2	3	3	3	5	7	12

Statistische Lagemaße	Anzahl
Median	2,00
arithmetische Mittel	2,90
Geometrisches Mittel	2,34
Harmonisches Mittel	2,01

Abb. 3: Einzelwerte und statistische Lagemaße des Garagenbeispiels

In diesem Beispiel kann die Linkanzahl 'Zwölf' als Ausreißer bestimmt werden. Die Abweichung ist verhältnismäßig gering, sodass das arithmetische Mittel dementsprechend weniger stark von den anderen Mittelwerten abweicht, als im ersten Beispiel. Das geometrische und harmonische Mittel liegen nah am erwarteten Mittelwert. Der Datenmenge liegt eine hierarchische Struktur zugrunde, dies spiegelt sich in der niedrigen durchschnittlichen Linkanzahl wieder.

---

[71] Fortsetzung des Beispiels aus 2.2.3.3 Geometrisches Mittel, S. 22.
[72] Vgl. Elovirta, Anderson: DITA Open Toolkit.

Bsp. 3) Dokumentation des DITA-Navigator

Ein weiteres Beispiel mit geringer Knotenanzahl ist die Dokumentation des DITA-Navigators.

x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9
2	2	2	3	4	5	5	5	6

Median	4,00
arithmetisches Mittel	3,78
geometrisches Mittel	3,47
harmonisches Mittel	3,16

Abb. 4: Übersicht zur Verteilung der Links und deren statistische Lagemaße

Dieses Beispiel enthält keinen Knoten, der als Ausreißer eingestuft werden kann. Am häufigsten enthalten die Knoten zwei oder fünf Links. Das erwartete Mittel liegt daher in diesem Beispiel bei zwei bis fünf. Alle vier Lagemaßen definieren den Mittelwert in diesem Bereich. Das arithmetische Mittel ordnet sich in diesem Beispiel zwischen den Median und dem geometrischen Mittel ein, da es keine Ausreißer gibt.

Bsp. 4) Website von le-tex

Die Website von le-tex soll als umfangreichstes Beispiel mit 84 Zahlenwerten die Unterschiede der Lagemaße bei größeren Datenmengen veranschaulichen. Hierfür wurden die einzelnen Knoten nach ihrer Linkanzahl gruppiert und in Abb. 5 (unten) als Diagramm abgebildet.

Das Diagramm soll die Verteilung der Links auf die Knoten darstellen. Jede Kugel stellt dabei dar, wie viele Knoten eine bestimmte Anzahl Links haben. Die Anzahl der Links ist auf der y-Achse dargestellt.

Knoten mit drei oder sechs Links kommen am häufigsten vor. Knoten mit fünf oder sieben Links kommen häufiger vor als Knoten mit zwei oder vier Links, sodass dieser Bereich als Orientierung und erwarteter Mittelwert genutzt werden soll. Dieser Bereich ist mit einer roten Ellipse gekennzeichnet. Des Weiteren gibt es einen Ausreißer mit 25 Links. Tabelle 2 (unten) fast die daraus resultierenden Lagemaße zusammen.

Lagemaß	Mittelwert
Median	6,00
Arithmetisches Mittel	6,88
Geometrisches Mittel	5,28
Harmonisches Mittel	3,67

Tabelle 2: Lagemaße der le-tex Website

Das harmonische Mittel ist dem häufigsten Wert (drei Links pro Knoten) am nächsten, der als weniger relevant eingestuft wurde. Das arithmetische und geometrische Mittel sowie der Median befinden sich im Bereich der häufigsten Werte. Der Median ist im Diagramm als grüne Linie gekennzeichnet. Das arithmetische Mittel wird wieder, wie bereits aus den vorherigen Beispielen bekannt, von dem einen Ausreißer mit 25 Links am stärksten beeinflusst.

Abb. 5: Zuordnung der Knoten zur Anzahl der Links

Aufgrund der starken Beeinflussung durch Ausreißer wird das arithmetische Mittel als Lagemaß zur Bestimmung der durchschnittlichen Linkanzahl ausgeschlossen. Das harmonische Mittel scheidet ebenfalls aus. Die auf diesen Weg berechneten Mittelwerte weichen stark von den erwarteten ab.

Der Median birgt den Vorteil, dass er immer einen Wert annimmt, der in den Zahlenwerten tatsächlich vorkommt. Als Nachteil wird dem Median angerechnet, dass die Zahlenwerte erst sortiert werden müssen und im Anschluss daran zwei Fälle zur Berechnung unterschieden werden müssen. Daher wird das geometrische Mittel als Lagemaß ausgewählt, um die durchschnittliche Linkanzahl zu bestimmen.

2.2.4 Bewertung der Linkhäufigkeit

Jeder Anzahl von vorhandenen Links soll ein Wert zugeordnet werden, der sich an der optimalen Anzahl von durchschnittlichen Links orientiert. Basierend auf der Millerschen Zahl und der Wahl des geometrischen Mittels als Lagemaß zur Bestimmung der durchschnittlichen Anzahl der Links wurden folgende Rahmenbedingungen für die Vergabe der Werte festgelegt:

- Sieben Links sollen die höchste Wertung erhalten.
- Enthält ein Knoten fünf bis neun (ausgenommen sieben) Links soll die Wertung nur geringfügig unter der höchsten Wertung liegen.
- Weniger als vier Links sollen eine geringe Wertung erhalten, da diese Linkanzahl auf eine rein hierarchisch strukturierte Gliederung der Daten hinweist und die interaktive Visualisierung somit nicht ihren Zweck erfüllt, Querverbindungen darzustellen, die ansonsten nicht zu erkennen sind. [73]
- Mehr als zehn Links pro Knoten sollen eine geringere Wertung erhalten, weil mit steigender Linkanzahl die Übersichtlichkeit des Navigators verloren geht. Bereits bei Querverbindungen zu elf weiteren Knoten werden diese sehr dicht nebeneinander dargestellt, sodass sich die Knoten berühren. Bei Querverbindungen zu 14 weiteren Knoten kommt es zur Überlagerung der Beschriftung der einzelnen Knoten.
- Der Linkanzahl Null soll der Wert Null zugeordnet werden, sodass die Berechnung des geometrischen Mittels Null ergibt. Sollte ein Knoten keine Verbindung zu anderen Knoten haben, sind die Daten nicht für diese Art der Visualisierung geeignet.
- Die Berechnung des geometrischen Mittels umfasst die Radizierung des Produkts aus den Zahlenwerten. Es ist nicht zulässig aus einer negativen Zahl die Wurzel zu ziehen. Daher dürfen die Wertungen nicht kleiner als Null sein.

Auf Grundlage der genannten Rahmenbedingungen wurden die Linkanzahlen von 0 bis 16 bewertet. Dabei soll der Wert für 16 Links allen weiteren höheren Linkanzahlen zugeordnet werden. Der Wert Null wurde für diesen Bereich ausgeschlossen, weil der gesamte Mittelwert null betragen würde. Eine zu hohe Linkanzahl eines einzelnen Knotens kann auch durch einen Ausreißer zustande kommen. Daraus lässt sich nicht schließen, dass die gesamten Daten nicht für die interaktive Visualisierung geeignet sind.

Als höchster Wert wurde '1' bestimmt, weil die Wertung in dem Prüfschritt als Prozentsatz ausgegeben werden soll.

Die Bewertung der Linkanzahl ist dabei von der Ermittlung gewichteter Durchschnitte zu unterscheiden. Der gewichtete Mittelwerts wird berechnet, in dem einzelne Werte unterschiedlich stark in die Berechnung einfließen.[74] Zur Ermittlung der durchschnittlichen Linkanzahl sind die Einzelwerte als gleichwertig anzunehmen, weil jeder Link von jeweils einem Knoten zu jeweils einem anderen Knoten führt.

---

[73] Siehe dazu auch 2.2.3.5 Vergleich der statistischen Lagemaße: Bsp. 2), S. 24.
[74] Vgl. Krämer: Statistik für alle, S. 78.

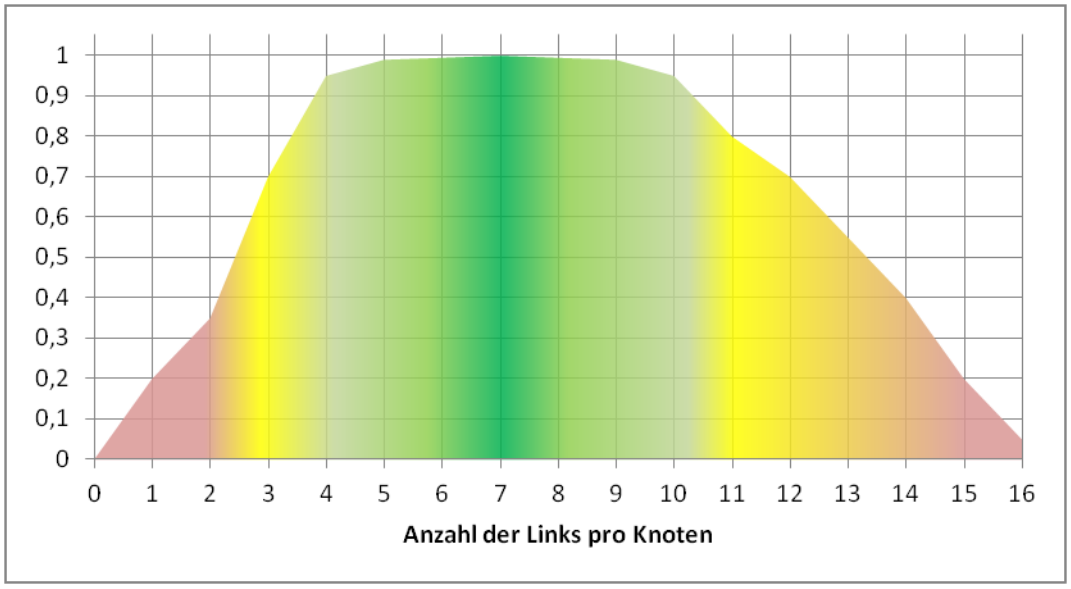


Abb. 6: Bewertung der durchschnittlichen Linkanzahl

Unter Berücksichtigung der vorgenommenen Wertung der einzelnen Linkanzahlen, wurden in Tabelle 3 die Mittelwerte für zwei Beispiele berechnet. Zum Vergleich sind die Mittelwerte der durchschnittlichen Linkanzahl ohne Wertung nochmals aufgeführt.

Beispiel	DITA-OT		le-tex	
Statistische Lagemaße	Anzahl	Wertung	Anzahl	Wertung
Median	2,00	35,00%	6,00	99,50%
Arithmetische Mittel	2,90	46,20%	6,88	72,32%
Geometrisches Mittel	2,34	41,04%	5,28	59,55%
Harmonisches Mittel	2,01	36,93%	3,67	39,41%

Tabelle 3: Auswirkung auf die Eignung der Daten durch die Bewertung der Linkanzahl

Das arithmetische Mittel des zweiten Beispiels liegt bei der Anzahl im optimalen Bereich von etwa sieben Links pro Knoten. Hier wäre eine Wertung von nahezu 100 % zu erwarten. Werden nun die Linkanzahlen der einzelnen Knoten bewertet und der Mittelwert aus diesen Bewertungen berechnet, ergibt sich mit 72 % eine deutlich schwächere Bewertung. Wird dieser Wert mittels Abb. 6 wieder in eine durchschnittliche Linkanzahl übersetzt, ergibt sich eine durchschnittliche Linkanzahl von ca. drei oder zwölf Links pro Knoten. Daraus ist ersichtlich, dass die Wertung der Linkanzahl pro Knoten die gesamte Bewertung der Daten verfälscht.

Um dieser Verfälschung entgegen zu wirken, soll die Bewertung der Linkhäufigkeit erst erfolgen, nachdem die durchschnittliche Linkanzahl berechnet wurde. Für die Bewertung soll dennoch die in Abb. 6 dargestellte Skale verwendet werden.

2.2.5 Eingrenzung der Knotenmenge

Die Anzahl der Knoten ist das zweite Kriterium, welches die Visualisierbarkeit von Querbeziehungen messbar machen soll.

Als optimale durchschnittliche Anzahl der Links wurde der Wert Sieben bestimmt.[75] Eine Datenmenge benötigt mindestens acht Knoten, damit ein Knoten sieben Querverbindungen enthalten kann. Daher werden acht Knoten als Untergrenze für die Visualisierbarkeit von Querbeziehungen in XML-Daten bestimmt.

Die Definition der Obergrenze wird nicht an diesem Wert gemessen. Die Millersche Zahl bezieht sich auf die Speicherung der Information[76], was bei der Navigation nicht zwingend notwendig ist. Zur Orientierung sollen daher drei Beispiele mit unterschiedlicher Knotenmenge verglichen werden. [77] Für jedes Beispiel wurde bereits in vorangegangenen Projekten eine Navigator-Anwendung erstellt.

Mit neun Knoten überschreitet die Dokumentation des DITA-Navigators[78] nur knapp die festgelegte Mindestanzahl von acht Knoten. Die geringe Knotenmenge bietet den Vorteil, dass die Beschriftung der Knoten auch dann lesbar ist, wenn alle Knoten angezeigt werden.

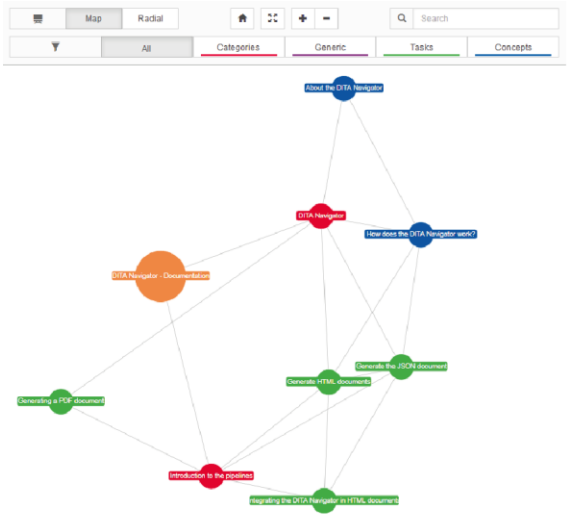
Sollen alle Knoten des Navigators sichtbar sein, muss bereits bei dem Garagenbeispiel des DITA-OT[79] die Beschriftung der 20 Knoten so klein dargestellt werden, dass diese nicht mehr lesbar ist. Ein Wechsel in die radiale Ansicht des Navigators schafft hierbei begrenzt Abhilfe. Die Beschriftung ist gerade noch lesbar, aber stark an der Grenze. Jedoch ermöglicht die erhöhte Knotenmenge im Vergleich zum vorherigen Beispiel mehr Querverbindungen. Gleichzeitig bringt die interaktive Visualisierung in diesem Fall den Vorteil, dass Verbindungen erkennbar sind, die in einer hierarchischen Navigation nicht sofort sichtbar wären. Dieser Vorteil stellt eines der Hauptziele des Navigators dar. Die verringerte Lesbarkeit der Beschriftungen kann durch die Zoom-Funktion aufgehoben werden. Dann würden nicht alle Knoten angezeigt werden, aber der Nutzer kann die Beschriftung lesen und den Kartenausschnitt so weit verschieben, dass er alle Knoten sieht, welche durch ein Linie mit dem ausgewählten Knoten verbunden sind. Daher wird der Nachteil der zu kleinen Schrift geringer gewichtet und die 20 Knoten als akzeptable Knotenmenge eingestuft.

Das dritte Beispiel, die Website von le-tex[80], ist mit 84 Knoten am umfangreichsten. Auch in diesem Fall ist die Beschriftung der Knoten nicht mehr lesbar, wenn alle Knoten angezeigt werden. Auch der Wechsel in die radiale Ansicht löst die Situation nicht auf. Wie im vorherigen Beispiel beschrieben, kann die Ansicht heran gezoomt und der Bildausschnitt entsprechend gewählt werden. Die Größe der Knotenmenge bringt entsprechend viele Querverbindungen

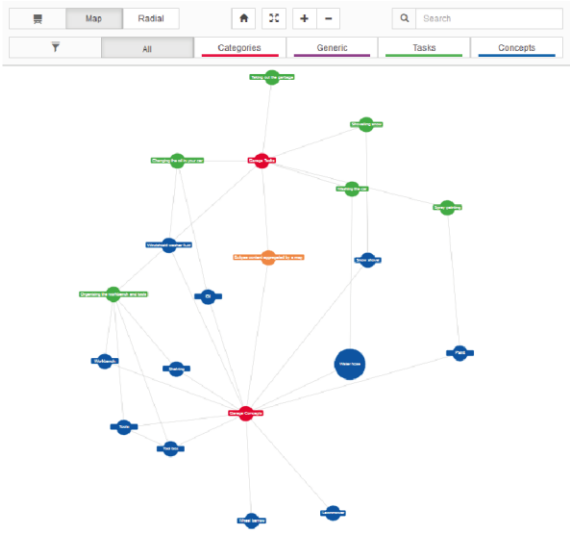
---

[75] Siehe dazu auch 2.2.3.5 Vergleich der statistischen Lagemaße, S. 23.
[76] Siehe dazu auch 2.2.2.2 Grenzen der Millerschen Zahl, S. 19.
[77] Siehe dazu auch Abb. 7: Darstellung aller Knoten im Kartenmodus für verschiedene Beispiele, S. 30.
[78] Vgl. DITA-Navigator Documentation.
[79] Vgl. DITA-Navigator.
[80] Vgl. le-tex.

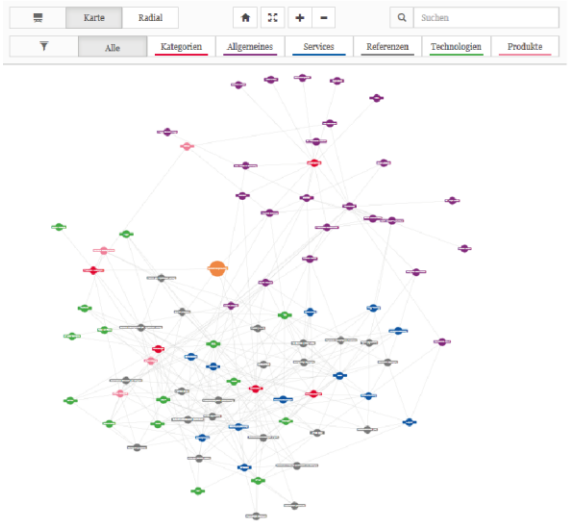
mit sich. Da jede Querverbindung als graue Linie dargestellt wird, sind Querverbindungen des aktuellen Knotens nicht sofort erkennbar. Erst wenn der Mauszeiger auf den gewünschten Knoten hovert, werden die aktuellen Querverbindungen hervorgehoben, in dem die übrigen Querverbindungen und Knoten transparenter angezeigt werden. Über diesen Weg wird das Ziel der interaktiven Visualisierung noch erreicht. Die einzelnen Knoten liegen sehr dicht beieinander, woraus ein weiterer Nachteil entsteht: an einigen Stellen kommt es zu Überlagerungen der Knotenbeschriftung. Dieser Fall tritt sowohl bei der radialen Ansicht als auch bei der Kartenansicht auf. Weitere Knoten sind denkbar, ohne dass die genannten Vor- und Nachteile verstärkt würden, daher wird die Obergrenze auf 90 Knoten festgelegt.



a) Dokumentation des DITA-Navigators mit 9 Knoten



b) Garagenbeispiel des DITA-OT mit 20 Knoten



c) le-tex Website mit 84 Knoten

Abb. 7: Darstellung aller Knoten im Kartenmodus für verschiedene Beispiele

2.2.6 Möglichkeiten zur Integration der Kriterien in die Navigator-Anwendung

Die ermittelten Kriterien sollen in die Navigator-Anwendung eingebunden werden. Hierfür soll in den Prozess zur Erstellung des Navigators eine Prüfschritt integriert werden, welcher dem

Nutzer einen Hinweis ausgibt, falls die gewählten Daten den aufgestellten Kriterien nicht ausreichend entsprechen.[81] Eine negative Einschätzung der Daten soll als Hinweis dienen, aber nicht die Erstellung des Navigators blockieren.

2.3 Halbautomatische Klassifikation der Knoten des Navigators

Als Klassifikation wird die systematische Einteilung von Objekten in Gruppen und Untergruppen bezeichnet. Gruppen und Untergruppen sind dabei durch bestimmte Merkmale charakterisiert. Die hieraus resultierende Verteilung der Objekte auf die verschiedenen Gruppen und Untergruppen soll gleichmäßig erfolgen, sodass keine Gruppe über- oder unterrepräsentiert ist. [82]

Im Rahmen der Navigator-Anwendung sollen die Knoten klassifiziert werden. Hierfür werden verschiedene Typen festgelegt, denen die Knoten anschließend zugeordnet werden. Diese Klassifikation ist als halbautomatisch zu bezeichnen, da die Benennung der Typen manuell und nur die Zuordnung der Knoten zu diesen automatisch erfolgt. Die Kriterien für diese Zuordnung müssen ebenfalls manuell festgelegt werden. [83]

Die gewählten Knoten-Typen werden im Navigator mit unterschiedlichen Farben dargestellt. Insbesondere bei einer hohen Knotenanzahl unterstützt die farbliche Differenzierung wesentlich die Orientierung innerhalb der Navigation. Des Weiteren kann die Navigator-Anwendung die verschiedenen Knoten-Typen filtern, sodass nur die Knoten einer bestimmten Kategorie angezeigt werden.

2.4 Techniken

Die folgenden Unterkapitel stellen Techniken vor, welche bei der Erstellung der Navigator-Anwendung zum Einsatz kommen. Dabei konzentrieren sie sich auf die Aspekte der einzelnen Techniken, welche für die Anwendung relevant sind.

2.4.1 CSS

2.4.1.1 Versionsgeschichte

Cascading Style Sheets, kurz CSS, ist eine Stylesheet-Sprache für die Gestaltung von HTML-Dokumenten. Entwickelt von Håkon Wium Lie und Bert Bos wurde sie als CSS Level 1 (kurz: CSS1) 1996 vom W3C erstmals als Standard veröffentlicht. Mittels CSS können Schrift, Farben und Layout von HTML-Dokumenten gestaltet werden. Als 'living standard' wird CSS beständig

---

[81] Siehe dazu auch 3.6.3 Überprüfung der Visualisierbarkeit, S. 83.
[82] Vgl. Beck (Hrsg.): Wiki der Informationswissenschaft, Klassifikation.
[83] Siehe dazu auch 3.5 Einbindung der halbautomatischen Klassifikation in die Navigator-Anwendung anhand der ausgewählten Beispiele, S. 69.

weiter entwickelt. Seit CSS3 wird die Stylesheet-Sprache durch Module erweitert und definiert. Die einzelnen Module werden weiterhin durch das W3C standardisiert. [84]

2.4.1.2 CSS-Syntax

Der Aufbau von CSS-Anweisungen erfolgt nach dem in Abb. 8 gezeigten Schema.[85]

```
Selektor
{
  Eigenschaft: Wert;
}

/* Kommentar */
```

Abb. 8: Struktur von CSS-Anweisungen

Im Folgenden wird ausschließlich auf die CSS-Selektoren eingegangen, da diese neben CSS ebenfalls für die Auswahl von Knoten in JavaScript, im JavaScript-Framework jQuery sowie in Node.js relevant sind.[86]

2.4.1.3 CSS-Selektoren

Der Standard für CSS3-Selektoren wurden im September 2011 vom W3C veröffentlicht. Grundsätzlich werden drei Arten von Selektoren unterschieden: einfache Selektoren, Pseudo-Elemente und Kombinatoren. Es erfolgt keine Unterscheidung zwischen Groß- und Kleinschreibung. [87]

a) Einfache Selektoren

Die einfachen Selektoren können nochmals unterteilt werden in Typselektor (`elementname`), Universalselektor (`*`), Attributselektoren (`[attributename]`; die Auswahl von Teilwerten ist möglich), Klassenselektor (`.classname`), ID-Selektor (`#elementID`) und Pseudoklassen. [88]

Der Typselektor spricht HTML-Elemente direkt mit deren Elementnamen an. Dieser Selektor kann auch Elemente in Abhängigkeit ihrer Namensräumen auswählen. Hierfür wird der gewünschte Namensraum, gefolgt von einem senkrechten Strich, vor dem Elementnamen angegeben. Mittels Sternchen (`*`) können alle Elemente, sowohl mit als auch ohne Namensraum, angesprochen werden. Wird kein Namensraum, aber der senkrechte Strich angegeben, werden nur Elemente ohne Namensraum ausgewählt. Wird für Selektoren kein voreingestell-

---

[84] Vgl. W3C (Hrsg.): What is CSS?; vgl. auch Wolf: HTML5 und CSS3, S. 41 und 301; vgl. auch Flanagan: JavaScript, S. 413 und 416.
[85] Vgl. Wolf: HTML5 und CSS3, S. 304 und 306.
[86] Siehe dazu auch 2.4.2.4 JavaScript-Selektoren, S. 41 und 2.4.3.5a) cheerio, S. 48.
[87] Vgl. CSS3 Selektoren Spezifikation, 3. Case sensitivity und 4. Selector syntax.
[88] Vgl. Wolf: HTML5 und CSS3, S. 323; vgl. auch CSS3 Selektoren Spezifikation, 4. Selector syntax.

ter Namensraum definiert, entspricht der einfache Typselektor dem Typselektor mit Sternchen. [89]

Der Universalselektor wird durch ein Sternchen repräsentiert. Er spricht jedes einzelne Element des HTML-Dokuments an. Ebenso wie der Typselektor kann auch der Universalselektor auf Elemente in Abhängigkeit ihrer Namensräume zugreifen. [90]

Attributselektoren greifen auf HTML-Elemente über bestimmte Attribute zu. Dabei kann sowohl überprüft werden, ob ein Attribut vorhanden ist, als auch der Wert eines Attributs abgefragt werden. Seit CSS3 kann darüber hinaus die Übereinstimmung von Teilwerten getestet werden. Tabelle 4 zeigt die verschiedenen Möglichkeiten des Attributselektors. [91]

Selektor	Auswahl
`[att]`	alle Elemente, denen ein Attribut mit dem Namen 'att' zugeordnet ist
`[att=value]`	alle Elemente, deren Attribut 'att' ausschließlich den Wert 'value' hat
`[att~=value]`	alle Elemente, deren Attribut 'att' den exakten Wert 'value' beinhaltet (dem `@att`-Attribut können weitere durch Leerzeichen getrennte Attributwerte zugeordnet sein)
`[att|=value]`	alle Elemente mit dem Attribut 'att', welches ausschließlich den Wert 'value' beinhaltet oder dessen Attributwert mit der Zeichenkette 'value-' beginnt
`[att^=value]`	alle Elemente mit dem Attribut 'att', dessen Attributwert mit der Zeichenfolge 'value' beginnt
`[att$=value]`	alle Elemente mit dem Attribut 'att', dessen Attributwert mit der Zeichenfolge 'value' endet
`[att*=value]`	alle Elemente mit dem Attribut 'att', dessen Attributwert die Zeichenfolge 'value' an einer beliebigen Stelle enthält

Tabelle 4: Übersicht über die CSS-Attributselektoren[92]

Auch Attributselektoren können in Abhängigkeit zu Namensräumen definiert werden.[93]

Sowohl der Klassenselektor als auch der ID-Selektor sind spezielle Formen des Attributselektors. Sie wählen Elemente anhand des `@class`- bzw. `@id`-Attributs aus. In beiden Fällen wird der Attributwert angegeben. Die Selektoren können mit dem Typselektor kombiniert werden. Wird vor dem Klassenselektor kein Typselektor angegeben, entspricht dies der Verwendung der beiden Selektoren in Kombination mit dem Universalselektor. Durch die Aneinanderreihung mehrerer Klassenselektoren können Elemente angesprochen werden, denen alle entsprechende Klassen zugewiesen sind. ID-Selektoren könnten theoretisch ebenfalls mit Klas-

[89] Vgl. CSS3 Selektoren Spezifikation, 6.1 Type selector; vgl. auch Wolf: HTML5 und CSS3, S. 324.
[90] Vgl. CSS3 Selektoren Spezifikation, 6.2 Universal selector; vgl. auch Wolf: HTML5 und CSS3, S. 333.
[91] Vgl. Wolf: HTML5 und CSS3, S. 336f; vgl. auch CSS3 Selektoren Spezifikation, 6.3 Attribute selectors.
[92] Vgl. Wolf: HTML5 und CSS3, S. 342 und 345; vgl. auch CSS3 Selektoren Spezifikation, 6.3.1. Attribute presence and value selectors und 6.3.2. Substring matching attribute selectors.
[93] Vgl. CSS3 Selektoren Spezifikation, 6. 3.3. Attribute selectors and namespaces.

senselektoren verbunden werden. Dies ist jedoch praktisch nicht sinnvoll, weil IDs in einem HTML-Dokument immer einmalig sind. [94]

Pseudoklassen wurden als Selektoren eingeführt, um die Auswahl von Elementen basierend auf Informationen, welche außerhalb des Dokumentenbaums liegen, zu ermöglichen. Solche Informationen sind bspw. Elemente, über denen sich der Mauszeiger gerade befindet, oder bereits besuchte bzw. noch nicht ausgewählte Links. Des Weiteren ermöglichen diese Selektoren bspw. die Auswahl von Kind-Elementen nach strukturellen Kriterien oder die Negation von Kriterien mittels `:not()`. Jede Pseudoklasse wird mittels Doppelpunkt, gefolgt vom Namen der Pseudoklasse, aufgerufen. Pseudoklassen können mit allen genannten Selektoren verbunden werden. [95]

b) Pseudoelemente

Durch den Einsatz von Pseudoelementen können bestimmte Elementinhalte (die erste Zeile oder der erste Buchstabe eines Elements) angesprochen, sowie Inhalte vor oder nach einem Element generiert werden. Beide Gruppen gewähren den Zugriff auf Positionen, die nicht direkt als strukturelle Elemente im HTML-Dokumentenbaum vorhanden sind. Ihr Aufruf erfolgt über einen doppelten Doppelpunkt (`::`), gefolgt vom Namen des Pseudoelements. Sie können mit einfachen Selektoren, jedoch nicht mit anderen Pseudoelementen verbunden werden. [96]

c) Kombinatoren

Ein Kombinator ist "ein Zeichen zwischen zwei Selektoren, das diese […] miteinander verkettet"[97]. Tabelle 5 beschreibt die Beziehungen zweier Selektoren, die mit Kombinatoren angesprochen werden können. Der erste Selektor ist hierbei eine Bedingung und der zweite Selektor gibt das Ziel an, welches ausgewählt werden soll. Innerhalb eines CSS-Selektors können beliebig viele einfache Selektoren (ggf. mit Pseudoelementen) durch Kombinatoren ineinander verschachtelt werden. [98]

---

[94] Vgl. Wolf: HTML5 und CSS3, S. 327f, 330 und 332; vgl. auch CSS3 Selektoren Spezifikation, 6.4 Class selectors und 6.5. ID selectors.
[95] Vgl. Wolf: HTML5 und CSS3, S. 345 und 356-358; vgl. auch CSS3 Selektoren Spezifikation, 6.6. Pseudo-classes.
[96] Vgl. Wolf: HTML5 und CSS3, S. 358; vgl. auch CSS3 Selektoren Spezifikation, 7. Pseudo-elements.
[97] Wolf: HTML5 und CSS3, S. 361.
[98] Vgl. CSS3 Selektoren Spezifikation, 8. Combinators; vgl. auch Wolf: HTML5 und CSS3, S. 361.

Kombinator	Beschreibung
B_Z	Das Element Z wird ausgewählt, wenn es ein beliebiger Nachfahre des Element B ist.
B > Z	Das Element Z wird ausgewählt, wenn es ein direkter Nachfahre des Element B ist.
B + Z	Das Element Z wird ausgewählt, wenn es ein direkter Nachbar des Element B ist. Beide Elemente haben einen gemeinsamen Elternknoten.
B ~ Z	Das Element Z wird ausgewählt, wenn es ein beliebiger Nachbar des Element B ist. Beide Elemente haben einen gemeinsamen Elternknoten.

Tabelle 5: Übersicht über die CSS-Kombinatoren[99]

2.4.2 JavaScript

2.4.2.1 Versionsgeschichte

JavaScript ist eine von Netscape (später Mozilla) entwickelte Programmiersprache für clientseitige Webanwendungen. Sie wurde 1995 ursprünglich als LiveScript vorgestellt. Die Anlehnung der Syntax an JAVA führte jedoch in Kombination mit Überlegungen zum Marketing dazu, dass die Sprache ihren jetzigen Namen erhalten hat. JavaScript wurde erstmals 1997 als ECMA[100]-Standard veröffentlicht. [101]

Neben der von Netscape entwickelten JavaScript-Sprache existiert auch eine vergleichbare, von Microsoft entwickelte Skriptsprache mit dem Namen JScript. Daraufhin folgte eine Phase, in der verschiedene Browser-Anbieter insbesondere im Hinblick auf die Skriptunterstützung stark konkurrierten. Das Ergebnis dieser uneinheitlichen Entwicklung ist, dass noch heute für Internet Explorer (vor Version 9) stellenweise ein separater JavaScript-Code erstellt werden muss. [102]

Die Bezeichnung JavaScript ist eine Handelsmarke, deren Sprachspezifikation mit dem ECMAScript als frei verfügbarer Standard veröffentlicht wurde. ECMAScript3 und JavaScript 1.5 sind prinzipiell identisch. Spätere Versionen von JavaScript enthalten Erweiterungen, welche nicht standardisiert sind. Dies kommt dadurch zustande, dass die Versionierung von JavaScript Mozilla unterliegt. 2016 wurde ECMAScript7 (auch ECMAScript2016) veröffentlicht. Die Unterstützung durch die verschiedenen Browser ist derzeit noch dürftig. Die 2009 veröffentlichte Version ECMAScript5 ist die letzte Version, welche von allen modernen Browsern vollständig unterstützt wird. Die neuste JavaScript-Version (1.8) wurde 2008 veröffentlicht. [103] Mit dem

[99] Vgl. Wolf: HTML5 und CSS3, S. 361; vgl. auch CSS3 Selektoren Spezifikation, 8. Combinators.
[100] Kurz für European Computer Manufacturers Association.
[101] Vgl. Wenz: JavaScript, S. 19; vgl. auch Flanagan: JavaScript, S. 1 und 12; vgl. auch W3CSchools (Hrsg.): JS Tutorial, JS Versions.
[102] Vgl. Wenz: JavaScript, S. 19 und 20.
[103] Vgl. Flanagan: JavaScript, S. 2; vgl. auch W3CSchools (Hrsg.): JS Tutorial, JavaScript Versions.

2011 veröffentlichten Patch (JavaScript 1.8.5) wurde u. a. die 'strict mode'-Unterstützung des ECMAScript5 eingebunden. [104]

2.4.2.2 Einbindung in HTML

In HTML-Dokumenten kann JavaScript an verschiedenen Stellen eingebunden werden:

- innerhalb des `<script>`-Elements
- als externes JavaScript-Dokument
- als Event-Handler innerhalb eines HTML-Elements
- als URL, die innerhalb eines Attributes auf das spezielle `javascript:`-Protokoll verweist

Die Einbindung innerhalb eines HTML-Dokuments muss nicht einheitlich erfolgen. [105]

a) `<script>`-Element

JavaScript-Code kann in einem HTML-Dokument innerhalb des `<script>`-Elements erfasst werden. Innerhalb des Elements gelten die JavaScript-Syntaxregeln[106]. Gleichzeitig werden in XHTML-Dokumenten bestimmte Sonderzeichen (bspw. < und &) als XML-Markup interpretiert. Um dies zu umgehen müssen diese Sonderzeichen entweder als HTML-Entitäten (`&lt;` und `&amp;`) umschrieben werden oder der gesamte Bereich innerhalb des `<script>`-Elements wird als `CDATA`[107]-Bereich ausgezeichnet. [108]

Für das `<script>`-Element ist keine feste Position vorgeschrieben. Regulär wird das `<script>`-Element jedoch im `<head>` des HTML-Dokuments platziert. Da Browser den JavaScript-Code sequentiell lesen, ist dies nur möglich, solange der JavaScript-Code nicht auf Elemente zugreift, die zum Zeitpunkt der Ausführung noch nicht erzeugt wurden. [109]

Innerhalb des `<script>`-Elements können auch andere Sprachen eingesetzt werden. XHTML schreibt daher das `@type`-Attribut vor, in dem der Skriptinhalt angegeben wird. Für JavaScript-Code wird dem Attribut der Wert `text/javascript` zugeordnet. In HTML5-Dokumenten ist `text/javascript` der default-Wert des `@type`-Attributs. [110]

b) Externes JavaScript-Dokument

Für die Einbindung von externen JavaScript-Dokumenten wird ebenfalls das `<script>`-Element genutzt. Hierfür wird das Element um das `@src`-Attribut erweitert. Wird dieses Attribut gesetzt, ignorieren die Browser den Text, der innerhalb des Elements steht. Externe

---

[104] Vgl. MDN (Hrsg.): Neu in JavaScript 1.8.5.
[105] Vgl. Wenz: JavaScript, S. 39; vgl. auch Flanagan: JavaScript, S. 311.
[106] Siehe dazu auch 2.4.2.3 Syntax, S. 40.
[107] Character Data: Zeichen innerhalb des CDATA-Bereichs werden nicht als XML-Markup interpretiert.
[108] Vgl. Flanagan: JavaScript, S. 312.
[109] Vgl. Gauchat: HTML5, CSS3 & JavaScript, S. 139.
[110] Vgl. Wenz: JavaScript, S. 42; vgl. auch Flanagan: JavaScript, S. 314.

JavaScript-Dokumente werden regelmäßig erst ausgeführt, wenn das komplette HTML vorliegt. Insbesondere bei umfangreichen HTML-Dokumenten können dadurch Verzögerungen bei der Ausführung des JavaScript-Codes entstehen. Per Konvention werden JavaScript-Dokumente mit der Dateiendung '.js' erstellt. Das `@src`-Attribut gibt die URL zu dem gewünschten JavaScript-Dokument an. [111]

Das `@type`-Attribut kommt bei externen JavaScript-Dokumenten entsprechend der direkten Einbindung von JavaScript-Code zum Einsatz. [112]

Die Nutzung von externen JavaScript-Dokumenten ist die am weitesten verbreitete und von der HTML5-Spezifikation empfohlene Methode zur Einbindung von JavaScript-Code. Sie bringt verschiedene Vorteile mit sich. Einerseits können mehrere HTML-Dokumente auf ein JavaScript-Dokument zurückgreifen. Des Weiteren erfolgt die Trennung von Inhalt und Verhalten der Website, wodurch die Ladezeiten der HTML-Dokumente verbessert werden können. Ein weiterer Vorteil ist der Einsatz von externen JavaScript-Bibliotheken[113] ohne diese auf dem eigenen Server speichern zu müssen. Mit dem CDN 'Google APIs' bietet bspw. Google eine Plattform, auf der die bekanntesten JavaScript-Bibliotheken hinterlegt sind. [114]

Die Nutzung von fremden JavaScript-Code ist nicht durch die Same-Origin-Policy[115] beschränkt. Dennoch gehen mit der Verwendung von fremden Code mehrere Risiken einher. Einerseits ist der Sicherheitsaspekt zu berücksichtigen: fremder Code kann auch unbekannte, schädliche Bestandteile beinhalten. [116] Andererseits besteht ein Risiko im Hinblick auf die Verfügbarkeit des Codes. Wenn die veröffentlichende Seite den Code entfernt oder es zu Ausfällen in deren Netzwerk kommt, kann dies zur Einschränkung der Funktionalität auf der eigenen Seite führen.

Das `<script>`-Element muss mit dem schließenden Tag beendet werden. Die verkürzte Schreibweise als `<script/>` ist nur in XHTML-Dokumenten zulässig. Externe JavaScript-Dokumente enthalten reines JavaScript und werden nicht von einem `<script>`-Element umgeben. [117]

---

[111] Vgl. Wenz: JavaScript, S. 44 und 45; vgl. auch Flanagan: JavaScript, S. 313; vgl. auch Gauchat: HTML5, CSS3 & JavaScript, S. 141.
[112] Vgl. Flanagan: JavaScript, S. 314.
[113] Siehe dazu auch 2.4.2.8 Externe JavaScript Bibliotheken, S. 45.
[114] Vgl. Flanagan: JavaScript, S. 313; vgl. auch Wenz: JavaScript, S. 43 und 45; vgl. auch Gauchat: HTML5, CSS3 & JavaScript, S. 136 und 141.
[115] Sicherheitskonzept für clientseitige Skriptsprachen. Demnach ist es verboten auf fremde Objekte zuzugreifen. Entscheidend für die Kategorisierung als fremd sind die Protokolle der ursprünglichen und der anvisierten Seite, sowie deren Port und der Hoster. Alle Kriterien müssen exakt übereinstimmen. (https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy)
[116] Vgl. Flanagan: JavaScript, S. 314.
[117] Vgl. Flanagan: JavaScript, S. 313.

c) Event-Handler

Events sind Ereignisse, bspw. Mausbewegungen oder Klicks auf Links, auf die JavaScript reagieren kann. Event-Handler verarbeiten die Events, indem sie die mit ihnen verknüpften JavaScript-Funktionen ausführen. Sie beginnen stets mit `on`. Pro Element kann nur ein Event-Handler aktiviert werden. [118]

Event-Handler können zusätzliche Aktionen auslösen, ohne die HTML-interne Funktion des Elements zu beeinträchtigen. Sie können jedoch auch das Standardverhalten von HTML-Elementen verhindern. Das Standardverhalten eines HTML-Links ist bspw. der Aufruf des Linkziels. [119]

Innerhalb eines HTML-Dokuments können Event-Handler als zusätzliches Attribut eines Elements verwendet werden. Als Wert des Attributs wird hierbei eine Funktionsreferenz (entspricht dem Funktionsnamen) angegeben. [120]

Als Alternative hierzu können Event-Handler im JavaScript-Code bestimmten HTML-Elementen zugeordnet werden. Hierfür wird das entsprechende HTML-Element mittels JavaScript-Selektoren ausgewählt. Anschließend wird der Event-Handler als Eigenschaft hinzugefügt. [121]

```
<!DOCTYPE html SYSTEM "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Event-Handler</title>
    <script type="text/javascript">
        window.onload = function(){
          document.getElementById('EHLink').onclick = function() {return false;};
        };
      </script>
  </head>
  <body>
    <a href="https://www.google.de/" onclick="return false;">Inline-JS</a>
    <a href="https://www.google.de/" id="EHLink">Event-Handling als Eigenschaft</a>
  </body>
</html>
```

Quellcode 2: Vergleich der Event-Handler Inline und als Eigenschaft

Quellcode 2 zeigt beide Varianten des Event-Handlers anhand des '`click`'-Events. Hierbei ist erkennbar, dass die Inline-Variante wesentlich kürzer ist als die Zuweisung mittels JavaScript-Selektor. Die Ergänzung des JavaScript-Codes um das `load`-Event ist notwendig, weil das Skript ansonsten ausgeführt wird, bevor das HTML vollständig aufgebaut ist. Der JavaScript-Selektor würde somit kein entsprechendes Element finden.

Im Rahmen der HTML5-Spezifikation wurde neben der Event-Handler Methode die Event-Listener-Methode als Standard implementiert. Event-Listener werden im JavaScript-Code er-

---

[118] Vgl. Wenz: JavaScript, S. 47; vgl. auch Gauchat: HTML5, CSS3 & JavaScript, S. 144; vgl. auch Flanagan: JavaScript, S. 445.
[119] Vgl. Wenz: JavaScript, S. 47 und 105.
[120] Vgl. Wenz: JavaScript, S. 47 und 102; vgl. auch Gauchat: HTML5, CSS3 & JavaScript, S. 144f.
[121] Vgl. Gauchat: HTML5, CSS3 & JavaScript, S. 145.

stellt. Hierfür werden sie mittels '`[element].addEventListener()`' an Objekte angebunden. Diese Methode ist vom W3C empfohlen und wird von den meisten Browsern unterstützt. Für frühere Internet Explorer-Versionen (vor der Versionen 9) müssen Event-Listener mittels '`[element].attachEvent()`' definiert werden. Die Event-Listener-Methode haben gegenüber der Event-Handler-Methode den Vorteil, dass einem Objekt mehrere Event-Listener hinzugefügt werden können. [122]

Damit JavaScript-Code browserunabhängig funktioniert, müssen stets beide Event-Listener-Methoden integriert werden. Dabei gibt es einerseits die Variante, alle Browser abzufragen und dementsprechend den Event-Listener auszuwählen, andererseits kann die Unterstützung des Event-Listeners abgefragt werden. Die zweite Variante ermöglicht es dem Entwickler unbekannte Browser, welche dennoch diese Methoden unterstützen, nicht auszuschließen. [123]

```
<!DOCTYPE html SYSTEM "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Event-Listener</title>
    <script type="text/javascript">
        window.onload = function(){
          var myLink = document.getElementById("myLink")

          if (myLink.addEventListener)
            //W3C Empfehlung
            myLink.addEventListener("click", function(event){
              event.preventDefault(); return false; });
          else if (myLink.attachEvent)
            // IE vor Version 9
            myLink.attachEvent("onclick", function(event){
              event.returnValue = false;});
        }
      </script>
  </head>
  <body>
    <a href="https://www.google.de/" id="myLink">Event-Listener</a>
  </body>
</html>
```

Quellcode 3: Abfrage der unterstützen Event-Listener-Methode

Das Quellcode 3 zeigt eine solche Abfrage nach der unterstützten Event-Listener-Methode. Das Event-Handler-Beispiel, bei dem ein HTML-Linkelement sein Standardverhalten nicht durchführt, wird hierbei wieder aufgegriffen.

d) `javascript:`-Protokoll

Eine weitere Variante um JavaScript in HTML-Dokumenten einzubinden, ist die Verwendung des speziellen `javascript:`-Protokolls. Diese Methode kann nur innerhalb von URLs[124] genutzt werden. Der darauf folgende JavaScript-Code wird von Browsern als eine Zeile

---

[122] Vgl. Wenz: JavaScript, S. 103f und 109; vgl. auch Gauchat: HTML5, CSS3 & JavaScript, S. 145.
[123] Vgl. Wenz: JavaScript, S. 114; vgl. auch Flanagan: JavaScript, S. 459.
[124] Kurz für Uniform Resource Locator.

interpretiert. In Browsern, die kein JavaScript unterstützen, führt diese Methode zu Fehlermeldungen. Der Einsatz des `javascript:`-Protokolls ist nicht mehr gebräuchlich. [125]

2.4.2.3 Syntax

a) Trennung von Anweisungen

Anweisungen werden in JavaScript mit einem Semikolon getrennt. In einigen Fällen genügt ein Zeilenumbruch zur Trennung der Anweisung. [126] Zur einheitlichen Gestaltung des Codes und zur Erhöhung des Verständnisses durch Dritte, sollten immer Semikolons zur Trennung der Anweisungen genutzt werden.

b) Namenskonventionen

JavaScript ist eine 'case-sensitive language', d. h. die Groß- und Kleinschreibung der Namen wird unterschieden. Namen identifizieren Variablen und Funktionen. Sie können aus Buchstaben, Ziffern und dem Unterstrich (_) bestehen. Das Dollar-Zeichen ($) ist auch möglich, aber es ist in der Regel für JavaScript-Frameworks reserviert. Der Name einer Variable darf nicht mit einer Ziffer beginnen. Neben den genannten Beschränkungen dürfen Schlüsselwörter der Sprache (bspw. `if`, `function` oder `while`) ebenfalls nicht als Name ausgewählt werden. [127]
Leerzeichen und Bindestriche sind als Name nicht erlaubt, weil der JavaScript-Parser bspw. den Bindestrich als Subtraktionsoperator interpretiert.[128] Daher sollten Binnenmajuskel (die sog. CamelCase-Schreibweise) oder Unterstriche für Variablen genutzt werden, deren Bezeichnung mehrere Wörter umfasst. Die CamelCase-Schreibweise bzw. Trennung der Worte mittels Unterstrich erhöht die Lesbarkeit des Codes.

c) Kommentare

In JavaScript gibt es zwei Möglichkeiten Kommentare zu erfassen. Einzeilige Kommentare werden mit einem doppelten Schrägstrich (`//`) gekennzeichnet. Mit einem Zeilenumbruch endet der kommentierte Bereich. Mehrzeilige Kommentare werden mit einem Schrägstrich und Stern (`/*`) eröffnet und mit einem Stern und Schrägstrich (`*/`) geschlossen. Kommentare dürfen nicht ineinander verschachtelt sein. Der Inhalt von Kommentaren wird von den JavaScript-Interpretern der verschiedenen Browser nicht ausgeführt. [129]

---

[125] Vgl. Flanagan: JavaScript, S. 315; vgl. auch Wenz: JavaScript, S. 46f.
[126] Vgl. Flanagan: JavaScript, S. 25.
[127] Vgl. Flanagan: JavaScript, S. 21 und 23-24; vgl. auch Wenz: JavaScript, S. 49 und 50.
[128] Vgl. Resig, Bibeault: Geheimnisse eine JavaScript-Ninjas, S. 355; vgl. auch SELFHTML (Hrsg.): JavaScript, Bezeichner.
[129] Vgl. Flanagan: JavaScript, S. 23; vgl. auch Wenz: JavaScript [2], S. 56.

2.4.2.4 JavaScript-Selektoren

Der Zugriff auf einzelne HTML-Dokumente kann grundsätzlich über zwei Wege erfolgen: über den Typ eines Elements[130] oder über das Document Object Model (i. F. kurz: DOM). Das DOM bildet alle Elemente eines HTML-Dokuments als Knoten ab. Jeder Knoten kann dabei den NodeType Tag, Attribut oder Text annehmen. Der NodeName beschreibt den HTML-Tag des Knotens als Zeichenkette. JavaScript kann die einzelnen Knoten nicht nur bearbeiten, sondern auch neue Knoten hinzufügen oder bestehende Knoten entfernen. [131]

Die Knoten des DOM können über verschiedene Eigenschaften ausgewählt werden:[132]

- `@id`-Attribut (`getElementById`)
- Elementnamen (`getElementsByTagName`)
- `@name`-Attribut[133] (`getElementsByName`)
- `@class`-Attribut (`getElementsByClassName`)
- mittels CSS-Selektoren[134] (`querySelectorAll`/`querySelector`)

Beginnt die Methode mit dem Plural (`getElements`), dann gibt diese Methode ein Array mit allen entsprechenden Knoten zurück. Diese Arrays können nur gelesen, aber nicht verändert werden. Jedem Wert des Arrays ist ein Index zugeordnet, über den der Zugriff auf die einzelnen Knoten erfolgen kann. Wird das HTML-Dokument verändert, aktualisiert sich das Array und neue Knoten werden hinzugefügt, bzw. nicht mehr vorhandene Knoten entfernt. Innerhalb des DOM können zusätzlich Eltern-, Kind- und Geschwister-Knoten angesprochen werden. [135]

Die Auswahl mit CSS-Selektoren ermöglicht eine präzisere Auswahl als die reinen JavaScript-Selektoren. Anders als bei den JavaScript-Methoden, wird das Array nicht aktualisiert, wenn sich das Dokument verändert. Während `querySelectorAll` alle zutreffenden Knoten in einem Array zurückgibt, wählt `querySelector` nur den ersten passenden Knoten aus. [136]

Bevor die Knoten des DOM angesprochen werden, muss grundsätzlich das Dokument selbst als Objekt angesprochen werden. Dies erfolgt über die globale Variable `document`. [137]

2.4.2.5 Variablen

Variablen werden genutzt um Daten während der Programmierung vorübergehend zu speichern. [138]

---

[130] Diese Methode wird vernachlässigt, da sie nur für bestimmte Elemente (bspw. `<img>`) einsetzbar ist.
[131] Vgl. Wenz: JavaScript, S. 121 und 124f.
[132] Vgl. Wenz: JavaScript, S. 126 - 130; vgl. auch Flanagan: JavaScript, S. 364 - 370.
[133] Dieses Attribut ist nur für bestimmte HTML-Elemente zulässig, bspw. Formulare.
[134] Siehe dazu auch 2.4.1.3 CSS-Selektoren, S. 32.
[135] Vgl. Wenz: JavaScript, S. 130; vgl. auch Flanagan: JavaScript, S. 367 und 371.
[136] Vgl. Flanagan: JavaScript, S. 370.
[137] Vgl. Flanagan: JavaScript, S. 308.
[138] Vgl. Wenz: JavaScript, S. 49.

Die Erklärung einer Variable erfolgt über den Aufruf `var`. Anschließend wird der Name der Variable bestimmt. Für die Auswahl des Namens gelten die Namenskonventionen von JavaScript[139]. Mit dem Gleichheitszeichen (=) kann der Variable ein Wert zugeordnet werden. Die Initialisierung der Variable kann auch zu einem späteren Zeitpunkt erfolgen. Variablen können als ein Teil von Schleifen definiert werden. [140]

Grundsätzlich ist zwischen globalen und lokalen Variablen zu unterscheiden. Globale Variablen gelten im gesamten JavaScript-Code, lokale Variablen nur innerhalb von Funktionen, in denen sie definiert werden. Parameter einer Funktion gelten ebenfalls als lokale Variablen. [141]

Es gibt nummerische Variablen, Zeichenketten als Variable und Boolesche Variablen. Diese Typen können während der Verwendung der Variablen verändert werden. [142]

2.4.2.6 Funktionen

Funktionen sind ein "Programmblock, der nicht sofort ausgeführt wird, aber explizit auf- bzw. abgerufen werden kann"[143]. Sie können innerhalb eines JavaScript-Codes mehrfach verwendet werden. Funktionen können sowohl Befehle ausführen, als auch Werte zurückgeben. Neben Funktionen, die von JavaScript definiert sind, besteht die Möglichkeit eigene Funktionen zu erstellen. Die Grundstruktur ist in beiden Fällen identisch. [144]

```
1  function Name (ParameterN){
2    //Definition lokaler Variablen
3    //Befehle, die ausgeführt werden sollen
4    return Wert;
5  }
6
```

Quellcode 4: Grundstruktur einer JavaScript-Funktion

Für die Auswahl des Namens gelten die Namenskonventionen von JavaScript[145]. Einmalig verwendete Funktionen müssen keinen Namen erhalten, wenn sie an der Stelle definiert werden, an welcher sie ausgeführt werden. Diese Funktionen werden als anonyme Funktionen bezeichnet. [146]

Die Anzahl der Parameter kann beliebig hoch sein, auch Funktionen ohne Parameter sind möglich. Parameter verhalten sich innerhalb der Funktion wie lokale Variablen. Echte lokale Variablen können an dieser Stelle definiert werden. Der Zugriff auf globale Variablen ist wei-

---

[139] Siehe dazu auch 2.4.2.3b) Namenskonventionen, S. 40.
[140] Vgl. Wenz: JavaScript, S. 49-50; vgl. auch Flanagan: JavaScript, S. 52.
[141] Vgl. Flanagan: JavaScript, S. 53.
[142] Vgl. Wenz: JavaScript, S. 50-51 und 58.
[143] Wenz: JavaScript, S. 72.
[144] Vgl. Flanagan: JavaScript, S. 163; vgl. auch Wenz: JavaScript, S. 72.
[145] Siehe dazu auch 2.4.2.3b) Namenskonventionen, S. 40.
[146] Vgl. Wenz: JavaScript, S. 75; vgl. auch Flanagan: JavaScript, S. 165.

terhin möglich. Funktionen können ineinander verschachtelt werden. Dies ermöglicht es, Werte von Parametern und Variablen an andere Funktionen weiter zu geben, bzw. die Rückgabewerte anderer Funktionen zu verarbeiten. Der Einsatz des Befehls 'return' ist optional. Der Wert nach 'return' ist der Rückgabewert der Funktion und ebenfalls optional. Der Befehl 'return' beendet die Funktion und kehrt im JavaScript-Code zu der Stelle zurück, an welcher die Funktion aufgerufen wurde. [147]

Der Aufruf von Funktionen erfolgt an beliebiger Stelle über den Namen der Funktion und ist somit bereits vor der Definition der Funktion möglich. Sind für eine Funktion Parameter definiert, werden die entsprechenden Werte beim Aufruf übergeben. [148]

2.4.2.7 Arrays

Ein Array ist eine geordnete Sammlung von Werten, ein Datenfeld. Innerhalb des Arrays wird jedem Wert ein Index zugeordnet, welcher die nummerische Position angibt. Dabei erhält der erste Wert den Index '0'. Der numerische Index kann in eine Zeichenkette umgewandelt werden. Dadurch entsteht ein assoziatives Array. [149]

Innerhalb eines Arrays können verschiedene Typen von Werten sowie andere Arrays erfasst werden. Dadurch ist einem Array selbst kein fester Typ zugeordnet. Die einzelnen Werte des Arrays bestimmen die Länge des Arrays, welche jedoch dynamisch ist. Jederzeit können Werte hinzugefügt oder entfernt werden. Daher können auch leere Arrays definiert werden. Die aktuelle Länge des Arrays kann mittels `.length` abgefragt werden. [150]

Die Definition eines Arrays erfolgt als Variable. Hierfür kann der Befehl '`new Array ()`' genutzt werden. In der verkürzten Schreibweise wird ein Array mit eckigen Klammern (Quellcode 5, Zeile 2) eröffnet. Die verkürzte Schreibweise entspricht dem JSON-Format. [151]

---

[147] Vgl. Wenz: JavaScript, S.72 und 75; vgl. auch Flanagan: JavaScript, S. 164 und 166.
[148] Vgl. Flanagan: JavaScript, S. 167.
[149] Vgl. Flanagan: JavaScript, S. 141; vgl. auch Wenz: JavaScript, S. 71; vgl. auch SELFHTML (Hrsg.): JavaScript/ Array/ Assoziative Arrays.
[150] Vgl. Flanagan: JavaScript, S. 141 und 144; vgl. auch Wenz: JavaScript, S. 138 und 139.
[151] Vgl. Flanagan: JavaScript, S. 141 und 142; vgl. auch Wenz: JavaScript, S. 71 und 138.

```
1  // Leeres Array a erstellen
2  var a = [];    // Länge: 0
3
4  // Zuordnung von 3 Werten zum Index 0 bis 2
5  a = [wert_1, wert_2, wert_3]; // Länge: 3
6
7  // neuer Index mit Wert
8  a[5] = wert_6; // Länge: 6
9        // a = [wert_1, wert_2, wert_3, leer, leer, wert_6]
10
11 // Länge des Arrays neu definieren
12 a.length = 4;  // Länge: 4
13       // a = [wert_1, wert_2, wert_3, leer]
```

Quellcode 5: Methoden zur Veränderung eines Arrays

Es gibt verschiedene Methoden zur Veränderung eines Arrays. Durch Angabe eines neuen Index und gleichzeitiger Zuordnung eines Wertes (Quellcode 5, Zeile 8), kann ein Array verlängert werden. Der neue Index muss nicht auf die erste freie Position im Array gesetzt werden. Die neue Länge ist in jeden Fall die angegebene Indexzahl erhöht um 1, auch wenn das Array hierdurch Indizes ohne Wert enthält. Wird die Länge des Arrays neu definiert (Quellcode 5, Zeile 12), entfallen alle Werte, die den gekürzten Indizes zugeordnet sind. Mittels '`delete a[n]`' können einzelne Werte aus dem Array entfernt werden, ohne dass die Länge des Arrays verändert wird. '`n`' gibt dabei den Index des zu löschenden Wertes an. [152]

Mit `.push`, `.pop` und `.splice` bietet JavaScript Funktionen um Werte zu einem Array hinzuzufügen oder aus diesem zu entfernen. Beliebig viele Werte, aber mindestens ein Wert, können mittels `.push()` am Ende des Arrays hinzugefügt werden. Rückgabewert dieser Funktion ist die neue Länge des Arrays. Um beliebig viele Werte, aber mindestens ein Wert, vom Ende des Arrays zu entfernen und gleichzeitig die Länge des Arrays zu verkürzen, kann `.pop()` genutzt werden. Im Gegensatz zur Kürzung des Arrays über die Längeneigenschaft, kann auf die entfernten Werte, als Rückgabewert der Funktion, zugegriffen werden. Als Kombination der beiden Funktionen kann `.splice()` genutzt werden, um gleichzeitig Werte zu entfernen und hinzuzufügen. Hierbei müssen mindestens zwei Parameter übergeben werden. Als erster Parameter wird der Index des Wertes angegeben, ab dem Werte entfernt werden sollen. Die Anzahl der zu entfernenden Werte wird durch den zweiten Parameter bestimmt. Weitere Parameter werden identisch zur `.push`-Funktion am Ende des Arrays hinzugefügt. [153]

In JavaScript können, neben den genannten Funktionen, weitere Funktionen genutzt werden, um bspw. die Werte innerhalb des Arrays zu sortieren. Auch ECMAScript5 enthält zusätzliche,

---

[152] Vgl. Flanagan: JavaScript, S. 144-146; vgl. auch Wenz: JavaScript, S. 138.
[153] Vgl. Flanagan: JavaScript, S. 145f und 151; vgl. auch Wenz: JavaScript, S. 139.

standardisierte Methoden. [154] Auf diese Funktionen kann im Rahmen dieser Arbeit nicht eingegangen werden, weil sie für die Navigator-Anwendung keine oder nur eine untergeordnete Relevanz besitzen.

2.4.2.8 Externe JavaScript Bibliotheken

Bibliotheken ermöglichen es die Funktionen von JavaScript zu erweitern bzw. die Nutzung von JavaScript zu erleichtern. Darüber hinaus sollen sie insbesondere die Entwicklung von browserunabhängigen JavaScript-Anwendungen vereinfachen. Die Bibliotheken sind oftmals als Open Source verfügbar. [155]

Einerseits gibt es Bibliotheken, welche auf ein bestimmtes (Teil-)Thema spezialisiert sind, andererseits sind auch Bibliotheken verfügbar, welche als eine Art allumfassende Funktionssammlung anzusehen sind. In beiden Fällen sind Bibliotheken modular aufgebaut, sodass die Möglichkeit besteht nur benötigte Teile in die Website einzubinden und diese somit nicht mit nicht benötigten Inhalten zu überfüllen. [156]

Die Einbindung der JavaScript-Bibliotheken kann für jede Bibliothek unterschiedlich sein. Während bspw. für die Initialisierung der jQuery-Bibliothek ein Verweis wie auf ein reguläres JavaScript-Dokument erfolgt, muss bspw. für die Angular.js-Bibliothek, neben dem Verweis auf das entsprechende JavaScript-Dokument, auch eine Anpassung des HTML-Wurzelelements erfolgen. [157] Das JavaScript-Framework jQuery ist eine der beliebtesten Bibliotheken. [158]

2.4.3 Node.js

2.4.3.1 Versionen

Node.js ist "im Wesentlichen […] die JavaScript-Engine aus dem Chrome-Browser, aber als allein lauffähiges Programm"[159]. Die Software ist als Download frei verfügbar. Als JavaScript-Interpreter unterstützt Node.js alle JavaScript- und ECMAScript5-Selektoren. Die Anbindung an die Unix API[160] ermöglicht u. a. die Arbeit mit Prozessen und Dateien. [161] Node.js soll die Basis für das Spider-Modul des HTML-Navigators bilden. Damit das Tool die gespiderten HTML-Dokumente lokal speichern kann, ist die Anbindung an die Unix-API essentiell.

---

[154] Vgl. Flanagan: JavaScript, S. 149-157; vgl. auch Wenz: JavaScript, S. 140-143.
[155] Gauchat: HTML5, CSS3 & JavaScript, S. 151; vgl. auch Wenz: JavaScript, S. 371.
[156] Vgl. Mintert: Hilf, wenn Du kannst/ Her mit den Rich Internet Applications.
[157] Vgl. SELFHTML (Hrsg.): jQuery, Einbindung; vgl. auch Google (Hrsg.): AngularJS, The Basics.
[158] Vgl. Gauchat: HTML5, CSS3 & JavaScript, S. 152; vgl. auch Wenz: JavaScript, S. 371; vgl. auch Resig, Bibeault: Geheimnisse eine JavaScript-Ninjas, S. 23 und 30.
[159] Wenz: JavaScript, S.446.
[160] Unix-Schnittstelle zur Programmierung von Anwendung.
[161] Vgl. Node.js Foundation (Hrsg.): Node.js, Home; vgl. auch Flanagan: JavaScript, S. 296f.

Die aktuellste Version Node.js v7.4 wurde im Oktober 2016 veröffentlicht. Gleichzeitig ist weiterhin die Version 6.9.4 verfügbar, die dem Long Term Support (im Folgenden kurz: LTS) unterliegt. Grundsätzlich gibt es jederzeit eine Version, welche die neusten Features beinhaltet, und maximal zwei Versionen, welche auf Basis des LTS weiterhin gepflegt werden. [162] Entsprechend der Empfehlung von Node.js wird im Rahmen der Arbeit die LTS-Version genutzt.

2.4.3.2 Ausführung des Node-Skripts

Nach der Installation kann Node.js mit dem Befehl 'node' über die Bash Konsole[163] ausgeführt werden. Dem Aufruf muss der Pfad zum gewünschten JavaScript-Dokument hinzugefügt werden. Anschließend können beliebig viele Argumente übergeben werden. Vor dem JavaScript-Dokument können verschiedene Optionen bestimmt werden. [164] Auf die Optionen kann im Rahmen dieser Arbeit nicht eingegangen werden, weil die Möglichkeiten zu umfangreich sind. Alle Bestandteile des Aufrufs in der Konsole speichert Node.js in einem Array. Das Array kann im JavaScript über die globale Variable '`argv`' aufgerufen werden. Für diese und weitere globale Prozessvariablen hat Node.js den Namensraum '`process`' definiert. Den Index '0' des Arrays '`process.argv`' belegt der Befehl node. Alle folgenden Elemente (Optionen, Pfad zum JavaScript-Dokument und Argumente) werden in der angegebenen Reihenfolge als Einzelwerte im Array erfasst. [165] Diese Prozessvariable ermöglicht es, innerhalb des JavaScript-Codes eigene Variablen zu definieren, welche anschließend die gewünschten Werte des Nutzers bspw. über eine Front-End-Pipeline erhalten. [166]

2.4.3.3 Besondere Funktionen

Node.js bietet eine umfangreiche Auswahl an Funktionen. Einige der Funktionen sind bereits durch JavaScript definiert. Bspw. ist mit der JavaScript-Methode '`console.log()`' in Node.js eine Standardausgabefunktion implementiert, welche insbesondere für die Entwicklung und dem damit zusammenhängenden Debugging hilfreich ist. Innerhalb dieser Funktion können beliebig viele Argumente (Zeichenketten, Funktionen, Variablen und Platzhalter) in die Konsole geschrieben werden. Enthält das erste Argument der Funktion keinen Platzhalter, werden alle Argumente der Funktion zu einer Zeichenkette verbunden. Sind Platzhalter im ersten Argu-

---

[162] Vgl. Node.js Foundation (Hrsg.): Node.js, Downloads; vgl. auch Node.js Foundation (Hrsg.): Node.js Long-term Support Working Group; LTS Plan.
[163] Siehe dazu auch 2.5.2 Bash Konsole, S. 57.
[164] Vgl. Node.js Documentation: Usage & Example; vgl. auch Flanagan: JavaScript, S. 296.
[165] Vgl. Node.js Documentation: Process, Stichwort process.argv; vgl. auch Hughes-Croucher, Wilson: Node, 5. Helper API's/ Processes.
[166] Siehe dazu auch 3.6.1.6 Ausgaben des Spider-Moduls, S. 78.

ment enthalten, werden die folgenden Argumente in der Reihenfolge, in der sie aufgelistet sind, an die Stellen der Platzhalter eingefügt. [167]

Die Einbindung von lokal gespeicherten Modulen erfolgt über Variablen, welche die spezielle Node.js-Funktion `require()` enthalten. Als Argument erhält diese Funktion den Namen des gewünschten Moduls. Bei der anschließenden Ausführung des JavaScript-Codes mit Node.js wird das benannte Modul geladen und einmalig ausgeführt. Der Rückgabewert der Funktion ist ein Objekt, das alle Formeln und Variablen beinhaltet. Im folgenden JavaScript-Code können diese anschließend verwendet werden. Mit Modulen kann die Node.js-Plattform individuell erweitert werden. Beim Download von Node.js wird für die Verwaltung der Module der Paketmanager npm mit ausgeliefert. Module enthalten bspw. Funktionen und Variablen, die dazu bestimmt sind, in anderen JavaScript-Dokumenten aufgerufen und verwendet zu werden. [168]

2.4.3.4 Installation von Node.js-Modulen

Bevor Module in einem JavaScript-Dokument eingebunden werden können, müssen diese installiert werden. Hierfür können sie einzeln mittels '`npm install <…>`' aufgerufen werden. Insbesondere bei der Installation mehrerer Module bietet sich die Nutzung einer Paketdatei an. Diese wird im JSON-Format[169] erstellt. Der Konvention folgend, wird das Dokument unter dem Namen '`package.json`' gespeichert. [170]

package.json

```
{
  "name": "html-spider",
  "version": "0.0.0",
  "dependencies": {
    "cheerio": "git://github.com/cheeriojs/cheerio.git",
    "url-parse": "git://github.com/unshiftio/url-parse.git",
    "request": "git://github.com/request/request.git",
    "file-system":"git://github.com/douzi8/file-system.git"
  }
}
```

Quellcode 6: Installationspaket für Node.js-Module

Mit der Angabe eines Namens und einer Version wird im Quellcode 6 die Mindestanforderung an eine Paketdatei erfüllt. Zusätzlich werden im Objekt `dependencies` Abhängigkeiten zu vier Modulen angegeben. Im Beispiel erfolgt der Verweis auf die entsprechenden Git-URLs. Die Angabe einer Versionsnummer, einer allgemeinen URL oder einer GitHub-URL ist ebenso möglich. Die Paketdatei wird über die Konsole installiert, indem der Befehl 'npm install' ausgeführt wird.

---

[167] Vgl. Node.js Documentation: Console/ console.log([data][, …args]); vgl. auch Flanagan: JavaScript, S. 296.
[168] Vgl. Flanagan: JavaScript, S. 296; vgl. auch Node.js Documentation: Modules; vgl. auch Hughes-Croucher, Wilson: Node, 8. Extending Node/ Modules und Package Manger.
[169] Siehe dazu auch 2.4.4.3 JavaScript Object N, S. 52.
[170] Vgl. npm Documentation: Getting Started/ Using a package.json.

Hierbei sucht der npm im aktuellen Verzeichnis nach einer 'package.json', erstellt ein neues Unterverzeichnis 'node_modules' und lädt alle Module in der entsprechenden Version in das neue Verzeichnis herunter. [171]

2.4.3.5 Node.js-Module zur Entwicklung des Spider-Moduls

In den folgenden Abschnitten werden vier Module vorgestellt, die im Rahmen des HTML-Navigators für die Erstellung des Spider-Moduls genutzt werden sollen.

a) cheerio

Das cheerio-Modul ist eine Implementierung des jQuery[172]-Kerns, sodass die vereinfachten jQuery-Methoden zur Auswahl von Elementen genutzt werden können. Es wurde 2012 von Matt Mueller unter der MIT-Lizenz (Open Source) veröffentlicht. Die aktuelle Version 0.22.0 ist seit August 2016 auf GitHub verfügbar. [173]

Die jQuery-Bibliothek zeichnet sich insbesondere durch die `$`-Funktion aus. Innerhalb dieser Funktion werden CSS-Selektoren genutzt. Diese Variante verkürzt den JavaScript-Code wesentlich. [174] Bevor der Zugriff auf die HTML-Elemente erfolgen kann, muss das Dokument geladen werden. Dieser vorbereitende Schritt ist in jQuery implementiert. Das cheerio-Modul hat diese Implementierung nicht übernommen. Stattdessen liefert es hierfür die Funktion `load()`. Als Argument wird das gewünschte Element als Zeichenkette übergeben. Das Modul sieht vor diese Funktion als Variable `$` zu definieren. Anschließend kann somit die Verarbeitung der DOM-Elemente ähnlich zu jQuery mit der `$`-Funktion erfolgen. [175]

Für den Zugriff und die Manipulation der DOM-Element enthält das cheerio-Modul vielfältige Methoden. Tabelle 6 soll daher nur die Methoden beschreiben, welche für die Erstellung des Spider-Moduls verwendet werden.

Methode	Beschreibung
`.attr(name, value)`	Auswahl des ersten Attributes, dessen Name mit dem Argument `name` übereinstimmt. Dem gefundenen Attribut wird ein Wert zugewiesen, wenn das Argument `value` bestimmt wird. Beide Argumente werden als Zeichenkette übergeben.
`.each(function())`	Iteriert über ein cheerio-Objekt und führt für jedes passende Element die angegebene Funktion aus.
`this`	Übergibt das aktuelle DOM-Element an die Funktion.

Tabelle 6: Übersicht über ausgewählte cheerio-Funktionen zur DOM-Manipulation [176]

[171] Vgl. npm Documentation: Getting Started/ Using a package.json, Installing npm packages locally und Configuring npm/ package.json.
[172] Die jQuery-Bibliothek soll den Zugriff auf Elemente und deren Manipulation innerhalb von JavaScript-Dokumenten vereinfachen. (http://www.jquery.org)
[173] Vgl. Mueller: Cheerio; vgl. auch Mueller: Cheerio GitHub Repository, History.
[174] Vgl. Wenz: JavaScript, S. 373.
[175] Vgl. Mueller: Cheerio, Loading.
[176] Vgl. Mueller: Cheerio, Attributes und Traversing.

b) url-parse

Das url-parse-Modul transformiert eine URL von einer Zeichenkette in ein Objekt, welches die Bestandteile der URL als separate Werte enthält. Es wurde 2015 von Unshift.io und Arnout Kazemir unter der MIT-Lizenz (Open Source) veröffentlicht. Die aktuelle Version 1.1.7 ist seit Oktober 2016 auf GitHub verfügbar. [177]

Die zentrale Funktion des url-parse-Moduls ist `URL()`. Als Argument erhält diese Funktion eine absolute oder relative URL. Quellcode 7 zeigt am Beispiel der URL http://www.zeit.de/index den Rückgabewert der Funktion als ein Objekt mit den Bestandteilen der URL. [178]

```
URL {
  slashes: true,
  protocol: 'http:',
  hash: '',
  query: '',
  pathname: '/index',
  auth: '',
  host: 'www.zeit.de',
  port: '',
  hostname: 'www.zeit.de',
  password: '',
  username: '',
  origin: 'http://www.zeit.de',
  href: 'http://www.zeit.de/index' }
```

Quellcode 7: Rückgabewert des url-parse-Moduls

c) request

Das request-Modul soll HTTP-Anfragen so einfach möglich gestalten. Es wurde 2011 von Mikael Rogers unter der Apache-2.0-Lizenz (Open Source) veröffentlicht. Die aktuelle Version 2.79.1 ist seit November 2016 auf GitHub verfügbar. [179]

Im Hintergrund greift das request-Modul auf das Node.js-Objekt '`http.IncomingMessage`' zurück. Dieses Objekt enthält verschiedene Daten zur HTTP-Anfrage, bspw. den Statuscode. Eine HTTP-Anfrage wird mit der Funktion `request()` ausgelöst. Als erstes Argument erhält sie eine URL. Eine callback-Funktion wird als zweites Argument übergeben. Die callback-Funktion enthält wiederum drei Parameter:

- error: Tritt kein Fehler auf, ist der Wert null. Andernfalls enthält error entsprechende Informationen zum aufgetretenen Fehler.
- response: Mit diesem Parameter erfolgt der Zugriff auf das `http.IncomingMessage`-Objekt.
- body: Eine erfolgreiche HTTP-Anfrage gibt als Antwort einen body zurück, der den Inhalt des angefragten Dokuments in der ursprünglichen Syntax enthält. [180]

---

[177] Vgl. npm (Hrsg.): url-parse, Usage; vgl. auch unshift.io (Hrsg.): url-parse, License und Releases.
[178] Vgl. npm (Hrsg.): url-parse, Usage.
[179] Vgl. npm (Hrsg.): request; vgl. auch Rogers: request, License.
[180] Vgl. npm (Hrsg.): request, Super simple to use; vgl. auch Node.js Documentation: HTTP/ Class:http.IncomingMessage.

Das Spider-Modul soll das request-Modul nutzen, um den Statuscode der anvisierten Website abzufragen. Nur wenn als Status 'OK' (Statuscode 200) zurückgegeben wird, sollen weitere Schritte folgen.

d) file-system

Node.js enthält bereits diverse file-system-Funktionen, die als Modul '`fs`' integriert sind. Das file-system-Modul will diese nicht ersetzen, sondern deren Fähigkeiten stärken. Der Kern des Moduls ist es, die Schnittstelle zur Dateiverwaltung zu vereinfach, indem nicht geprüft werden muss, ob das angestrebte Verzeichnis besteht. Es wurde 2014 unter der ISC-Lizenz (Open Source) veröffentlicht. Die aktuelle Version 2.2.2 wurde zuletzt im November 2016 auf GitHub aktualisiert. [181]

Im Rahmen der Navigator wird die Schnittstelle zur Dateiverwaltung genutzt, um die gespiderten HTML-Dokumente zu speichern. [182]

2.4.4 Weitere Web-Technologien

Im Folgenden werden verschiedene Web-Technologien, welche im Navigator für die Visualisierung eingesetzt werden, umrissen.

2.4.4.1 Ajax

Der Kern von Ajax[183] ist, dass sich die Inhalte einer Seite verändern, ohne dass die Seite neu geladen wird. Hierfür werden im Hintergrund Daten vom Server angefragt und erhalten, auch nachdem die Seite bereits geladen wurde. Des Weiteren können, ebenfalls im Hintergrund, Daten an den Server gesendet werden. Diese Datenübertragung kann dem früheren Akronym entsprechend mittels XML durchgeführt werden. Jedoch hat sich die Übertragung als reiner Text oder im JSON-Format durchgesetzt. [184]

Für den Einsatz von Ajax ist das Objekt `XMLHttpRequest` essentiell. Dieses Objekt enthält alle Daten, die mit dem Server ausgetauscht werden. Nachdem das Objekt erstellt ist, wird mit der Funktion `open()` die Anfrage bestimmt. Dabei werden mehrere Parameter übergeben:

- `method`: Der Typ der Anfrage wird bestimmt (`GET` oder `POST`).
- `url`: Angabe der URL zum Dokument, dessen Inhalt geladen werden soll.

---

[181] Vgl. Node.js Documentation: File-System; vgl. auch npm (Hrsg.): file-system; vgl. auch file-system (GitHub).

[182] Siehe dazu auch 3.6.1.6 Ausgaben des Spider-Moduls, S. 78.

[183] Ursprünglich wurde AJAX als Akronym für Asynchronous JavaScript And XML bekannt. Nachdem AJAX äußerst selten im Zusammenhang mit XML eingesetzt wurde, wandelte sich das Akronym AJAX zum Begriff Ajax, der für einen nicht eindeutig definierten Technologiemix zur Erstellung von JavaScript-basierten HTTP-Anfragen steht. Vgl. Wenz, JavaScript, S. 289.

[184] Vgl. Wenz: JavaScript, S. 289-291; vgl. auch W3CSchools (Hrsg.): JS AJAX, AJAX Introduction.

- `async`: Bestimmung des Kommunikationsablauf. Als Standardwert ist die synchrone Kommunikation (`false`) festgelegt, für asynchrone Kommunikation muss der Wert auf `true` gesetzt werden.
- optional können noch der Nutzername (`user`) und das dazugehörige Passwort (`psw`) als Parameter übergeben werden.

Anschließend muss auf das Ereignis `readystatechange` gewartet werden. Der `readyState` kann fünf Werte annehmen, von nicht initialisiert (`0`) bis fertig (`4`). Mit der Funktion `send` wird die Anfrage abgeschlossen. Die Möglichkeit im letzten Schritt `POST`-Parameter zu übermitteln, wird an dieser Stelle vernachlässigt, da hierfür neben JavaScript weitere Technologien zum Einsatz kommen. [185]

Die HTTP-Anfrage kann mit Hilfe der jQuery-Bibliothek deutlich verkürzt werden. In diesem Framework ist es ausreichend die Funktion `ajax()` aufzurufen. Als Parameter wird ein Objekt übergeben, welches verschiedene Einstellungen für die HTTP-Anfrage enthält. Bis auf `url` sind alle Einstellungen optional und können mit Standardwerten versehen werden. `url` enthält die URL, an welche die HTTP-Anfrage gerichtet ist, als Zeichenkette. Als weitere Eigenschaften können bspw. `dataTyp` und `success` konfiguriert werden. `success` wird dabei eine callback-Funktion zugeordnet, die ausgeführt wird, nachdem die Anfrage erfolgreich durchgeführt wurde. Mit der Eigenschaft `dataType` wird angegeben, welcher Datentyp vom als Antwort vom Server erwartet wird. Mögliche Datentypen sind `xml`, `json`, `script`, `html` oder `text`. Ist kein Datentyp angegeben versucht jQuery diesen auf Basis des MIME type zu erraten. [186]

2.4.4.2 D3.js

Die Data-Driven Documents JavaScript-Bibliothek (im Folgenden kurz: D3.js) nutzt HTML, SVG[187] und CSS um Daten zu visualisieren. Seit der Version D3 4.0 ist die JavaScript-Bibliothek keine einzelne Bibliothek mehr, sondern modular gestaltet. Die, aus dieser Unterteilung resultierenden kleinen Bibliotheken, sind darauf ausgelegt weiterhin zusammen zu arbeiten. Zu den Stärke von D3.js zählen die einfache Darstellung von großen Datensätzen sowie Interaktivität und Animationen. Die Bibliothek steht auf GitHub unter der BSD Lizenz (Open Source) zur Verfügung und wird beständig gepflegt. [188]

Für den Navigator wird der Diagrammtyp Force-Directed Graph genutzt. Die Basis bildet hierbei eine JSON-Datei, welche die Knoten (Nodes) und Verbindungen (Links) beinhaltet. Dieser

---

[185] Vgl. Wenz: JavaScript, S. 293 - 295; vgl. auch Flanagan: JavaScript, S. 494 - 496; vgl. auch W3CSchools (Hrsg.): JS AJAX, XMLHttpRequest Object Methods.
[186] Vgl. The jQuery Foundation (Hrsg.): jQuery API Documentation, jQuery.ajax().
[187] Scalable Vector Graphics: W3C Standard zur Beschreibung zweidimensionaler Vektorgrafiken.
[188] Vgl. Bostock: Data-Driven Documents; vgl. auch Bostock: Changes in D3 4.0.

Diagrammtyp liefert bereits im Standard-Layout Berechnungen für die Positionierung der Nodes und Links. [189]

2.4.4.3 JavaScript Object Notation

JavaScript Object Notation (im Folgenden kurz: JSON) entstammt der JavaScript Spezifikation für Arrays und Objekte. JSON ist demzufolge ein Objekt, welches Arrays und weitere Objekte beinhaltet. Mit dem Namensraum JSON werden zwei Funktionen für die Objekte ermöglicht, welche bereits durch das ECMA-Script5 definiert sind. Gleichzeitig ist JSON ein Dokumentenformat, welches durch einen eigenen ECMA-Standard (ECMA-404, veröffentlicht im Oktober 2013) definiert wird. [190] Neben dem diesem Standard wird JSON ebenfalls durch den Standard IETF RFC 7159 (veröffentlicht im März 2014) definiert. Der IETF-Standard hat sich dabei das Ziel gesetzt, das bis zur Veröffentlichung entstandene Korrekturverzeichnis in den neuen Standard zu übertragen, Inkonsistenzen zwischen anderen JSON-Spezifikationen zu entfernen und Praktiken hervorzuheben, die zu Problemen der Interoperabilität führen können. [191]

Ein JSON-Dokument enthält entweder ein Objekt oder ein Array, in denen beliebig viele Objekte und Arrays enthalten sein können. Objekte werden der JavaScript-Syntax entsprechend mit geschweiften Klammern (`{}`) umschlossen, Arrays mit eckigen Klammern (`[]`). Objekte enthalten eine ungeordnete Sammlung von Namen-Werte-Paare, Arrays enthalten eine geordnete Liste von Werten. Die einzelnen Daten innerhalb des Objekts, bzw. des Arrays werden durch Kommas getrennt. Zeichenketten müssen abweichend zur JavaScript-Syntax zwingend in doppelte Anführungszeichen gesetzt werden. Obwohl das Dokumentenformat an die JavaScript-Syntax angelehnt ist, enthält es nur Text, sodass neben JavaScript auch andere Sprachen dieses Format einbinden können. [192]

Die für den Navigator benötigten Nodes und Links sollen automatisch aus den jeweiligen Ausgangsdatenformaten erzeugt und als JSON-Datei gespeichert werden, bevor sie mit D3.js verarbeitet werden. Hierfür soll ein JSON-Objekt erstellt werden, welches die zwei Namen nodes und links enthält. Die dazugehörigen Werte sollen jeweils in einem Array erfasst werden, welche wiederum mehrere Objekte enthalten. Ziel ist es, dass jeder Link und jede Seite als Objekt im entsprechenden Array erfasst wird.

---

[189] Vgl. Bostock: Force-Directed Graph.
[190] Vgl. Wenz: JavaScript, S. 299; vgl. auch Flanagan: JavaScript, S. 786; vgl. auch Introducing JSON.
[191] Vgl. JSON Spezifikation ECMA-404; vgl. auch JSON Spezifikation IETF RFC 7159.
[192] Vgl. Introducing JSON; vgl. auch W3CSchools (Hrsg.): JS Tutorial, JS JSON.

2.4.5 XML-Technologien

2.4.5.1 XPath und XQuery

XPath (kurz für: XML Path Language) und XQuery (kurz für: XML Query Language) sind zwei vom W3C entwickelte Abfragesprachen zur Adressierung von Knoten in XML-Dokumenten. Sie basieren auf dem gleichen Datenmodell und unterstützen identische Funktionen und Operatoren. Die aktuellste Version 3.0 wurde im April 2014 als Standard veröffentlicht. Seit dieser Version sind u. a. standardisierte Winkel- und Exponentialfunktionen verfügbar. [193] Einzelne Funktionen und Operatoren werden aufgrund deren Vielfältigkeit an dieser Stelle nicht erörtert.

XPath ist Teil der XSL[194]-Familie, die zur Transformation und Präsentation von XML-Dokumenten in verschiedene Ausgabeformate konzipiert wurde. XPath nutzt eine kompakte, nicht-XML-Syntax um die Verwendung von XPath innerhalb von URIs und XML-Attributwerten zu erleichtern. Die Ausdruckssprache wurde konzipiert, um in anderen Sprachen, bspw. XSLT, eingebunden zu werden. XPath arbeitet mit verschiedenen Knotentypen, Achsen und Pfaden. Die Adressierung der Knoten in XML-Dokumenten ist als ausführliche und verkürzte Notation möglich. Die XPath-Notation muss innerhalb eines Dokuments nicht einheitlich erfolgen. [195]

XQuery ist auf den XPath-Ausdrücken aufgebaut und nutzt eine XML-basierte Syntax. Die Sprache unterscheidet Groß- und Kleinschreibung, Zeichenketten können in doppelten oder einfachen Anführungszeichen erfasst werden, Variablen werden mit dem Dollar-Zeichen ($) definiert und alle Elemente, Attribute sowie Variablen müssen der XML-Namenskonvention entsprechen. [196]

2.4.5.2 XProc

XProc ist eine von Norman Walsh entwickelte und vom W3C als Standard veröffentlichte XML Pipeline Language. Die Technologie dient zur Verarbeitung von XML-Dokumenten in einer Prozesskette. Hierfür werden in einer Pipeline Schritte (sog. Steps) beschrieben, die anschließend durch einen XProc-Prozessor sequentiell abgearbeitet werden. Es besteht die Möglichkeit andere Techniken, bspw. XSLT, in einen Step einzubinden. Jede Pipeline hat mindestens einen Eingangsport (Input-Port), dem beliebig viele XML-Dokumente übergeben werden. Nach der Verarbeitung mit einem XProc-Prozessor erscheint das Ergebnis am Ausgangsport (Output-Port). Als Ergebnis können beliebig viele XML-Dokumente ausgegeben werden. Andere Doku-

---

[193] Vgl. XPath/XQuery Functinos and Operators 3.0 Spezifikation.
[194] Extensible Stylesheet Language.
[195] Vgl. W3C (Hrsg.): XSL; vgl. auch SelfHTML (Hrsg.): XML/XSL/XPath; vgl. auch XPath Spezifikation.
[196] Vgl. XQuery Spezifikation; Introduction; vgl. auch W3CSchools (Hrsg.): XQuery Tutorial, XQuery Introduction und XQuery Syntax.

mentenformate können zusätzlich als Option erzeugt werden. Sowohl beim Input-Port als auch am Output-Port besteht die Möglichkeit, dass kein XML-Dokument übergeben wird. Der Zugriff auf die Inhalte der XML-Dokumente erfolgt mittels XPath. [197]

Innerhalb einer Pipeline können externe Befehle ausgeführt werden. Dieser Step wird mittels `p:exec` initialisiert. Es können verschiedene Optionen übergeben werden. Verpflichtend ist die Option '`command`', welche den auszuführenden Befehl festlegt. Tabelle 7 stellt weitere Optionen vor, die für die Ausführung des '`node`'-Befehls verwendet werden sollen. [198]

Option	Wert	Beschreibung
`cwd`	Zeichenkette	Ändert das Arbeitsverzeichnis, bevor der Befehl ausgeführt wird. Der Pfad muss relativ zum aktuellen Arbeitsverzeichnis angegeben werden.
`result-is-xml`	Boolescher Wert	Als Standwardwert ist `true` definiert und das Ergebnis wird als XML-Dokument geparst. Ist das Ergebnis kein XML (`false`), wird es als Text ohne MarkUp ausgegeben.
`args`	Zeichenkette	Übergibt dem Befehl beliebig viele Argumente. Mehrere Argumente werden mit einem Leerzeichen getrennt und als einzelne Werte interpretiert.
`arg-separator`	Zeichenkette	Definiert einen alternativen Separator für die Auflistung mehrerer Argumente. Als Standardwert ist ein Leerzeichen definiert.

Tabelle 7: Übersicht über ausgewählte Optionen des p:exec-Step [199]

Das geplante Spider-Modul kann mit dieser Methode in die Pipeline zur Erstellung des HTML-Navigators eingebunden werden.

2.4.5.3 XSLT

XSLT (kurz für Extensible Stylesheet Language Transformation) ist eine vom W3C als Standard veröffentlichte Transformationssprache. Die Technologie dient zur Formatierung von XML-Dokumenten und deren Transformation in verschiedene Dokumentenformate. Hierfür werden Templates erstellt, welche die Baumstruktur des XML-Dokuments in eine, dem Zielformat entsprechende, Baumstruktur übersetzen. Daher können grundsätzlich nur XML-gerechte Dokumentenformate Ziel der Transformation sein. Zur Formatierung von XML-Dokumenten enthält die XSL-Komponente der Technologie verschiedene Style-Eigenschaften, sowie diverse Möglichkeiten, den logischen Ablauf der Datenpräsentation zu steuern. Der Zugriff auf die Inhalte der XML-Dokumente erfolgt mittels XPath. [200]

---

[197] Vgl. data2type (Hrsg.): Das Konzept von XProc; vgl. auch XProc Spezifikation; vgl. auch Walsh: XML Processing.
[198] Vgl. XProc Spezifikation, 7.2 Optional Steps/p:exec.
[199] Vgl. XProc Spezifikation, 7.2 Optional Steps/p:exec.
[200] Vgl. XSLT 1.0 Spezifikation; vgl. auch SelfHTML (Hrsg.): XML/XSL/XSLT.

Die Transformationssprache wurde erstmals im November 1999 vom W3C standardisiert. Der aktuellste Standard, XSLT 2.0 wurde im Januar 2007 veröffentlicht. Seit dieser Version kann ein Stylesheet standardisiert mehrere Ergebnisdokumente ausgeben und es muss hierfür nicht mehr auf zusätzliche Features des gewählten Prozessors zurückgegriffen werden. XSLT 3.0 war seit November 2015 als Spezifikationskandidat veröffentlicht. Bis März 2016 durften hierzu Kommentare an die 'W3C XSLT Working Group' übermittelt werden. Der neue Standard soll u. a. die Möglichkeiten zum Streaming verbessern und weitere zum Packaging definieren. Außerdem wird XSLT3.0 enger mit XPath 3.0 und dem Datenmodel, das beiden zugrunde liegt, verknüpft sein. [201]

Die Transformationsanweisungen eines Stylesheets müssen durch einen Prozessor geparst werden. Hierbei gibt es mehrere Möglichkeiten, an welcher Stelle die Transformation ausgeführt wird. Sie kann auf dem eigenen Webserver erfolgen, sodass beim Client im Browser kein XML mehr erkennbar ist, weil es bereits vor der HTTP-Anfrage in ein HTML übersetzt wurde. 2016 kündigte Michael Key einen neuen revolutionären Prozessor an, welcher die Transformation in den Browser verlagern soll. Im Juli 2016 wurde die erste Beta-Version des vollständig in JavaScript geschriebenen Saxon-JS-Prozessor veröffentlicht. Die HTTP-Anfrage des Client erhält somit als Antwort die XML-Quelldaten und dazugehörige Stylesheets. Der im Browser integrierte Saxon-JS-Prozessor transformiert anschließend die XML-Daten in HTML-Dokumente. Hierdurch können im Stylesheet Regeln definiert werden, welche auf Browser-Events reagieren. [202]

Jedes Stylesheet ist ein XML-Dokument, welches die vom W3C definierten Namensräume `xsl` und `xs` nutzt. Daher hat jedes Stylesheet das Wurzelelement `<xsl:stylesheet />`. Als Attribute des Wurzelelements können beliebig viele Namensräume für das Stylesheet angegeben werden, mindestens jedoch die `xsl`- und `xs`-Namensräume. Neben dieses beiden sind weitere Namensräume seit XSLT 2.0 vom W3C reserviert. Sie werden von XSLT-Prozessoren genutzt, um Attribute und Elemente des Namensraum zu erkennen, und ermöglichen u. a. den Zugriff auf vom W3C für diverse Sprachen standardisierte Funktionen und Operatoren. Daher dürfen diese reservierten Namensräume ausschließlich der Spezifikation entsprechend verwendet werden. Jedem Namensraum muss eine URI zugeordnet werden. Stylesheets dürfen um eigene Namensräume erweitert werden. Der Zugriff auf Attribute, Elemente oder Funktio-

---

[201] Vgl. XSLT 2.0 Spezifikation, Abstract und J Changes from XSLT 1.0 (Non-Normative); vgl. auch XSLT 3.0 Spezifikationskandidat, Abstract.

[202] Vgl. SelfHTML: XML/XSL/XSLT, XSLT im Browser und auf dem Server.

nen erfolgt anschließend mit dem entsprechend Namensraum. Bei der Einbindung externer Namensräume muss geprüft werden, ob der genutzte Prozessor diese Namensräume kennt. [203]

2.4.5.4 le-tex transpect

Das Framework le-tex transpect (im Folgenden kurz: transpect) wurde von le-tex entwickelt und unter der BSD-Lizenz (Open Source) veröffentlicht. Es beinhaltet eine Zusammenstellung verschiedener Konverter für XML-basierte Dokumentenformate und Erweiterungen zur Überprüfung von Dokumenten. Des Weiteren liegt transpect eine Methodologie zugrunde, die es ermöglicht einzelne Module zu einem Workflow zu verbinden. Einige der Module können als Standalone-Werkzeuge eingesetzt werden. [204] Im Folgenden wird ein Modul umrissen, welches bei der Erstellung des Navigators verwendet werden soll.

Mit dem Modul `xproc-util` stellt transpect eine der XProc-Kernbibliotheken zur Verfügung, die bspw. Zwischenergebnisse speichern kann. Außerdem enthält sie u. a. Bibliotheken für "lineare XSLT-Konvertierpipelines mit einem Mode pro Durchlauf"[205]. Alle enthaltenen Funktionen werden im transpect-Namensraum `tr` ausgeführt. [206] Das Modul soll in der Navigator-Anwendung genutzt werden, um alle HTML-Dokumente aus dem Verzeichnis auszulesen, das mit dem Spider-Modul erstellt wurde. Anschließend sollen die einzelnen HTML-Dokument in XHTML-Dokumente konvertiert werden. [207]

2.5 Werkzeuge

2.5.1 Atom Texteditor

Atom ist ein frei verfügbarer Texteditor. Er wurde im Rahmen des Electron-Frameworks[208] unter Einsatz von JavaScript, HTML und CSS geschrieben. Die aktuelle Version Atom 1.12 ist seit November 2016 über GitHub verfügbar. [209] Der Texteditor ist eine spezielle Version des Webbrowsers Chromium. Atom konnte nicht als reine Webanwendung entwickelt werden, da es nicht möglich gewesen wäre, Daten lokal zu speichern. [210]

---

[203] Vgl. XSLT 2.0 Spezifikation, 3 .1 XSLT Namespace und 3.2 Reserved Namespaces.
[204] Vgl. le-tex: le-tex transpect.
[205] le-tex: le-tex transpect.
[206] Vgl. le-tex: le-tex transpect; vgl. auch le-tex publishing services (Hrsg.): transpect, Modules/ xproc-util.
[207] Siehe dazu auch 3.6.2.1 Zusammenführung der HTML-Dokumente in eine rekursive Verzeichnisliste, S. 80.
[208] Das Electron-Framework basiert auf Node.js und Chromium.
[209] Vgl. electron (Hrsg.): electron; vgl. auch atom (Hrsg.): atom, releases.
[210] Vgl. Atom.io (Hrsg.): Flight Manual,

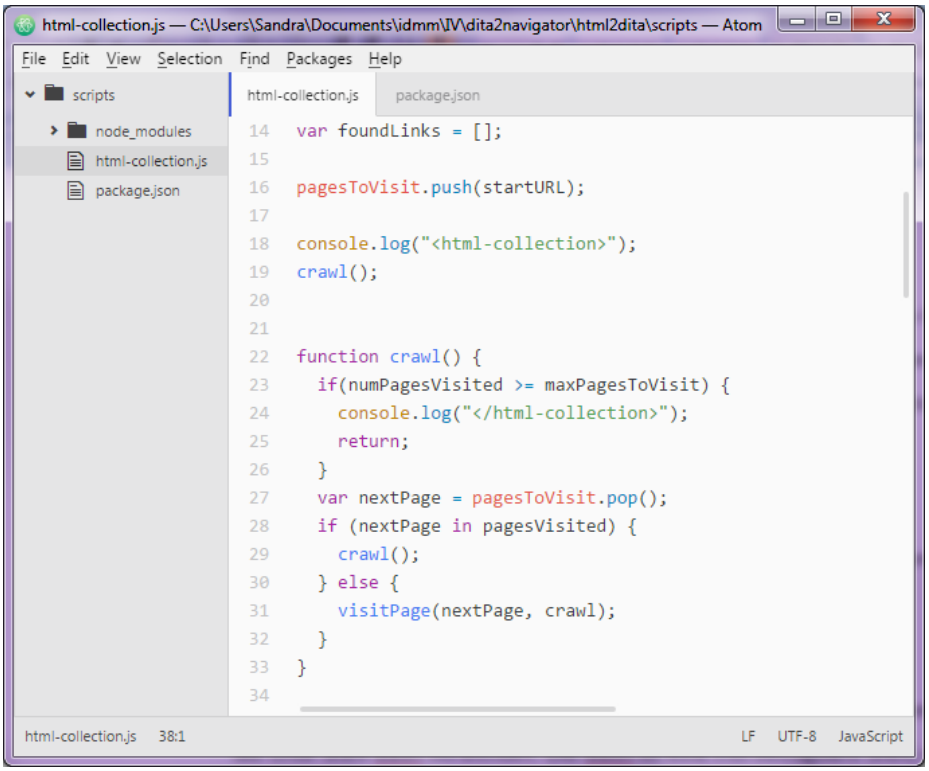


Abb. 9: Benutzeroberfläche von Atom mit Syntax-Highlighting am Beispiel eines JavaScript-Dokuments

Neben dem Syntax-Highlighting, bspw. für JavaScript (Abb. 9), bietet Atom eine automatische Vervollständigung von Funktionen und Variablen an.[211] Daher wurde Atom im Rahmen des Projektes für die JavaScript-Entwicklungsarbeiten, insbesondere für Node.js, verwendet.

2.5.2 Bash Konsole

Für den Aufruf der Transformationen über die Konsole wurde unter dem Betriebssystem Linux die Unix-Shell Bash verwendet. Die Bash ist eine frei verfügbare Shell, die 1987 von Brian Fox geschrieben wurde und seit 1990 von Chet Remney weiterentwickelt wird. [212]

Unter dem Windows-Betriebssystem erfolgte der Zugriff auf die Bash über Cygwin. Dies ist eine Sammlung von GNU- und Open Source-Tools, die es ermöglicht unter Windows bestimmte Linux-Tools auszuführen. Beliebigen Linux-Programme auszuführen, ist nicht möglich. [213]

Die Bash Konsole ermöglicht es Skripte zu erstellen und auszuführen.[214] Diese Funktion wurde im Rahmen des Projekts zur Erstellung von Front-End-Pipelines der einzelnen Navigatoren genutzt.

---

[211] Vgl. Atom.io (Hrsg.): Atom, Full-featured, right out of the box.
[212] Vgl. GNU (Hrsg.): GNU Bash.
[213] Vgl. Cygwin: Cygwin.
[214] Vgl. Ramey, Fox: Bash Reference Manual, S. 46.

2.5.3 Oxygen XML-Editor

Der Oxygen XML-Editor ist ein, von der Firma Syncro Soft SRL herausgegebener, Editor. Seit Oktober 2016 ist die Aktuelle Version 18.1 verfügbar. Der Editor ist sowohl für Entwickler als auch Autoren ausgelegt. Zur Bearbeitung der verschiedenen Daten stehen daher drei Modi zu Verfügung: Text, Raster und Autor. Im Textmodus erfolgt die Bearbeitung der Daten als Quellcode, der Rastermodus stellt die Daten in einer Tabelle dar und der Autormodus ist eine mit CSS formatierte WYSIWYG[215]-Ansicht. [216]

Der Editor unterstützt alle offiziellen XML-Standards (u. a. DITA, DocBook und HTML/XHTML) und prüft im Hintergrund laufend die Wohlgeformtheit und Validität der XML-Dokumente. Eine weitere Stärke des Editors ist das Syntax-Highlighting im Textmodus. [217]

Im Rahmen des Projektes wurde der XML-Editor für Analyse- und Entwicklungsarbeiten im Bereich der XML-Technologien genutzt. Alle Dokumente wurden in einem Oxygen-Projekt gespeichert. Das ermöglicht einerseits das benötigte Dokument im Oxygen XML-Editor schneller aufzurufen, andererseits können in dem implementierten Transformationswerkzeug Pfade relativ zum Projekt bestimmt werden. Dadurch kann das Projekt an anderen Arbeitsplätzen genutzt und Transformationen durchgeführt werden, ohne dass diese nochmals konfiguriert werden müssen.

2.5.4 XSLT-Prozessor Saxon

Der Saxon-Prozessor ist Teil eines Programmpakets[218] zur Verarbeitung von XML-Dokumenten. Er unterstützt XSLT 1.0 und 2.0 vollständig, XSLT 3.0 fast vollständig. Der Saxon-Prozessor wird von Michael Kay in der Firma Saxonica entwickelt. Seit der Version 9.3 erfolgt die Weiterentwicklung in einem Team von bis zu vier Personen, wobei Michael Kay weiterhin federführend bleibt. Der Saxon-Prozessor kann über die Plattformen Java und .NET genutzt werden. [219]

Den XSLT-Prozessor gibt es in verschiedenen Ausführungen:

- Saxon-HE (Home Edition, Open Source)
- Saxon-PE (Professional Edition, kommerziell)
- Saxon-EE (Enterprise Edition, kommerziell)
- Saxon-CE (Client-Edition, Open Source)
- Saxon 6.5.5 (Open Source)

---

[215] Kurz für 'What You See Is What You Get'.
[216] Vgl. SyncRO Soft: Oxygen XML Editor.
[217] Vgl. SyncRO Soft: Oxygen XML Editor.
[218] Das Programmpaket Saxon beinhaltet neben dem XSLT-Prozessor auch Prozessoren zur Verarbeitung von XPath (bis XPath 3.0, rückwärtskompatibel), XQuery (1.0 und 3.0) und XML Shema (1.0 und 1.1).
[219] Vgl. Saxonica: About Saxon/ What is Saxon? und Historical Note.

Die Ausführungen bieten jeweils einen unterschiedlichen Umfang an Funktionen. Saxon 6.5.5 steht weiterhin als Prozessor für XSLT 1.0 zur Verfügung, wird aber nicht mehr weiter entwickelt. [220]

Im Rahmen des Projektes wird der Saxon-Prozessor über den Calabash-Prozessor aufgerufen sobald eine XSL-Transformation durchgeführt werden muss.

2.5.5 XProc-Prozessor Calabash

Der Calabash-Prozessor ist eine von Norman Walsh entwickelte Implementierung zur Verarbeitung von XProc. Die Implementierung wurde auf den Saxon-Prozessor aufgebaut und ist daher nur in Verbindung mit diesem einsetzbar. Welche Ausführung des Saxon-Prozessors benötigt wird, hängt davon ab, in welchem Umfang der Calabash-Prozessor verwendet werden soll. Der Calabash-Prozessor an sich ist als Open Source verfügbar. Er kann im Gegensatz zum Saxon-Prozessor nur über die Plattform Java[221] genutzt werden. Die Anwendung ist kommandozeilenbasiert. Auch ein Aufruf über das Transformationswerkzeug im Oxygen XML-Editor ist möglich. Die aktuellste Version 1.1.9-96 wurde im September 2015 veröffentlicht. [222]

Für die Erstellung des Navigators wurde der Prozessor über die Kommandozeile der Bash Konsole aufgerufen. Durch die Ergänzung des Befehls '`-D`' sind auf diesem Weg mehr Debugging-Informationen verfügbar, als bei einem Aufruf über das Transformationswerkzeug des Oxygen XML-Editors.

---

[220] Vgl. Saxonica: Our Products; vgl. auch Saxonica: Saxon 9.7 product comparison (Feature Matrix); vgl. auch Kay: About SAXON.

[221] Version 1.5 oder höher.

[222] Vgl. Walsh: XML Calabash; vgl. auch What is XML Calabash?; vgl. auch data2type: Das Konzept von XProc/ XProc-Prozessoren.

3 Umsetzung

3.1 Verwendete Beispiele

Die Navigator-Anwendung wird anhand von drei Beispielen entwickelt. Hierbei soll dennoch eine möglichst universelle Ansprache der benötigten Elemente und Attribute erfolgen, sodass die Anpassung an andere Quelldokumente des gleichen Formats möglich ist. Die folgenden Kapiteln sollen diese Beispiele vorstellen. Anschließend wird die Struktur der Beispiele, mit dem Fokus auf die, für den Navigator relevante Struktur, analysiert.

3.1.1 Website der le-tex als Beispiel für DocBook

Der Navigator wurde ursprünglich für die Website der le-tex entwickelt. Ziel war eine interaktive Navigation, mit der sich das Unternehmen bspw. auf Messen repräsentieren kann. Der Navigator sollte auch als optionales Feature auf der regulären Website eingebunden werden. Für die Entwicklung des Navigators wurde die JavaScript-Bibliothek D3.js ausgewählt. [223]

Die Inhalte der Website werden in zwei DocBook-Dokumenten gepflegt, eins als Basis für die deutschsprachigen Seiten und eins für die englischsprachigen. Hierbei wurde der Metadaten-Bereich `<authorgroup>`, unterteilt in `<author>` und `<othercredit>`, genutzt, um Informationen zu verschiedenen Ansprechpartnern bereitzustellen. Jeder aufgelistete Ansprechpartner wird als eine einzelnes HTML-Dokument gerendert. Die vorgegebene hierarchische Struktur wird im Rendering als Navigation mit zwei Ebenen wiedergegeben. Gemeinsam mit weiteren Angaben zum Unternehmen ist `<authorgroup>` im `<info>`-Block des DocBook-Dokuments erfasst. Die Index-Seite wird aus dem `<preface>` des DocBook-Buches erstellt. Im Bereich `<appendix>` werden verschiedene Angaben zur Lokalisierung spezieller Elementen (bspw. für Bild-Referenzen) gemacht. Die fünf Hauptbereiche der Website (Unternehmen, Dienstleistungen, Produkte/Lösungen, Technologien und Referenzen)[224] werden den `<part>`-Elementen des DocBook-Dokuments entnommen. Jeder Part wird als einzelnes HTML-Dokument gerendert und enthält mehrere Kapitel (`<chapter>`), welche ebenfalls in einzelne HTML-Dokumente transformiert werden. Bis auf `<authorgroup>` ist jedem genannten Element eine ID zugeordnet.

Interne Links werden in dem DocBook-Dokument immer mit dem `@linkend`-Attribut angesprochen. Hierbei werden Ansprechpartnerüber das `<xref>`-Element verlinkt, andere Linkziele werden im `<link>`-Element erfasst. Grundsätzlich zielen alle mit `@linkend` adressiert Querbeziehungen zu Seiten, welche vom Navigator abgebildet werden. `<link>`-

[223] Vgl. Jache: Interaktive Visualisierung von Querbeziehungen in XML-Daten mit D3.js, S. 6.
[224] Vgl. le-tex.

Elemente, deren ID mit der Zeichenkette 'buchmesse-archive' beginnt, sind hiervon ausgenommen, weil diese zu PDF-Dokumenten führen.

3.1.2 Das Garagenbeispiel des DITA-OT als Beispiel für DITA

DITA-OT besteht aus einer Reihe von Java-basierten Open Source Werkzeugen zur Verarbeitung von XML-Daten im DITA-Format. Vordergründig ist es ein Werkzeug zur Veröffentlichung von DITA-Inhalten in verschiedenen Ausgabeformaten, bspw. HTML 5, PDF, DocBook oder HTML Help. Das DITA-OT gehört nicht zu OASIS, sondern ist eine unabhängige, separate Implementierung. [225]

Der DITA-Navigator wurde anhand des Garagenbeispiels des DITA-OT entwickelt. Hierbei wurde die `hierarchy.ditamap` der Version DITA-OT 1.2.1 verwendet. Das Garagenbeispiel umfasst mehrere Concept- und Task-Topics. Die hierarchische DITA-Map ist in ebendiese zwei Bereiche unterteilt.

```
<map title="Eclipse content aggregated by a map">
<topicref href="tasks/garagetaskoverview.xml" type="concept">
  <topicref href="tasks/changingtheoil.xml" type="task"/>
  <topicref href="tasks/organizing.xml" type="task"/>
  ...
</topicref>
<topicref href="concepts/garageconceptsoverview.xml" type="concept">
  <topicref href="concepts/lawnmower.xml" type="concept"/>
  <topicref href="concepts/oil.xml" type="concept"/>
  ...
</topicref>
</map>
```

Quellcode 8: Auszug aus der hierarchischen DITA-Map des DITA-OT-Garagenbeispiels

Jedem Topic ist eine ID und ein `@xml:lang`-Attribut zugeordnet. Querverbindungen zu anderen Topics werden ausschließlich als relative Pfade im `<related-links>`-Container angegeben. Keines der Topics enthält Querverweise zu externen Ressourcen. Die Indexseite wird aus der DITA-Map erzeugt und jedes Topic wird als einzelnes HTML-Dokument gerendert. Die vorgegebene hierarchische Struktur wird im HTML-Rendering als Navigation mit zwei Ebenen umgesetzt.

3.1.3 Zeit Online als Beispiel für HTML-Dokumente

3.1.3.1 Begründung der Auswahl des Beispiels

Als Basis für den HTML-Navigator wurde die Zeit Online Website[226] genutzt. Zeit Online versteht sich als "eine der größten deutschsprachigen Plattformen für anspruchsvollen Online-Journalismus"[227], die sich als Ziel gesetzt hat relevante Nachrichten den Lesern sofort zu prä-

---

[225] Vgl. Anderson, Elovirta, Fienhold Sheen: DITA Open Toolkit.
[226] http://www.zeit.de/index.
[227] Zeitverlag (Hrsg.): Marken & Produkte, Zeit Online.

sentieren. Abb. 10 gibt eine Übersicht der verschiedenen Ressorts der Plattform. Neben den klassischen Ressorts einer Zeitung bietet Zeit Online diverse Dienstleistungen an sowie ausgewählte Schwerpunkte, in denen alle Beiträge des Themas zusammengestellt werden. [228] Dieses Beispiel wurde ausgewählt, mit der Vermutung das innerhalb der Ressorts interessante Querverbindung entstehen. Diese Annahme wurde durch eine Beobachtung verstärkt, bei dem ein Artikel, der sich mit einer Dokumentation über die Bundeskanzlerin befasst, im Ressort Politik prominent platziert wurde. Nach der Auswahl des Artikels und der entsprechenden Weiterleitung zum Seite mit dem vollständigen Artikeltext, wurde das Ressort 'Kultur / Film & TV' als aktives Ressort angezeigt. Mit dem HTML-Navigator sollen weitere solche Querbeziehungen innerhalb der Website identifiziert und visualisiert werden.

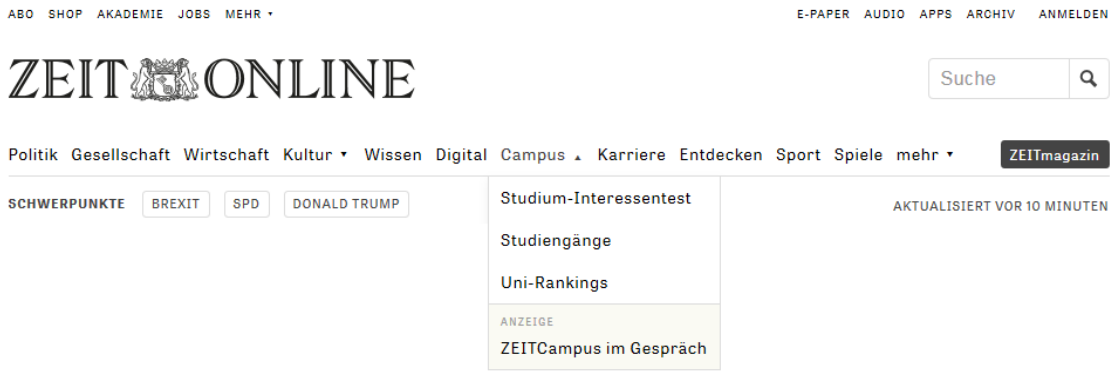


Abb. 10: Screenshot des Zeit Online Navigationsbereichs

3.1.3.2 Zusammensetzung der URL

Die Plattform nutzt HTML 5-Dokumente zur Strukturierung ihrer Inhalte. Der `<head>` der Dokumente enthält u. a. die Meta-Information '`description`', welche durch Kommas getrennte Schlagworte und Wortgruppen zur Beschreibung des Inhalts der Seite enthält. Dem `<body>` jedes Dokuments wird ein `@data-unique-id`-Attribut zugeordnet. Als Wert erhält dieses Attribut eine URL, welche mit '`http://xml.zeit.de/`' beginnt und anschließend den aktuellen Pfad angibt. Für die Startseite wird dieses Attribut mit '`index`' erweitert. Übersichtsseiten der Ressorts erhalten eine Erweiterung der genannten Basis nach dem Schema '`[ressort]/index`'. Für Artikel wird das Attribut dem Schema '`[Jahr]/[Monat]/[Titel]`' entsprechend ergänzt. Als Titel werden hierbei Schlagworte, getrennt durch Bindestriche, verwendet.

3.1.3.3 Struktur der Website

Im Folgenden wird ein Auszug aus der Struktur der HTML-Dokumente beschrieben. Dieser Auszug soll exemplarisch für die komplexe Struktur der Plattform stehen und verdeutlichen, wie

---

[228] Vgl. Zeitverlag (Hrsg.): Marken & Produkte, Zeit Online.

sorgfältig die Auswahl der Elemente und Attribute für die Erstellung des Navigators erfolgen muss.

Für alle HTML-Dokumente der Plattform gilt, dass grundsätzlich jedes Linkziel mittels absoluter Pfade adressiert wird. Neben dem `@href`-Attribut können `<a>`-Elemente teilweise `@class`- und `@title`-Attribute enthalten. Alle beschriebenen Elemente gehören zum `<body>` des HTML-Dokuments.

Der `<header>` enthält ein `<div>`-Container mit der ID '`navigation`', welchem mehrere `<nav>`- und `<div>`-Container untergeordnet sind. Der `<nav>`-Container mit dem Attribut `@class = 'nav__ressorts nav__ressorts--fitted'` enthält wiederum eine ungeordnete Liste, in der alle Ressorts eingetragen sind. Die Listenelemente wiederum beinhalten ein `<a>`-Element oder nochmals eine ungeordnete Liste, in deren Listenelemente anschließend der Link zum Ressortteilbereich enthalten ist.

Innerhalb des '`navigation`'-`<div>`-Containers befindet sich ein weiterer `<div>`-Container, der wiederum einen `<div>`-Container mit der ID '`nav__tags`' beinhaltet. An dieser Stelle werden ausgewählte Themen-Seiten als Schwerpunkte verlinkt. Die hierin ausgewiesenen `<a>`-Elemente haben die Zeichenkette '`nav__tag`' als `@class`-Attribut.

Teilbereiche der einzelnen Ressorts werden ebenfalls in `<div>`-Container erfasst. Diese Container werden über das `@class`-Attribut mit den Werten '`cp-region cp-region--parquet`' identifiziert. Auf der Startseite der Plattform werden die Container, mit diesen Attributwerten, genutzt um einerseits die einzelnen Ressorts, durch die Vorschau auf ausgewählte Artikel, zu präsentieren. Andererseits werden weitere Angebote, bspw. der ze.tt-Blog oder das Zeit Magazin, in diesen Containern dargestellt.

3.2 Anforderungen an die Navigator-Anwendung

3.2.1 Geplante Funktionalität und deren Zusammenspiel

Die Navigator-Anwendung soll für die drei ausgewählten Dokumentformate ein JSON-Dokument erstellen können, welches die Basis für die Visualisierung mit D3.js bildet. Hierfür soll eine Konfigurationsdatei entwickelt werden, in der der Anwender Angaben zum genutzten Dokumentenformat vornehmen kann. Innerhalb der Konfigurationsdatei sollen insbesondere für HTML-Dokumente zusätzliche Angaben zur Klassifizierung der Querverbindungen gemacht werden. Die Anwendung soll die Angaben aus der Konfigurationsdatei verarbeiten und anschließend als Shell-Skript über die Konsole ausführen.

Das Rendering von DocBook- und DITA-Dokumenten soll als Option zur Verfügung stehen.

Als DocBook-Dokumente sollen nur solche verarbeitet werden, die dem Typ Buch entsprechen und keine externen Abschnitte laden. Die DITA-Dokumente können neben allgemeinen Topics

auch Concept-, Task- und Reference-Topics enthalten. Spezialisierte DITA-Topics oder andere DocBook-Typen sind derzeit nicht als Bestandteile der Navigator-Anwendung vorgesehen.
Allen entsprechenden Links sollen die notwendigen `click`-Events über ausgelagerte JavaScript-Dokumente zugewiesen werden. Darüber hinaus soll die Navigator-Anwendung für jedes Dokumentenformat eine Demoversion bereitstellen.
Für HTML-Dokumente als Quelle für den Navigator soll die Möglichkeit bestehen lokale Dokumente auszuwählen oder eine URL anzugeben. Wird eine URL angegeben, soll ein Prozess ausgelöst werden, der die Inhalte der angegebenen Website ausliest (Spider-Modul) und in XHTML-Dokumente umwandelt. Der Anwender soll eingrenzen, welche Links hierbei berücksichtigt werden sollen.
Anhand der aufgestellten Kriterien für die Visualisierbarkeit von Querbeziehungen sollen die Dokumente, aus denen der Navigator erstellt wird, bewertet werden. Das Prüftool soll einen Hinweis ausgeben, falls die Kriterien nicht ausreichend erfüllt werden, aber die Erstellung des Navigators nicht blockieren.
Die verschiedenen Bestandteile der Navigator-Anwendung sollen möglichst modular gestaltet werden. Außerdem soll deren Entwicklung mit Open Source Technologien realisiert werden.

3.2.2 Entwicklungsparadigmen

Für die Entwicklung der Navigator-Anwendung soll sprechender Code genutzt werden. Insbesondere die Bezeichnungen von Funktionen und Variablen sollen keine Abkürzungen enthalten, bzw. nur solche Abkürzung, die nachvollziehbar sind. Zur Strukturierung des Codes sollen Kommentare verwendet werden. Außerdem sollen an allen notwendigen Stellen des Codes weitere Kommentare erfasst werden, um insbesondere selbst erstellte Funktionen verständlich zu erklären. Ziel ist es, dass ein fachkundiger Dritter den Code zügig verstehen kann, sodass die Weiterentwicklung und Wartung der Anwendung selbstständig durchgeführt werden kann.
Funktionen und Variablen sollen im JavaScript-Code möglichst in der CamelCase-Schreibweise bezeichnet werden, in XML-Dokumenten wird die Bindestrich-Schreibweise bevorzugt. Werden Funktionen und Variablen aus bestehenden Ressourcen übernommen, soll deren Bezeichnung nicht angepasst werden.
Die Modularisierung der Anwendung soll dadurch unterstützt werden, dass einzelne Schritte möglichst in separaten Dokumenten erfasst werden. Hierdurch soll die Übersichtlichkeit innerhalb der Dokumente und die Wiederverwendbarkeit der einzelnen Schritte erhöht sowie die zukünftige Wartung erleichtert werden.

3.3 Grundstruktur des Navigators

Als Datenbasis der Navigator-Anwendung werden alle Knoten und Links, dem Schema in Quellcode 9 entsprechend, in einem JSON-Dokument erfasst. Die Kernanwendung des Navigators wird im JavaScript-Dokument '`navigator-core.js`' erstellt, weitere benötigte JavaScript-Funktionen liefert die '`navigator-run.js`'. Dabei enthält die Kernanwendung alle Berechnungen zur Positionierung der Knoten im Navigator. Basis der '`navigator-core.js`' ist die D3.js-Bibliothek. Die '`navigator-run.js`' beinhaltet verschiedene Event-Listener und Funktionen zur Steuerung des Navigators, sowie den Pfad des JSON-Dokuments. Das JSON-Schema für den Navigator sowie die beiden JavaScript-Dokumente wurden im Rahmen der Entwicklung des DocBook-Navigators definiert. [229]

```
{
  "nodes": [
    {
      "type": "Knoten-Typ, als Basis für den Filter und CSS-Klassen",
      "url": "url (HTML) des aktuellen Knotens, zum laden der Seite",
      "name": "Name, der im Navigator angezeigt wird",
      "keywords": "Schlagwörter als Liste mit Leerzeichen getrennt, für die Suchfunktion",
      "id": "ID des aktuellen Knotens als Referenz für die Links"
    },
    ...
  ],
  "links": [
    {
      "source": "ID des Knotens, der den Link enthält",
      "target": "ID des Knotens, auf den der Link zielt"
    },
    ...
  ]
}
```

Quellcode 9: Schema des JSON-Dokuments

Für die Einbindung in HTML-Dokumente enthält die Navigator-Anwendungen die Steuerungsleiste als XHTML-Dokument in deutscher und englischer Ausführung. Die Steuerung enthält Icons des bootstrap-Frameworks. Außerdem wird dieses Framework zur Umsetzung des responsiven Designs genutzt. Daher muss dieses ebenfalls in das gewünschte HTML-Dokument geladen werden. Der Navigator stellt diese Bibliothek mit zur Verfügung. Dennoch kann sie auch von einer anderen Plattform bezogen werden.

3.4 Anpassungen der bestehenden Anwendungen

3.4.1 DITA-Navigator

Die Erzeugung des Navigators erfolgt in mehreren Schritten, die aufeinander aufbauen und in einer letzten Pipeline zusammengefasst werden. Für alle Pipelines wird der Namensraum `xmlns:kn='http://www.le-tex.de/namespace/kiosknavigator'` definiert. Zur Verarbeitungen der Daten werden XSLT-Dokumente in die Pipelines eingebunden.

---

[229] Vgl. Jache: Interaktive Visualisierung von Querbeziehungen in XML-Daten mit D3.js, S. 45.

Ausgangspunkt des DITA-Navigators ist eine die DITA-Map. Die erste Pipeline 'expand' ist zweigeteilt. Zuerst wird die Map um die Inhalte der Topics, die in ihr verlinkt sind, erweitert, indem der Topic-Inhalt dem jeweiligen <topicref>-Element untergeordnet wird. Zusätzlich erhält jedes <topicref>-Element das Attribut @xml:base. Dieses Attribut enthält die vollständige URI des Topics. Der zweite Teil der Pipeline erzeugt für jedes Topic Schlüsselwörter, in dem alle Füllwörter und doppelten Wörter entfernt werden. Die zu entfernenden Füllwörter sind in ein separates XML-Dokument ausgelagert. Zur Verwendung der nach Sprachen eingeteilten Füllwörter werden regionale Erweiterungen der Sprachcodes aller <topicref>-Elemente entfernt.[230] Außerdem werden in diesem Teil der Pipeline bestimmte Zeichen in einen JSON-gültigen Ausdruck umwandelt. Als Output der Pipeline wird ein XML-Dokument erzeugt, welches im Rahmen der Anwendung als expandierte DITA-Map bezeichnet wird. Die expandierte DITA-Map wird an die nächste Pipeline weitergegeben. Der DITA-Navigator enthält ein zusätzliches Feature, mit dem aus der expandierten DITA-Map ein PDF- oder mehrere HTML-Dokumente erstellt werden können.

Die zweite Pipeline 'analyze' erzeugt aus der expandierten DITA-Map eine XML-Repräsentation der Knoten und Querbeziehungen, welche später im Navigator dargestellt werden sollen. Die jeweiligen Eigenschaften der Knoten <node> und Querbeziehungen <link> werden in dem XML-Dokument als Attribute der Elemente erfasst. [231] Dem Wurzelelement wurde der Name 'navigator-data' zugewiesen. Im Rahmen der Anwendung wird dieses Dokument als 'nav-data-XML' bezeichnet.

Als Querbeziehungen, die zu visualisieren sind, stehen in dem Garagen-Beispiel die per DITA-Map erzeugte Hierarchie zur Verfügung, sowie Links im <related-links>-Container. Die Links in den Topic führen alle zu anderen Topics, die auch in der DITA-Map verlinkt sind. Dabei dienen diese nicht als zusätzliches Navigationsinstrument mit Verweisen auf vorangegangene, folgende und übergeordnete Topics. Die Verlinkungen erfolgen zwischen den beiden Topictypen, aber auch innerhalb eines Topictyp.

Die darauffolgende Pipeline 'jsonify' wandelt das XML-Dokument in ein JSON-Dokument um. Das JSON-Dokument ist jedoch als einziger Output der Pipeline nicht zulässig. Daher wird sie als Option der Pipeline erzeugt. Der XML-Output ist identisch mit dem XML-Input und wird mit dem Befehl <sink/> verworfen.

---

[230] Der Sprachcode 'en-us' wird bspw. auf 'en' verkürzt. Derzeit sind in dem XML-Dokumente deutsche und englische Füllwörter enthalten. Jedes Füllwort ist als ein Element dargestellt und einem Element untergeordnet, welches als Attribut den Sprachcode enthält. Somit ist es möglich zu den bestehenden Sprachen weitere Füllwörter zu erfassen und weitere Sprachen hinzuzufügen.

[231] Siehe dazu auch 3.3 Grundstruktur des Navigators, S. 65.

In der Pipeline 'dita2navigator' werden die drei genannten Pipelines importiert und als eine Pipeline zusammengefasst. Hierbei wird das JSON-Dokument als Option der 'jsonify'-Pipeline lokal gespeichert.

Die notwendigen Anpassungen des DITA-Navigators sind auf das HTML-Feature begrenzt. Einerseits wird ein Event-Listener ergänzt, sodass die `click`-Events vom HTML-Markup getrennt werden. Andererseits wird der Aufruf des Navigators prominenter als bisher im `<header>` der HTML-Dokumente platziert. In Abb. 11 ist der Link zum Navigator auf einer Ebene mit dem Link zur Startseite dargestellt. Der Aufruf über das rote Feld ist weiterhin möglich.

Abb. 11: Screenshot der Hauptnavigation des Garagenbeispiels

3.4.2 DocBook-Navigator

Der DocBook-Navigator wurde für die Website der le-tex entwickelt. Die HTML-Dokumente werden mit einem XSLT-Stylesheet aus einem DocBook-Dokument des Typs Buch gerendert. Hierbei wird u. a. eine Sitemap[232] für die Website erstellt. Das JSON-Dokuments für den Navigator wird auf Basis dieser Sitemap-Struktur generiert.

Für die Navigator-Anwendung soll die Erstellung des DocBook-Navigators als Modul vom Rendering der Website losgelöst werden. Wie bereits bei dem DITA-Navigator soll das DocBook-Dokument ausgewertet und die daraus resultierende Struktur in einer nav-data-XML ausgegeben werden. Hierfür wird die 'analyze'-Pipeline des DITA-Navigator an das DocBook-Dokument angepasst.[233] Alle DocBook-Elemente, die im Navigator als Knoten dargestellt werden sollen, besitzen ein `@id`-Attribut. Dieses enthält bei den Elementen `<author>` und `<othercredit>` zu Beginn die Zeichenkette '`(Author|Othercredit)-`'. Es wird innerhalb des Navigators zur Adressierung der einzelnen Knoten genutzt. Auch die Benennung von HTML-Dokumente greift auf dieses Attribut zurück.[234] Daher wird diese Zeichenkette aus den IDs der entsprechenden Knoten entfernt.

---

[232] Sitemap bezeichnet die vollständige, hierarchisch strukturierte Darstellung aller Dokumente einer Website. (https://www.sitemaps.org/de/).

[233] Siehe dazu auch Anhang C: XSLT-Stylesheet des XProc-Steps 'analyze-docbook', S. XCVI.

[234] Vgl. Jache: Interaktive Visualisierung von Querbeziehungen in XML-Daten mit D3.js, S. 47.

Das Stylesheet wird um den DocBook-Namensraum erweitert. Dieser wird anschließend den XPath-Ausdrücken als Standardwert zugeordnet, sodass die DocBook-Elemente ohne Angabe des Namensraum ausgewählt werden können. Sowohl die nav-data-XML als auch das DocBook-Dokument enthalten <link>-Elemente. Für die nav-data-XML ist kein Namensraum definiert, daher muss der fehlende Namensraum, wie im Quellcode 10 Zeile 27 dargestellt, für `<link>`-Elemente der nav-data-XML als Bedingung für die Auswahl angegeben werden.

```
13  <xsl:template match="/">
14    <navigator-data>
15      <nodes>
16        <xsl:apply-templates
17              select="preface | //part | //chapter | //author | //othercredit"
18              mode="nodes"/>
19      </nodes>
20      <links>
21        <xsl:variable name="prelim" as="element(*)*">
22          <xsl:apply-templates
23              select="preface | //part | //chapter | //author | //othercredit"
24              mode="links"/>
25        </xsl:variable>
26        <xsl:for-each-group
27              select="$prelim[self::*:link][namespace-uri() = '']"
28              group-by="kn:link-key(.)">
29          <xsl:apply-templates select="current-group()[1]"/>
30        </xsl:for-each-group>
31      </links>
32    </navigator-data>
33  </xsl:template>
```

Quellcode 10: Auszug aus der analyze-docbook.xsl

Abweichend zum DITA-Navigator werden die Schlüsselwörter für den DocBook-Navigator erst bei der Erstellung der nav-data-XML erzeugt. Hierbei werden ebenfalls die Funktionen der DITA-Pipeline genutzt, sodass die sprachspezifischen Füllwörter aus dem externen XML-Dokument geladen werden. Die aus dem DocBook-Dokument erstellte nav-data-XML kann anschließend die 'jsonify'-Pipeline des DITA-Navigators verwenden.

Der DocBook-Navigator enthält ebenfalls ein Feature zum HTML-Rendering des DocBook-Dokuments. Das bestehende XSLT-Stylesheet wird ebenfalls hinsichtlich der `click`-Events angepasst, sodass diese in ein JavaScript-Dokument ausgelagert sind. Des Weiteren werden Templates entfernt, welche zur Erstellung des Navigators verwendet wurden.

3.4.3 HTML-Rendering als Feature für DocBook und DITA

Das HTML-Rendering wird als Demonstration der Navigator-Anwendung für DocBook und DITA integriert. Die Basis für beide Dokumentenformate bildet das XSLT-Stylesheet der le-tex Website, wobei dieses für den DITA-Navigator angepasst wurde. Das von Marcel Langer entwickelte

Design der le-tex Website[235] wird hierbei als CSS-Dokument in die Demo-Version des DITA- und DocBook-Navigators eingebunden.

Beide Dokumentenformate greifen auf die gleichen Skripte zu. In der 'nav-run.js' muss jedoch der Pfad zur entsprechenden 'navigator.json' angepasst werden. In dieses Dokument wurden die `click`-Events ausgelagert, welche das Standardverhalten der HTML-Links unterdrücken, während der Navigator geöffnet ist.

3.5 Einbindung der halbautomatischen Klassifikation in die Navigator-Anwendung anhand der ausgewählten Beispiele

3.5.1 Basis für die Klassifikation der Dokumentenformate

Die Knoten in der jeweiligen Navigator-Anwendung enthalten die Eigenschaft 'type'. Der Wert dieser Eigenschaft soll durch die halbautomatische Klassifikation der Knoten definiert werden. In allen drei Dokumentenformaten sollen Knoten, welche keinem speziellen Typ zugeordnet werden können, den Typ 'generic' erhalten. Die Indexseite soll nicht klassifiziert werden. Ihr wird manuell ein einmaliger Typ zugeordnet, damit CSS-Stylesheets diesen Knoten genauso wie die anderen adressieren kann.

Die Klassifikation erfolgt für jedes Dokumentenformat im Rahmen der Pipeline 'analyze', aus der die nav-data-XML erzeugt wird. Im Folgenden werden diese Prozesse getrennt nach den Dokumentenformaten beschrieben, weil die hierfür eingesetzten Methoden zwischen den verschiedenen Formaten variiert und stark auf die jeweiligen Beispiele zugeschnitten ist.

3.5.2 DITA

Im Garagenbeispiel ist jedem `<topicref>`-Element ein `@type`-Attribut zugeordnet. Die Knoten des Navigators sollen daher grundsätzlich den Typ des Topics, welchen sie repräsentieren, zugeordnet bekommen. Ausnahme hiervon sind die zwei `<topicref>`-Elemente, denen weitere `<topicref>`-Elemente untergeordnet sind. Sie sollen den Knoten-Typ 'category' erhalten. Das `<map>`-Element enthält kein `@type`-Attribut. Ihr wird daher der Knoten-Typ 'map' manuell zugewiesen. Die Kategorisierung erfolgt in der Pipeline 'analyze'. Hierfür wird ein XSLT-Template erstellt, welches `<topicref>`-Elemente im Modus 'nodes' anspricht. Innerhalb des Templates wird eine Variable 'type' erstellt, deren Wert mittels `<xsl:choose>` definiert wird. Diese Element verbindet mehrere aufeinanderfolgende Abfragen zum Namen des Elternelements des aktuellen `<topicref>`-Elements. Dem Ergebnis der Abfrage entsprechend wird der Knoten-Typ entweder als Zeichenkette festgelegt oder aus dem `@type`-Attribut des Elements ausgelesen. Anschließend wird das nav-data-Element `<node>` erstellt und neben

[235] Vgl. le-tex; vgl. auch Langer: Grafik und Illustration zur Kommunikation.

weiteren Attributen erhält es das Attribut `@type`, dessen Wert über die Variable type definiert wird.

3.5.3 DocBook

Im DocBook-Beispiel erfolgt die Zuordnung der Knoten-Typen über die ID des aktuellen oder übergeordneten `<part>`-Elements. Hierfür wird eine Funktion erstellt, welche genau ein Element übergeben bekommt. Innerhalb der Funktion werden ebenfalls mehrere aufeinander folgende Abfragen miteinander verbunden. Vergleichbar zur DITA-Methode wird auch in der Funktion für DocBook das Elternelement zur Klassifikation des aktuellen Elements angesprochen. Quellcode 11 zeigt jedoch, dass abweichend zum DITA-Template anschließend das `@xml:id`-Attribut und nicht der Name des Elternelements überprüft wird.

```
87  <xsl:function name="letex:node-type" as="xs:string">
88    <xsl:param name="content-item" as="element(*)" />
89    <xsl:choose>
90      <xsl:when test="$content-item/self::part">
91        <xsl:value-of select="'category'"/>
92      </xsl:when>
93      <xsl:when test="$content-item/parent::part/@xml:id eq 'technology'">
94        <xsl:value-of select="'technology'"/>
95      </xsl:when>
96      <xsl:when test="$content-item/parent::part/@xml:id eq 'products'">
97        <xsl:value-of select="'products'"/>
98      </xsl:when>
99      <xsl:when test="$content-item/parent::part/@xml:id eq 'services'">
100       <xsl:value-of select="'services'"/>
101     </xsl:when>
102     <xsl:when test="$content-item/parent::part/@xml:id eq 'case-studies'">
103       <xsl:value-of select="'story'"/>
104     </xsl:when>
105     <xsl:otherwise>
106       <xsl:value-of select="'generic'"/>
107     </xsl:otherwise>
108   </xsl:choose>
109 </xsl:function>
```

Quellcode 11: Klassifikation der Knoten im DocBook-Beispiel

3.5.4 HTML

Auf der Zeit Online Plattform wird bereits eine Klassifikation der Daten vorgenommen. Als Klassen werden hierbei die in Abb. 12 benannten Ressorts und deren untergeordneten Themenbereiche genutzt. Sie führen jeweils zu einer Indexseite. Die grün gekennzeichnete Themenbereiche sind dabei bereits von der Hauptseite aus über das jeweilige Dropdown-Menü erreichbar. Die erste Ausnahme hiervon sind die drei Unterpunkte Brexit, SPD und Donald Trump. Sie werden separat von der Hauptnavigation direkt auf der Indexseite im Bereich Schwerpunkte verlinkt. [236] Ihre URL beginnt einheitlich mit '`http://www.zeit.de/thema/`'.

[236] Stand 27. Januar 2017.

Im Gegensatz zu den Ressorts existiert keine Indexseite als übergeordnete Instanz der drei Unterpunkte. Eine weitere Ausnahme bildet der Navigationspunkt 'mehr'. Dieser kann ebenfalls nicht ausgewählt werden. Grau hinterlegte Menüeinträge führen zu Magazinen der Plattform. Die Internetpräsenz der Magazine soll nicht vom Navigator erfasst werden.

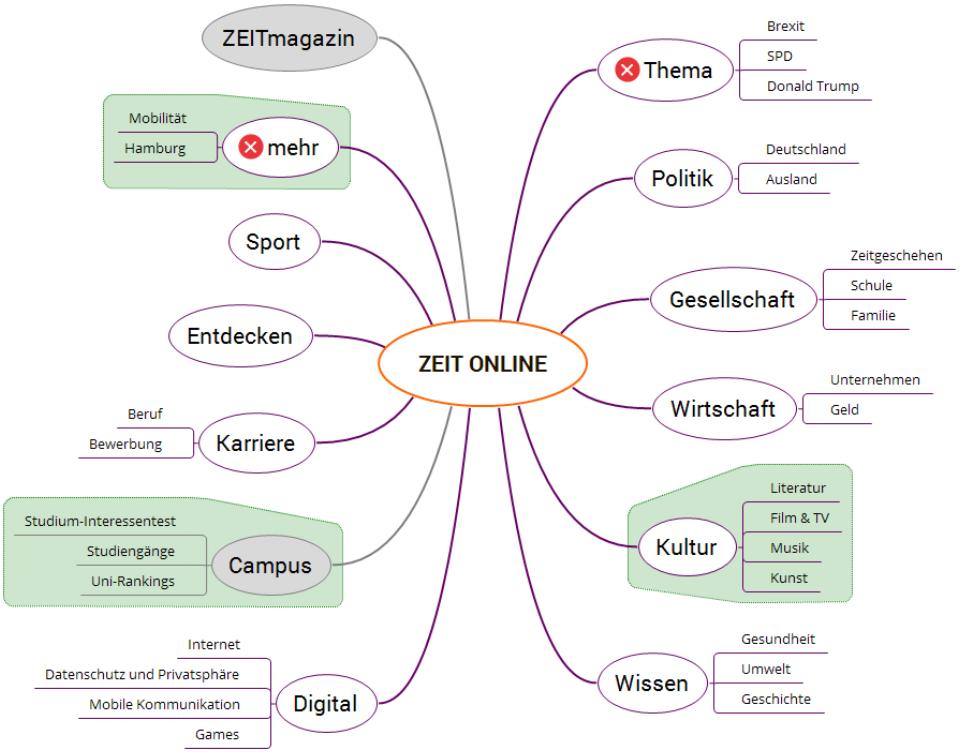


Abb. 12: Zeit Online Struktur der Ressorts und Themenbereiche

Für die Navigator-Anwendung soll nicht jedes Ressort als eigenständige Kategorie dargestellt werden. Eine Aufgabe des Navigators ist es, die Orientierung auf der Seite zu unterstützen. Farblich getrennte Kategorien sollen hierbei helfen. Würde nun jedes Ressorts eine eigene Kategorie darstellen, müssten 13 verschiedene Farben (eine Farbe für den Hauptknoten und jeweils eine Farbe für die Ressorts) ausgewählt werden. Diese Anzahl an Farben würde die Orientierung dann nicht mehr unterstützen, sondern dem Nutzer mit zu vielen Informationen auf einmal erschlagen. Andererseits wären den jeweiligen Kategorien nur verhältnismäßig wenige Knoten zugeordnet. Daher werden die Ressorts für die halbautomatische Klassifizierung innerhalb der Navigator-Anwendung zu folgenden Knoten-Typen zusammen gefasst:

- '`map`' für die Startseite[237]
- '`tags`' für alle Themenseiten
- '`ancient`' für die klassischen Tageszeitungsressorts Politik, Gesellschaft und Wirtschaft

[237] Die Bezeichnung des Knoten-Typs ist auf die ursprüngliche 'analyze'-Pipeline des DITA-Navigators zurück zuführen. Bei diesem wird die DITA-Map als Startseite gerendert.

- '`culture`' für das Ressort Kultur
- '`lifestyle`' für die moderneren Tageszeitungsressort Digital, Karriere und Wissen
- '`generic`' für alle Knoten, die keinem anderen Knotentyp zugeordnet werden können

Der Startseite wird der Knotentyp manuell zugeordnet. Die anderen Knoten-Typen werden wie bei dem DITA-Navigator in einem Template als dynamische Variable definiert. Für alle HTML-Dokumente wird im Rahmen der Verarbeitung zur expandierten DITA-Map die ID aus der absoluten URL des jeweiligen Dokuments erzeugt. Dadurch steht die Bezeichnung des jeweiligen Ressorts immer am Anfang der ID, sodass die Zuordnung des Knoten-Typs auf Basis des `@id`-Attributs der einzelnen `<topicref>`-Elemente erfolgen kann.

3.6 Erweiterung der Anwendung

Der HTML-Navigator setzt sich aus zwei Parts zusammen: das Spider-Modul und die anschließende Erstellung des Navigators. Die Zweiteilung des HTML-Navigators soll ermöglichen, dass er einerseits aus lokalen HTML-Dokumenten erstellt werden kann, andererseits soll er auf beliebige Websites aufgesetzt werden können. Neben dem HTML-Navigator soll ein Prüftool in die Navigator-Anwendung integriert werden.

3.6.1 Entwicklung des Spider-Moduls

3.6.1.1 Basis des Spider-Moduls

Die Entwicklung des Spider-Moduls erfolgte auf Basis des 'JavaScript/Node.js-Webcrawlers' von 'netinstruction'. Dieser Webcrawler soll ein bestimmtes Wort suchen und solange weitere Seiten besuchen, bis er das Wort gefunden hat. [238] Das Spider-Modul hat weiterhin die Kernfunktion des Webcrawlers, dass Websites automatisch besucht und Links eingesammelt werden. Die Funktion zur Suche nach einem bestimmten Wort wurde entfernt. Weitere Funktionen wurden angepasst und ergänzt. Bspw. speichert das Spider-Modul im Gegensatz zum Webcrawler die HTML-Dokumente der besuchten Seiten.

3.6.1.2 Vorbereitung

Für einzelne Aspekte des Spider-Moduls, bspw. HTTP-Anfragen, werden Module eingesetzt, welche diese Schritte vereinfachen. Zur Installation der verschiedenen Module wird eine Paketdatei erstellt.

Die Versionsnummer ist insbesondere für die Veröffentlichung eigener Module relevant. [239] Während der Entwicklung des Projektes wird daher keine Versionsnummer bestimmt. Eine Lizenz wird ebenfalls nicht angegeben, solange das Projekt nicht veröffentlicht wird. Wie in

---

[238] Vgl. netinstructions (Hrsg.): How to make a web crawler in JavaScript / Node.js.
[239] Vgl. npm Documentation: Getting Started, 13 Semantic versioning and npm.

Quellcode 12 dargestellt, werden als Versionsnummern der eingebunden Module jeweils die Versionen angegeben, die während der Entwicklung genutzt wird. Mit dem Zeichen '^' vor der Versionsnummer erhält npm die Information, dass die genannte Version für die Anwendung kompatibel ist. Während der Installation mit npm werden somit die aktuellsten Versionen heruntergeladen. Ausnahme hiervon ist das request-Modul. Dieses wird als feste Version '2.76.0' angegeben, weil das Spider-Modul mit der aktuellsten Version des Moduls nicht zuverlässig ausgeführt werden konnte.

```
{
  "name": "javascript-webspider",
  "version": "0.0.0",
  "description": "JavaScript based Spidertool for HTML-Navigator",
  "author": "Sandra Zitzmann",
  "dependencies": {
    "cheerio": "^0.22.0",
    "url-parse": "^1.1.7",
    "request": "2.76.0",
    "file-system": "^2.2.2"
  }
}
```

Quellcode 12: Paketdatei des Spider-Moduls

3.6.1.3 Variablen des Spider-Moduls

a) Übersicht

Das eigentliche Spider-Modul ist ein JavaScript-Dokument. [240] Im ersten Block dieses Dokuments werden diverse Variablen definiert. Die Variablen können in vier Kategorien eingeordnet werden:

- Variablen zur Einbindung von Node.js-Modulen
- Variablen, deren Werte von der Konsole übergeben werden
- weitere Variablen zur Ausführung des Spider-Moduls
- Debugging-Array

Die Einbindung der Node.js-Module wird mit der `request`-Funktion realisiert. Das Debugging-Array wird als Kategorie genutzt, da es die Funktion des Spider-Moduls nicht beeinflusst, sondern den Prozess und eventuell auftretende Fehler dokumentieren soll. Zu Beginn wird die Variable `debug` als leeres Array initiiert.

b) Variablen mit Wertzuweisung über die Konsole

Anschließend erfolgt die Definition der Variablen `startURL`, `maxPagesToVisit` und `targetDir`. Diesen wird erst mit dem Aufruf über die Konsole ein Wert zugewiesen, indem

[240] Siehe dazu auch Anhang B: JavaScript Code des Spider-Moduls, S. XCII.

die Eingaben aus der Konsole als Array in der globalen Prozessvariable `argv` an das Skript übergeben werden. Daher muss beim Aufruf des Arrays der Index angegeben werden. Die ersten beiden Werte des Arrays sind reserviert[241], sodass ab dem dritten Wert des Arrays der Wert der entsprechenden Variable übergeben wird. Als `startURL` kann eine beliebige URL angegeben werden. Für die Variable `maxPagesToVisit` soll eine Zahl angegeben werden. Dieser Wert ist als Obergrenze für das Spider-Modul gedacht. Es besucht und speichert maximal so viele HTML-Dokumente, wie mit dieser Variable angegeben. Als `targetDir` soll die Bezeichnung für das Verzeichnis angegeben werden, in dem die HTML-Dokumente gespeichert werden sollen. Grundsätzlich wird das Verzeichnis, mit den HTML-Dokumenten eine Verzeichnisebene über dem aktuellen Verzeichnis erstellt.[242] Diese Variablen sollen es ermöglichen, das Spider-Modul in einem beliebigen Umfang oder für andere Websites zu nutzen.

c) Weitere Variablen

In der dritten Kategorien werden Variablen definiert, die nicht durch den Nutzer des Spider-Moduls verändert werden sollen. Die Variable `pagesVisited` enthält zu Beginn ein leeres Objekt, in dem während der Ausführung des Skriptes alle Links hinzugefügt werden, die das Spider-Modul bereits besucht hat. Gleichzeitig wird der Wert der Variable `numPagesVisited` um eins erhöht. Mit dieser Variable soll geprüft werden, dass der in `maxPagesToVisit` angegebene Wert nicht überschritten wird. Daher enthält diese Variable zu Beginn den Zahlenwert null. Mit der Variable `pagesToVisit` wird ein leeres Array eröffnet, in dem Links gesammelt werden, die anschließend besucht werden. Die Variable `currentURL` nutzt das url-parse-Modul, um die angegebene URL in ein Objekt umzuwandeln. Auf zwei Elemente dieses Objekts greift bspw. die Variable `baseURL` zu. Diese ist für das Zeit Online-Beispiel auskommentiert, weil hier alle Links als absolute URL angegeben werden. Erfolgt die Adressierung der Links mit relativen Pfaden, kann die `baseURL` genutzt werden, um eine gültige URL an das `pagesToVisit`-Array zu übergeben.

3.6.1.4 Spiderprozess

Nach der Definition der Variablen beginnt der Spiderprozess, indem die angegebene `startURL` in das `pagesToVisit`-Array übergeben wird. Dem `debug`-Array wird der Hinweis 'Start spidering' und die übergebene `startURL` hinzugefügt.

---

[241] Siehe dazu auch 2.4.3.2 Ausführung des Node-Skripts, S. 46.
[242] Siehe dazu auch 3.6.1.6 Ausgaben des Spider-Moduls, S. 78.

a) Startfunktion

Der Spiderprozess wird mit der zentralen Funktion `spider()` ausgeführt. Diese wird nach Übergabe der Startparameter ausgeführt und überprüft mehrere Situationen mittels konditionaler Abfragen. Abb. 13 stellt die Struktur dieser Abfragen innerhalb der `spider`-Funktion dar. Im folgenden soll [var] als Platzhalter für die Variable `numPagesVisited` stehen. Ist die maximale Anzahl der Seiten, die besucht werden sollen, erreicht, wird dem `debug`-Array die Mel-Meldung 'Done spidering [var] pages.' hinzugefügt. Wird die Funktion beendet, bevor sie die maximale Anzahl der besuchten Seiten erreicht hat, wird die Meldung 'Visited [var] pages. There are no more pages to visit.' an das `debug`-Array übergeben.

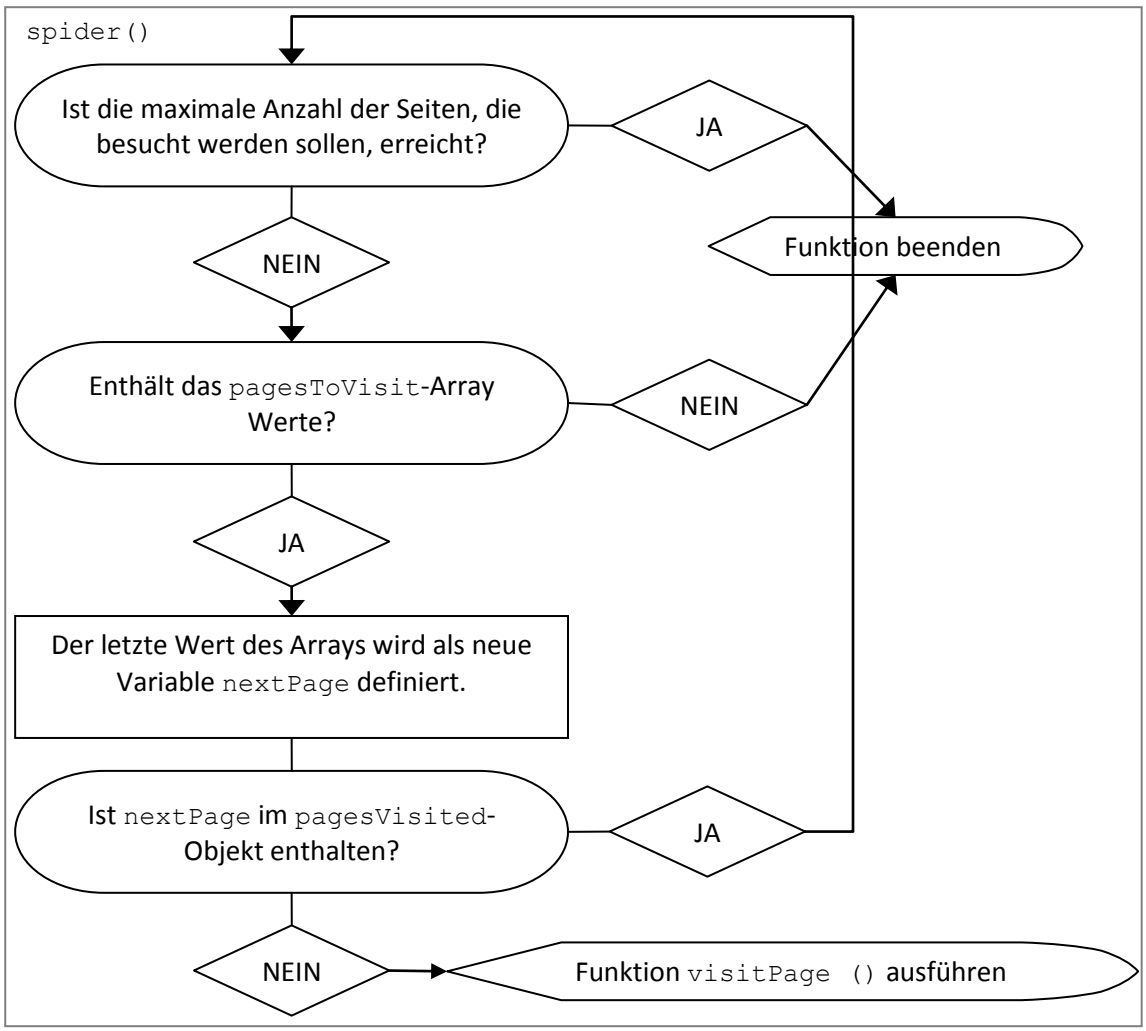


Abb. 13: Struktur der Funktion spider()

b) Auslösen der HTTP-Anfragen

Der Aufruf der `visitPage`-Funktion erfolgt mit zwei Parametern. Als erster Parameter wird mit `visitURL` die URL bestimmt, welche mit dieser Funktion angesprochen werden soll, andererseits muss eine `callback`-Funktion festgelegt werden. Die `spider`-Funktion übergibt die neu definierte Variable `nextPage` als Parameter `visitURL` und sich selbst als callback-Funktion. Innerhalb der `visitPage`-Funktion werden folgende Schritte ausgeführt:

- Die übergeben URL wird dem `pagesVisited`-Array hinzugefügt.
- Der Wert von `numPagesVisited` wird um eins erhöht.
- Eine HTTP-Anfrage an die `visitURL` wird unter Einsatz des http-request-Moduls ausgeführt.
- Die Antwort der HTTP-Anfrage wird verarbeitet, ihr 'body' wird mittels cheerio geladen und als Variable `$` definiert.
- Die Funktion `collectInternalLinks` wird aufgerufen, `$` wird als Parameter übergeben.[243]
- Die Funktion `collectSubLinks` wird aufgerufen, `visitURL` und `$` werden als Parameter übergeben.
- Die Funktion `storeIt` wird aufgerufen, `$.html()` werden als Parameter übergeben.[244]
- Ausführung der callback-Funktion.

Wie in Quellcode 13 dargestellt, werden alle genannten Funktionen innerhalb der HTTP-Anfrage ausgeführt. Bedingung hierfür ist, dass die angesprochene Website den StatusCode 200, '`OK`' als Antwort zurückgibt. Die Anfrage enthält als Antwort im 'body' das gesamte HTML-Dokument in Auszeichnungssprache. Anschließend wird dieser mittels cheerio als Wert der Variable `$` definiert, wodurch ab dieser Stelle die jQuery-Selektoren zur Auswahl von Elementen der HTML-Dokumente genutzt werden können. Die nachfolgenden Funktionen bekommen diese Variable als Parameter übergeben.

```
56    request(visitURL, function(error, response, body) {
57       if(response.statusCode !== 200) {
58         callback();
59         return;
60       }
61
62       var $ = cheerio.load(body);
63       collectInternalLinks($);
64       collectSubLinks (visitURL, $);
65       storeIt(visitURL, $.html());
66       callback();
67    });
```

Quellcode 13: Http-Anfrage innerhalb des Spider-Moduls

[243] Siehe dazu auch 3.6.1.5 Automatische Erkennung der Links, S. 77.
[244] Siehe dazu auch 3.6.1.6 Ausgaben des Spider-Moduls, S. 78.

3.6.1.5 Automatische Erkennung der Links

Die Funktionen zum einsammeln der Links erhalten die Variable $ als Parameter übergeben. Diese Variable enthält den gesamten Inhalt der Seite mit den entsprechenden Markup-Elementen. Die Auswahl der benötigten Funktionen zum einsammeln der Links, ebenso wie die Auswahl der entsprechenden Links muss für jede Website neu angepasst werden. Die Analyse der Seitenstruktur von Zeit Online[245] hat drei Bereiche identifiziert, in denen dauerhafte Querbeziehungen zu anderen Seiten erstellt werden.

Neben der getroffenen Auswahl der internen Querbeziehungen, enthalten die HTML-Dokumente weitere, welche nicht visualisiert werden. Als Plattform, die Nachrichten möglichst sofort präsentiert, ist die Kurzlebigkeit der präsentierten Artikel eine typische Eigenschaft. Jeder den Übersichtsseiten hinzugefügte oder entfernte Link hat Einfluss auf die internen Querbeziehungen. Demzufolge müsste mit jedem dieser Artikel das JSON-Dokument als Datenbasis für den Navigator neu erstellt werden, sodass diese Links vorerst nicht berücksichtigt werden sollen. Durch diese getroffene Einschränkung können nicht pauschal alle Links angesprochen, deren `@href`-Attribut die Zeichenkette '`www.zeit.de`' enthält.

```
70  function collectInternalLinks($) {
71    var internalLinks = $("nav.nav__ressorts li:not([class]) > a[href*='www.zeit.de']");
72    internalLinks.each(function() {
73        debug.push($(this).attr('href'));
74        pagesToVisit.push($(this).attr('href'));
75        //for relative paths use .push(baseURL + $(this).attr('href'))
76      });
77
78    var tagLink = $("header div.nav__tags > a.nav__tag");
79    tagLink.each(function(){
80      debug.push($(this).attr('href'));
81      pagesToVisit.push($(this).attr('href'));
82    });
83
84  }
85
86  function collectSubLinks (visitURL, $) {
87    if (visitURL == startURL) {
88      debug.push("Spider tool is not allowed to collect subLinks at frontpage.");
89    } else {
90      var subLinks = $("div[class='cp-region cp-region--parquet'] div.parquet-meta > a[href*='www.zeit.de']");
91      subLinks.each(function() {
92        debug.push($(this).attr('href'));
93        pagesToVisit.push($(this).attr('href'));
94        //for relative paths use .push(baseURL + $(this).attr('href'))
95      });
96    }
97  }
```

Quellcode 14: Funktionen zur Auswahl der relevanten internen Querbeziehungen

Zwei der Bereiche mit dauerhaften, internen Querbeziehungen sind in jedem HTML-Dokument enthalten. Zur Ansprache der Links wird der Funktion '`collectInternalLinks`' ausschließlich die Variable $ als Parameter übergeben. Wie in Quellcode 14 (Zeilen 71 und 78)

[245] Siehe dazu auch 3.1.3 Zeit Online als Beispiel für HTML-Dokumente, S. 61.

dargestellt, wird innerhalb der Funktion für jeden Bereich, der durchsucht werden soll, eine eigene Variable erstellt. Anschließend wird jeder so gefundenen Link dem `pagesToVisit`-Array hinzugefügt. An dieser Stelle erfolgt keine Überprüfung, ob dieser Link bereits enthalten ist, weil diese nicht zuverlässig ausschließen könnte, dass einzelne Links während des gesamten Spiderprozesses doppelt berücksichtigt würden.
Der dritte Bereich, in dem Links gesucht werden sollen, soll nicht auf der Startseite, sondern nur in den Ressort-Übersichtsseiten durchsucht werden. Daher wird in einer zweiten Funktion '`collectSubLinks`', neben dem Inhalt des HTML-Dokuments als Parameter '`$`', der Parameter '`visitURL`' erwartet. Der zweite Parameter wird mit der definieren '`startURL`' verglichen. Sind diese identisch, wird die Funktion beendet. Wie in Quellcode 14 (oben, Zeile 88) erkennbar ist, wird in diesem Fall zuvor eine Information an das `debug`-Array übermittelt.

3.6.1.6 Ausgaben des Spider-Moduls

Für beide Funktionen gilt: Existiert bereits ein Verzeichnis mit dem angegebenen Namen werden die Dokumente und Unterverzeichnis automatisch in das bestehende Verzeichnis geschrieben. Enthält das Verzeichnis bereits ein Dokument mit gleichen, wie das Dokument, welches in dem Moment erstellt werden soll, wird das alte Dokument automatisch überschrieben.

a) myDebug()

Das Spider-Modul wird als ein erster von mehreren Schritten in eine XProc-Pipeline integriert. Anschließend erfolgt der Aufruf und die Übergabe der Prozessvariablen des Node-Skriptes als Teil einer Pipeline mit dem Calabash-Prozessor. Bei der anschließenden Ausführung des Node-Skriptes werden mit `console.log()` versendete Informationen nicht an die Bash Konsole übergeben. Daher wird das debug-Array genutzt, um alle Informationen zu erhalten, die während der Ausführung normalerweise in der Bash Konsole angezeigt werden sollen. Hierbei wird immer übermittelt, wenn das Skript eine neue Seite anspricht. Alle passenden Links, welche im aktuellen Dokument gefunden wurden, werden darunter aufgelistet. Am Ende des Spiderprozesses erhält das debug-Array zudem eine abschließende Bemerkung, welche entweder dokumentiert, dass die maximale Seitenanzahl erreicht wurde, oder dass alle passenden Seiten bereits besucht wurden.
Der Aufruf der Funktion erfolgt an den Endpunkten der `spider`-Funktion.[246] Als Parameter erhält sie das debug-Array mit allen gesammelten Informationen. Alle Werte des Arrays werden mit der JavaScript-Methode '`join()`' zu einer Zeichenkette zusammengefügt. Hierbei wird bestimmt, dass jeder Wert auf einer neuen Zeile geschrieben wird. Diese Zeichenkette wird als

[246] Siehe dazu auch Abb. 13 in 3.6.1.4a) Startfunktion, S. 75.

zweiter Parameter der Funktion `fs.writeFile()` übergeben. Als erster Parameter wird eine Kombination aus drei Variablen übergeben, welche den Pfad und den Namen des zu erstellenden Textdokuments bestimmen. Für das Textdokument wird relativ zum aktuellen Verzeichnis ein neues Verzeichnis '`debug`' erstellt. Insbesondere im Rahmen der Entwicklung sollten die verschiedenen Textdokumente nicht miteinander überschrieben werden. Daher wird die aktuelle Uhrzeit bei Ausführung des Skriptes als Teil des Dateinamen erzeugt.

b) storeIt()

Damit der HTML-Navigator für die gespiderte Website erstellt werden kann, müssen diverse Schritte durchgeführt werden. Hierfür werden die HTML-Dokumente als lokal gespeicherte Ressourcen benötigt. Den Speichervorgang übernimmt die Funktion `storeIt()`, welche innerhalb der Funktion '`visitPage`' nach den '`collect…Links`'-Funktionen aufgerufen wird. Entsprechend dem Quellcode 15 erwartet die Funktion zwei Parameter. Die Kommentare hinter der Definition der Parameter und Variablen soll die gewählte Abkürzung für Dritte nachvollziehbar machen.

```
111  // Store html-dir
112  function storeIt (curPage, $){              // cur[rent]Page
113    var curURL = new URL(curPage);            // cur[rent]URL
114    var path = curURL.pathname;
115    var storeHTML = '../' + targetDir  + path + '.html';
116    fs.writeFile( storeHTML, $, function(err){
117      if (err) console.log(err);
118    });
119  }
```

Quellcode 15: Funktion zur Speicherung der HTML-Dokumente

Als erster Parameter wird die URL der gerade besuchten Seite benötigt. Mit Hilfe des url-parse-Moduls wird hier erneut eine URL in ein Objekt umgewandelt. Anschließend wird die im Objekt enthaltene Eigenschaft '`pathname`' abgefragt. Die resultierende Zeichenkette ergibt den relativen Pfad der aktuellen Seite. Gemeinsam mit der globalen Variable '`targetDir`', welche beim Aufruf des Skriptes übergeben wurde, und der Zeichenkette '`.html`' wird eine neue Variable erstellt, die als erster Parameter an die Funktion `fs.writeFile()` übergeben wird. Das Skript wird im Verzeichnis 'scripts' ausgeführt. Der Pfad des neuen Verzeichnisses wird innerhalb der Funktion relativ zum aktuellen Verzeichnis angegeben. Grundsätzlich sollen die HTML-Dokumente nicht in dem scripts-Verzeichnis gespeichert werden. Daher wird im Quellcode 15, Zeile 115 als erster Teil des Speicherpfads der Wechsel in das übergeordnete Verzeichnis durchgeführt.

Der zweite Parameter der '`storIt`'-Funktion wird ohne weitere Bearbeitung als zweiter Parameter der '`fs.writeFile`'-Funktion übergeben. Als Wert enthält dieser die Variable '`$`' aus

der '`visitPage`'-Funktion, erweitert um die jQuery-Methode '`.html()`'. In dieser Kombination wird das Wurzelelement des HTML-Dokuments und sein gesamter Inhalt ausgewählt. Somit kann das HTML-Dokument unverändert abgespeichert werden.
Das aus dieser Funktion resultierende Verzeichnis, enthält alle besuchten Seiten und Unterverzeichnisse entsprechend der Struktur der Zeit Online Plattform.

3.6.2 Erstellung des HTML-Navigators

3.6.2.1 Zusammenführung der HTML-Dokumente in eine rekursive Verzeichnisliste

Die gespiderten HTML-Dokumente müssen nicht zwingend XHTML-konform sein. Damit diese in einer XProc-Pipeline weiter verarbeitet werden können, müssen sie daher zunächst zu einem gültigen XHTML-Dokument umgewandelt werden. Hierfür werden alle HTML-Dokumente in einer rekursiven Verzeichnisliste zusammengefasst, aus der anschließend eine expandierte DITA-Map erzeugt wird. Zur Erstellung der rekursiven Verzeichnisliste werden die 'file-uri'-Methode und die 'recursive-directory-list'-Methode des transpect-Moduls 'xproc-util' genutzt. Diese Funktionen werden im transpect-eigenen Namensraum '`tr`' ausgeführt.

a) Rekursive Verzeichnisliste

In der Pipeline 'read-html-dir' wird die 'recursive-directory-list'-Methode des transpect-Moduls ausgeführt. Diese erwartet als Input einen Verzeichnisnamen oder eine URI zu einer Datei, aus welcher die Verzeichnisliste erstellt werden soll. Das Ergebnis dieser Pipeline ist ein XML-Dokument, in dem das Hauptverzeichnis sowie alle Unterverzeichnisse als `<c:dir>`-Element dargestellt werden. Innerhalb dieser Elemente werden alle gefundenen Dateien als `<c:file>`-Elemente erfasst. Jedes `<c:dir>`- und `<c:file>`-Element wird ein Attribut `@name` zugewiesen, in dem die lokale URI des Verzeichnis bzw. der Name der Datei angegeben ist. Des Weiteren wird jedem `<c:file>`-Element der geparste Inhalt des entsprechenden HTML-Dokuments als `<html:html>`-Element untergeordnet. '`c`' und '`html`' sind vom W3C definierte Namensräume.
Innerhalb der Pipeline wird ein XProv-Viewport (siehe Quellcode 16, Zeile 34) eröffnet. In diesem wird für jedes `<c:file>`-Element, dessen `@name`-Attribut mit der Zeichenkette '.html' endet, das Ergebnis der 'file-uri'-Methode importiert.

```
<p:option name="dir"> [2 lines]

<p:output port="result" primary="true"> [5 lines]
<p:serialization port="result" indent="true"/>

<p:import href="http://transpect.io/xproc-util/file-uri/xpl/file-uri.xpl"/>
<p:import
  href="http://transpect.io/xproc-util/recursive-directory-list/xpl/recursive-directory-list.xpl"/>
<p:import href="read-html.xpl"/>

<tr:file-uri name="file-uri" make-unique="false">
  <p:with-option name="filename" select="$dir"/>
</tr:file-uri>

<tr:recursive-directory-list name="list" >
  <p:with-option name="path" select="/*/@local-href"/>
</tr:recursive-directory-list>

<p:viewport match="c:file[ends-with(@name, '.html')]" name="vp">
  <kn:read-html name="rh">
    <p:with-option name="html" select="resolve-uri(/*/@name, base-uri(/*))"/>
  </kn:read-html>
  <p:sink/>
  <p:insert match="/*" position="first-child">
    <p:input port="source">
      <p:pipe port="current" step="vp"/>
    </p:input>
    <p:input port="insertion">
      <p:pipe port="result" step="rh"/>
    </p:input>
  </p:insert>
</p:viewport>
```

Quellcode 16: Pipeline 'read-html-dir'

b) Verarbeitung der einzelnen HTML-Dokumente

Die Pipeline 'read-html' führt die 'file-uri'-Methode des transpect-Moduls aus. Hierbei wird für ein einzelnes HTML-Dokument eine HTTP-Anfrage ausgeführt. Die Antwort dieser Anfrage enthält u. a. den Inhalt des HTML-Dokuments. Unter Angabe des `@content-type`-Attributs wird dieser Inhalt den Syntaxregeln entsprechend umgewandelt, bspw. werden nicht geschlossene Elemente während der Umwandlung geschlossen. Ergebnis dieser Pipeline sind gültige HTML-Dokumente.

3.6.2.2 Umwandlung in eine expandierte DITA-Map

Die Umwandlung der rekursiven Verzeichnisliste in eine expandierte DITA-Map soll die Verwendung der weiteren DITA-Pipelines ermöglichen. Hierfür wird jedes HTML-Dokument, in ein `<topicref>`-Element umgewandelt. Dieses neue Element erhält ein Attribut `@id`. In dem Attribut wird der lokale Pfad des übergeordneten Verzeichnisses mit dem `@name`-Attribut des `<c:file>`-Elements der rekursiven Verzeichnisliste kombiniert und mit den in Quellcode 17 dargestellten Funktionen angepasst, sodass eine ID für jedes `<topicref>`-Element definiert wird. Aus diesen beiden Quellen wird außerdem die URL des HTML-Dokuments rekonstruiert und als Attribut `@href` an das `<topicref>`-Element übergeben.

```
34  <xsl:template match="c:file[html:html]" mode="#default">
35    <topicref>
36      <xsl:attribute name="id"
37        select="concat(
38                  replace(substring-after(base-uri(parent::c:directory),'html-collection//'), '/', '-'),
39                  replace(@name, '\.html$', ''))" />
40      <xsl:attribute name="href"
41        select="concat( 'http://www.zeit.de',
42                        substring-after(base-uri(parent::c:directory),'html-collection/'),
43                        substring-before(@name, '.html'))" />
44      <xsl:apply-templates mode="topic"/>
45    </topicref>
46  </xsl:template>
```

Quellcode 17: Definition der Attribut des <topicref>-Elements

Die resultierende expandierte HTML-DITA-Map weicht strukturell von der expandierten DITA-Map des DITA-Navigators ab. Einerseits bleiben `<dir>`-Elemente erhalten, damit die Verschachtelung der `<topicref>`-Elemente entsprechend wiedergegeben wird. Andererseits wird die DITA-Map in diesem Fall nicht als Index-Seite und Ausgangspunkt des Navigators genutzt. Diese Funktion soll das erste `<topicref>`-Element übernehmen.

Vom Inhalt der HTML-Dokumente sollen nur für den Navigator notwendige Informationen in die DITA-Map übertragen werden. Notwendige Information sind internen Querbeziehungen, der Titel sowie alle Textknoten. Diese werden im allgemeinen DITA-Topic `<topic>` erfasst. Hierbei wird der Titel als `<title>`, alle Textknoten als `<p>`-Elemente im `<body>` des Topics und alle dauerhaften internen Querverbindung unterhalb des `<body>` im `<related-links>`-Container erfasst. Des Weiteren enthält der `<body>` des Topics alle anderen internen Querverbindungen des aktuellen HTML-Dokuments als `<xref>`-Elemente.

3.6.2.3 Erstellung des HTML-Navigators aus den verarbeiteten HTML-Daten

Die Pipeline 'analyze' des DITA-Navigators wird auch bei der Erstellung des HTML-Navigator genutzt. Innerhalb dieser Pipeline erfolgt die Klassifikation der einzelnen Knoten des jeweiligen Navigators. [247] Durch die strukturellen Abweichungen der expandierten HTML-DITA-Map gegenüber der expandierten DITA-Map, müssen darüber hinaus weitere Anpassungen zur Auswahl der Knoten durchgeführt werden. Wie bereits bei dem DocBook-Navigator, werden die Schlüsselwörter des HTML-Navigators als Teil der 'analyze'-Pipeline erzeugt.

In der 'analyze'-Pipeline des DITA-Navigators wird die DITA-Map als Indexseite und Ausgangspunkt des Navigators genutzt. Die gespiderte Website hat bereits eine Indexseite, sodass diese als Ausgangspunkt des Navigators genutzt werden soll. Bereits bei der Umwandlung der Verzeichnisliste in die expandierte HTML-DITA-Map, wurde dem entsprechendem <topicref>-Element als 'index' als ID zugewiesen. Daher kann die Indexseite in der 'analyze'-Pipeline über die ID angesprochen und als Startpunkt des Navigators definiert werden. Der Konten-Name

[247] Siehe dazu auch 3.5.4 HTML, S. 70.

wird, wie bei den anderen <topicref>-Elementen, aus dem <title>-Element des Topics erstellt.

Die Auswahl der Navigator-Links, die aus der hierarchischen Struktur der Map basieren, erfolgt ebenfalls über die IDs der <topicref>-Elemente. Für die <link>-Elemente, die innerhalb der Topics Querbeziehungen zu anderen Topics erstellen, kann als source-Eigenschaft des Navigator-Links ebenfalls die ID des aktuellen <topicref>-Elements genutzt werden. Als target des Navigator-Links wird das @href-Attribut des ursprünglichen <link>-Elements entsprechend angepasst. D. h. die Zeichenkette 'http://www.zeit.de/' wird entfernt und alle '/' werden mit Bindestrichen ersetzt.

Das Ergebnis dieser Pipeline ist die nav-data-XML aus welcher mit der 'jsonify'-Pipeline das JSON-Dokument als Basis für den Navigator erstellt wird.

3.6.3 Überprüfung der Visualisierbarkeit

3.6.3.1 Positionierung des Prüftools

Das Prüftool bewertet die Visualisierbarkeit der Querbeziehungen anhand der Knotenanzahl und der durchschnittlichen Linkanzahl. Es soll für alle Dokumentenformate der Navigator-Anwendung einsetzbar sein. Daher wird die Prüfung der Visualisierbarkeit anhand der nav-data-XML durchgeführt. Dieses XML-Dokument enthält alle Knoten und alle Links, sodass sie unabhängig vom Eingangsdokumentenformat als Basis für das Prüftool geeignet ist.

Die Auswertung soll mittels XSLT erfolgen und als Meldung in der Konsole während der Ausführung des Prozesses zur Erstellung des Navigators ausgegeben werden.

3.6.3.2 Auswertung der Knoten

Als Kriterium für die Visualisierbarkeit von Querbeziehungen in XML-Daten wurde die Anzahl der Knoten festgelegt. Maximal 90 Knoten, mindestens acht Knoten soll das Dokument enthalten, damit eine sinnvolle Darstellung mit dem Navigator möglich ist. Daher soll in einem ersten Schritt des Prüftools die Anzahl der Knoten ermittelt werden. Hierfür wird ein Variable erstellt, welche mit der in XSLT enthaltenen count()-Methode alle <node>-Elemente der nav-data-XML zählt. Der erhaltene Wert wird mit <xsl:message> an die Konsole übergeben. Gleichzeitig er als Parameter an die Funktion zur Berechnung der durchschnittlichen Linkanzahl übergeben werden.

3.6.3.3 Ermittlung der Linkanzahl pro Knoten

Die Basis, für die Berechnung der durchschnittlichen Linkanzahl als Kriterium für die Visualisierbarkeit von Querbeziehungen, bildet eine Datenmenge, bei der jedem Wert x die Anzahl der enthaltenen Links zugeordnet wird. Die Anzahl der Querbeziehungen eines <node> kann dem entsprechend dem Template in Quellcode 18 über dessen ID ermittelt werden.

```
60  <xsl:template match="nodes">
61    <xsl:for-each select="//node">
62      <xsl:value-of select="count(//link[@source = ./@id]) + count(//link[@target = ./@id])" />
63    </xsl:for-each>
64  </xsl:template>
```

Quellcode 18: Template zum Zählen der Links für jeden Knoten

3.6.3.4 Ermittlung und Bewertung der durchschnittlichen Linkanzahl

Anhand der ermittelten Werte soll die durchschnittliche Linkanzahl als geometrisches Mittel berechnet werden. In XSLT können keine Wurzeln n -ten Grades berechnet werden. Daher erfolgt die Umstellung der Formel zur Berechnung des geometrischen Mittels wie in Abb. 14 dargestellt. Für die Berechnung soll n die Anzahl der Knoten sein und jeder Wert x steht für die Anzahl der Links eines Knoten.

$$g = \sqrt[n]{x_1 \cdot x_2 \cdot \ldots \cdot x_n}$$

$$g = (x_1 \cdot x_2 \cdot \ldots \cdot x_n)^{\frac{1}{n}}$$

Abb. 14: Berechnung des geometrischen Mittels ohne Wurzeloperator

Die Berechnung, sowie die anschließende Bewertung konnte im XSLT noch nicht umgesetzt werden.

3.6.3.5 Ausgabe des Ergebnisses

Die Pipeline soll in jedem Navigator vor der Pipeline 'jsonify' integriert werden. Die Auswertung der nav-data-XML soll dabei vollständig an die Konsole übergeben werden. Innerhalb des XProc-Steps soll kein neues Dokument erstellt und die nav-data-XML dem Output-Port mit dem identischen Inhalt übergeben werden, den das Dokument am Input-Port hatte.

Die Ausgabe soll einerseits die ermittelte Knotenanzahl ausgeben. Dabei soll ein Hinweis enthalten sein, falls das Dokument zu wenige oder zu viele Knoten beinhaltet. Der Nutzer soll somit informiert werden, dass die Darstellung des Navigators durch die Anzahl der Knoten negativ beeinflusst wird.

Ein zweiter Teil der Ausgabe in der Konsole soll die Bewertung der ermittelten durchschnittlichen Linkanzahl sein. Hierbei soll ein Prozentsatz Angegeben werden mit der Anmerkung, dass die Datenquelle zu diesem Prozentsatz für eine optimale Visualisierung der Querbeziehungen geeignet ist. Sollten zu wenige oder zu viele Links vorhanden sein, soll auch an dieser Stelle ein Hinweis ausgegeben werden, der den Nutzer darüber informiert, dass zu wenige bzw. zu viele Querbeziehungen innerhalb des Dokuments bestehen.

Das Prüftools konnte im Rahmen dieser Arbeit noch nicht erfolgreich umgesetzt werden.

3.7 Zusammenführung der Komponenten

Für jeden Navigator wurde eine Bash-Skript erstellt, welches die Front-End-Pipeline des jeweiligen Navigators aufruft. Die Optionen der verschiedenen Pipelines können hierbei an die eigenen Dokumente angepasst werden. Quellcode 19 zeigt anhand des DITA-Navigators, wie das Skript als Standardwerte das Beispiel des jeweiligen Dokumentenformats nutzt, falls es ohne Angabe der Optionen ausgeführt wird.

```
1  #!/bin/sh
2  if [ -z "$ditamap"]; then
3      ditamap=source/hierarchy.ditamap
4  fi
5
6  calabash/calabash.sh -o expanded-map=temp.xml -i source="$ditamap" xpl/dita2navigator.xpl
```

Quellcode 19: Aufruf der Front-End-Pipeline als Bash-Skript

Der HTML-Navigator hat zwei Front-End-Pipelines erhalten: eine beginnt bei einem bestehenden Verzeichnis die Navigator-Daten zu erstellen, die zweite Pipeline führt erst das Spider-Modul aus und erstellt anschließend aus dem gespiderten Verzeichnis die Navigator-Daten.

Jeder Navigator greift bis zur Analyse der Dokumente auf individuelle Pipelines zu. Erst nachdem die nav-data-XML erstellt wurde, wird von allen Navigatoren die 'jsonify'-Pipeline des DITA-Navigators verwendet.

4 Auswertung

4.1 Ergebnis

Ziel der Arbeit war einerseits die Definition von Kriterien für die Visualisierbarkeit von Querbeziehungen in XML-Daten. Andererseits sollte der DocBook-Navigator modularisiert und gemeinsam mit dem DITA-Navigator in einer Navigator-Anwendung zusammengeführt werden. Diese Anwendung sollte um einen HTML-Navigator, welcher ein Spider-Modul beinhaltet, und um ein Prüftool zum Abgleich der Eingangsdaten mit den definierten Kriterien für die Visualisierbarkeit erweitert werden. Innerhalb des Navigators sollten die Knoten halbautomatisch klassifiziert werden.

Die Definition der Kriterien erfolgte auf Basis der Millerschen Zahl. Im Zusammenhang mit der Anforderung einer Navigation ist das Konzept jedoch umstritten. Es wurde trotzdem gewählt, weil andere Ansätze zur Definition der Kriterien keine automatisch messbaren Werte ergaben. Daher sind die Kriterien sehr willkürlich bestimmt. Zur Verifizierung der Kriterien müsste eine Datensammlung erstellt werden, bei der verschiedenste Personen die Visualisierbarkeit der Querbeziehungen bewerten. Die Kriterien könnten würden dabei die Basis für die Aufstellung der Testdaten bilden.

Die Bewertungsskala für die optimale Linkhäufigkeit wurde auf Basis von ganzzahligen Linkanzahlen erstellt, obwohl die durchschnittliche Linkanzahl mit dem geometrischen Mittel berechnet wird. Das geometrische Mittel muss anschließend für die Bewertung anhand der Skala auf einen ganzzahligen Wert gerundet werden. Dadurch wird die Bewertung verfälscht. Ein Lösungsansatz hierfür ist es, die Bewertungsskala in eine Funktion umzurechnen. Da die Bewertung auch bei einer hohen durchschnittlichen Linkanzahl nicht den Wert Null ergeben soll, könnte hierfür eine logarithmische Funktion erstellt werden.

Auswertung der Beispiele:

Die Modularisierung des DocBook-Navigators konnte erfolgreich umgesetzt werden. Die Zusammenführung dieses Navigators gemeinsam mit dem DITA-Navigator in eine Navigator-Anwendung wurde dahingehend umgesetzt, dass beide Dokumentenformate in ein XML-Dokument mit identischer Struktur (die nav-data-XML) umgewandelt werden. Ab diesem Dokument greifen beide Navigatoren auf gemeinsame XProc-Schritte, zur Erstellung der Navigator-Daten als JSO-Dokument, zu.

Der HTML-Navigator konnte ebenfalls in diese Anwendung integriert werden. Bereits auf dem Weg zur nav-data-XML kommen dabei XProc-Schritte zum Einsatz, welche auf dem DITA-Navigator basieren und nur geringfügig angepasst werden mussten. Das Spider-Modul des HTML-Navigators wurde erfolgreich auf Basis von JavaScript und Node.js entwickelt.

Die Navigator-Anwendung wurde vollständig mit Open Source Technologien entwickelt.

Die Entwicklung des Prüftools als Teil der Navigator-Anwendung konnte nicht verwirklicht werden. Als Ansatz wurde die Auswertung der nav-data-XML mittels XSLT gewählt. Nachdem dieser bisher nicht zu den gewünschten Ergebnissen geführt hat, ist zu überlegen eine andere Technologie für die Umsetzung des Prüftools auszuwählen. Diese Überlegungen konnten im Rahmen der vorliegenden Arbeit aus zeitlichen Gründen nicht weiter verfolgt werden.

Die Entwicklung der Navigator-Anwendung wurde anhand von drei Beispielen durchgeführt. Insbesondere die Klassifikation ist dadurch auf diese Beispiele zugeschnitten. Andererseits wurde festgestellt, dass die Klassifikation immer nur halbautomatisch erfolgen kann. Die drei Beispiele haben dafür verschiedene Ansätze präsentiert, welche auf andere Dokumente übertragen werden können.

Unter Berücksichtigung der aufgestellten Kriterien für die Visualisierbarkeit von Querbeziehungen in XML-Daten und der durchgeführten Klassifikation hat sich die Zeit Online Website als ungünstiges Beispiel für den Navigator erwiesen. Der Inhalt aktualisiert sich täglich, sodass nicht alle Seiten der Plattform integriert werden können. Gleichzeitig zeigen aber erst die einzelnen Artikelseiten die Verbindung zwischen den Ressorts auf. Bspw. wurde im Dezember ein Artikel[248] veröffentlicht, der dem Ressort Kultur (TV & Film) zuzuordnen ist. In dem Artikel geht es um eine Dokumentation über die Bundeskanzlerin, sodass dieser Artikel auf der Übersichtsseite des Ressort Politik (Deutschland) präsentiert wurde. Solche Querverbindungen konnten mit dem HTML-Navigator nicht abgebildet werden, sodass er in diesem Beispiel nur die hierarchische Struktur der Website wiedergibt. Eine Möglichkeit alle Seiten (und die damit verbundenen Querbeziehungen außerhalb der hierarchischen Struktur) im Navigator abzubilden, wäre eine laufende Aktualisierung des JSON-Dokuments, welches die Basis des Navigators bildet. Dagegen sprechen jedoch die Kriterien für interaktive Visualisierbarkeit von Querbeziehungen. Als Obergrenze wurden 90 Knoten bestimmt, welche von der Plattform voraussichtlich innerhalb von Stunden überschritten würde.

4.2 Offene Anforderungen und Weiterentwicklung

Die beiden Schwerpunkte für die Weiterentwicklung der Navigator-Anwendung sind die Erstellung des Prüftools, sowie die Erstellung einer Konfigurationsdatei, in der bspw. die Angaben zur Klassifizierung der Dokumente erfasst werden können. Bisher wird die Klassifizierung als Teil der 'analyze'-Pipeline des jeweiligen Dokumentenformats durchgeführt.

Innerhalb der Anwendung sollte sprechender Code verwendet werden. Dieser ist an mehreren Stellen noch ausbaufähig. Außerdem sollten Namen im JavaScript in der CamelCase-

[248] http://www.zeit.de/gesellschaft/2016-12/angela-merkel-arte-portraet-fluechtlingskanzlerin

Schreibweise gewählt werden und in XML-Dokumenten sollte die Bindestrich-Schreibweise genutzt werden. Diese Vorgaben wurden noch nicht konsequent umgesetzt. Insbesondere JavaScript-Bezeichnungen könnten verbessert werden. An verschiedenen Stellen werden Variablen unter ihrem Namen als Parameter in einer Funktion übergeben. Dadurch kann der gleiche Name einmal Variable und ein- oder mehrmals Parameter sein, und der Code wird schnell unübersichtlich.

Neben den genannten Schwerpunkten könnte die Navigator-Anwendung noch hinsichtlich der Dokumententypen erweitert werden. Bisher werden als DocBook-Dokumente ausschließelich solche des Typs 'Buch' verarbeitet. Der DITA-Navigator verarbeitet nur das allgemeine Topic, sowie die seit der ersten Version verfügbaren Concept- Task- und Reference-Topics. Im Rahmen der DITA-Version 1.3 wurden weitere spezialisierte Topics standardisiert. Der DITA-Navigator ist außerdem derzeit nur darauf ausgelegt, dass die DITA-Map als Startseite und Ausgangspunkt des Navigators gerendert wird. Die Auswahl eines anderen Topics als Startseite wäre ebenfalls eine Weiterentwicklungsmöglichkeit.

Drüber hinaus könnten einige Aspekte des Spider-Moduls noch verbessert und weiterentwickelt werden. Bisher erfolgt noch kein Error-Handling der HTTP-Angaben. Außerdem sind die erforderlichen Typen der Variablen, die über die Konsole übergeben werden, bisher nur als Kommentare hinterlegt. Hier könnte untersucht werden, inwiefern der erforderliche Typ einer Variable festgelegt werden kann.

Für die gesamte Navigator-Anwendung wurden noch keine ausführlichen Testläufe durchgeführt. Des Weiteren sollte die Verzeichnisstruktur der Navigator-Anwendung überarbeitet werden. Diese ist bisher noch stark auf den DITA-Navigator ausgelegt.

Abschließend lässt sich sagen, dass die Anwendung in Kombination mit passenden Websites großes Potenzial hat die Orientierung auf der Website wesentlich zu erleichtern. Gleichzeitig bietet sie noch vielfältige Weiterentwicklungsmöglichkeiten.

ANHANG

ANHANG A: MAILWECHSEL ZWISCHEN MILLER UND HALPERN

Im folgenden ist der Mailwechsel zwischen George Miller und Mark Halpern zur Gültigkeit der Millerschen Zahl dokumentiert.

From: Mark Halpern
To: George Miller
Subject: citation for your disclaimer

Dear Professor Miller,

The director of the technical writing group which I serve as editor has issued an edict that lists and procedures in our printed and screen-displayed documentation should not exceed seven, or maybe nine at the most, items. This is of course a silly rule, whatever its origin, but I think that its source in this case is a fading memory of a third-hand report of a bad reading of your classic 1956 paper.

Of course you are in no way responsible for the misreadings of your paper, and the silly things done in its name, but I hope you can help my organization, at least, climb back out of the pit it has dug for itself, using your paper as its shovel. I have seen somewhere a quotation from you, or a paraphrase of your words, in which you deplore the strange conclusions that some have drawn from your paper, and express your dismay over all the half-baked rules that people have promulgated, citing it as their authority.

In my attempt to get our director to rescind his bad rule, I would like to be able to quote your very words against him; would you tell me where I might find such words? Or, if what you've said in the past is not on record, could I induce you to say now that nothing in your paper should be taken as warrant for asking Moses to discard at least one, and preferably three, of the Commandments?

With my thanks,
Mark Halpern

From: George Miller
To: Mark Halpern
Subject: Re: citation for your disclaimer

Many years ago landscape architects used my +/-7 paper as a basis to pass local laws restricting the number of items on a billboard. It was funded by the big motel chains; if you run a mom-and-pop motel you have to put a lot of information on your sign, but if you have a franchise everybody knows you have hot and cold running water, color televisions, free breakfasts, etc. The restriction on billiard content was driving the small motels out of business.

The same argument was used in the Lady Bird Johnson Act to prohibit billboards within X feet of highways, and the billiard industry (a strange group that deserves an essay of its own) was hurting. They hired a man to travel around from town to town trying to refute the claims that more than 7 items of information could cause accidents. The man's wife did not like her husband being

constantly on the road, so she asked him about it. He told her that the root of his trouble was some damn Harvard professor who wrote a paper about 7 bits of information. She, being herself a psychologist, said that she did not think that that was what Professor Miller's paper said.

Armed with this insight, he looked me up and told me the whole story about my career, unknown to me, in the billboard industry. There was much more to it than I have outlined here, and I was shocked. So shocked that I wrote a long letter thing to set the record straight. The letter was published in the monthly journal of the billboard industry and that was the end of it. Unfortunately, I no longer have a copy of the letter an I don't recall the name of the journal (this was all back in the early 70s) so I cannot quote to you its contents. But the point was that 7 was a limit for the discrimination of unidimensional stimuli (pitches, loudness, brightness, etc.) and also a limit for immediate recall, neither of which has anything to do with a person's capacity to comprehend printed text.

If you want to quote the original article, it is on line and you can find a pointer to it at www.cogsci.princeton.edu/~wn. But if that is too time consuming - yes, you are right: nothing in my paper warrants asking Moses to discard any of the ten commandments.

Good luck,
g.

From: Mark Halpern
To: George Miller
Subject: RE: citation for your disclaimer

Deaf Professor Miller,

Thank you for full and prompt reply; just what I was hoping for. Armed with your words, I may be able to convince my boss that your paper does not support the notion that if a procedure has fourteen steps, you have to divide it arbitrarily into two procedures, each meaningless in itself.

In being able to quote you against his misinterpretation of your work, I feel like Woody Allen in that movie where he gets into an argument with a pretentious know-it-all about some thesis of Marshall McLuhan's, and has the pleasure Of bringing into the scene the real McLuhan, who tells the know-it-all that he's completely wrong, and Allen is right.

Quelle: http://members.shaw.ca/philip.sharman/miller.txt

ANHANG B: JAVASCRIPT CODE DES SPIDER-MODULS

```
// Node.js packages
var request = require('request');
var cheerio = require('cheerio');
var URL = require('url-parse');
var fs = require('file-system');

// Variables defined at console
var startURL = process.argv[2];          // URL
var maxPagesToVisit = process.argv[3];   // Number
var targetDir = process.argv[4];         /* String with
name of new directory. Will be create on dir up.
                                            Overwrites
existing directory!*/

// Other Variables
var pagesVisited = {};
var numPagesVisited = 0;
var pagesToVisit = [];
var currentURL = new URL(startURL);
//var baseURL = currentURL.protocol + "//" +
currentURL.hostname;
      // activate baseURL, if there are relative link targets

// Debugging-Array
var debug = [];

// Start
pagesToVisit.push(startURL);

debug.push("Start spidering: "  + currentURL + ".")
spider();

// Define functions
function spider() {
  if(numPagesVisited >= maxPagesToVisit) {
    debug.push("Done spidering "  + numPagesVisited + "
pages.");
    myDebug(debug);
```

```
    return;
  }

  if (pagesToVisit.length > 0) {
      var nextPage = pagesToVisit.pop();
      if (nextPage in pagesVisited) {
        spider();
      } else {
          visitPage(nextPage, spider);
        }
    } else{
        debug.push("Visited " + numPagesVisited + "  pages.
There are no more pages to visit." );
        myDebug(debug);
      }
};

function visitPage(visitURL, callback) {
  pagesVisited[visitURL] = true;
  numPagesVisited++;
  debug.push("Visting page " + visitURL)

  request(visitURL, function(error, response, body) {
     if(response.statusCode !== 200) {
       callback();
       return;
     }

     var $ = cheerio.load(body);
     collectInternalLinks($);
     collectSubLinks (visitURL, $);
     storeIt(visitURL, $.html());
     callback();
  });
}

function collectInternalLinks($) {
  var internalLinks = $("nav.nav__ressorts li:not([class])
```

```
>a[href*='www.zeit.de']" );
  internalLinks.each(function() {
      debug.push($(this).attr('href'));
      pagesToVisit.push($(this).attr('href'));
      //for relative paths use .push(baseURL +
$(this).attr('href'))
    });

  var tagLink = $("header div.nav__tags > a.nav__tag" );
  tagLink.each(function(){
    debug.push($(this).attr('href'));
    pagesToVisit.push($(this).attr('href'));
  });

}

function collectSubLinks (visitURL, $) {
  if (visitURL == startURL) {
    debug.push("Spider tool is not allowed to collect
subLinks at frontpage." );
  } else {
    var subLinks = $("div[class='cp-region
cp-region--parquet'] div.parquet-meta >
a[href*='www.zeit.de']");
    subLinks.each(function() {
      debug.push($(this).attr('href'));
      pagesToVisit.push($(this).attr('href'));
      //for relative paths use .push(baseURL +
$(this).attr('href'))
    });
  }
}

// Store several information, that are logged during
spidering
function myDebug (debug){
  var debugPath = '../debug/';
  var debugFileName = 'nodejs.txt';
```

```
  var currentdate = new Date();
  var myDate = 'debug--' +
currentdate.getHours()+currentdate.getMinutes()+currentdat
e.getSeconds();
  console.log("debugPath = " + debugPath + myDate +
debugFileName);
  fs.writeFile(debugPath+myDate+debugFileName,
debug.join('\r\n'), function(err){
     if (err) console.log(err);
   });
}

// Store html-dir
function storeIt (curPage, $){                  // cur[rent]Page
  var curURL = new URL(curPage);                // cur[rent]URL
  var path = curURL.pathname;
  var storeHTML = '../' + targetDir  + path + '.html';
  fs.writeFile( storeHTML, $, function(err){
    if (err) console.log(err);
  });
}
```

ANHANG C: XSLT-STYLESHEET DES XPROC-STEPS 'ANALYZE-DOCBOOK'

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet
     xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
      xmlns:xs="http://www.w3.org/2001/XMLSchema"
      xmlns:kn="http://www.le-tex.de/namespace/kiosknavigator"
   xmlns:letex="http://www.le-tex.de/namespace"
     xmlns:dbk="http://docbook.org/ns/docbook"
  exclude-result-prefixes="xs dbk"
 xpath-default-namespace="http://docbook.org/ns/docbook"
  version="2.0">


  <xsl:template match="/">
    <navigator-data>
      <nodes>
        <xsl:apply-templates select="preface | //part |
//chapter | //author | //othercredit" mode="nodes"/>
      </nodes>
      <links>
        <xsl:variable name="prelim" as="element(*)*">
          <xsl:apply-templates select="preface | //part |
//chapter | //author | //othercredit" mode="links"/>
        </xsl:variable>
        <xsl:for-each-group
select="$prelim[self::*:link][namespace-uri() = '']"
group-by="kn:link-key(.)">
          <xsl:apply-templates select="current-group()[1]"/>
        </xsl:for-each-group>
      </links>
    </navigator-data>
  </xsl:template>

  <xsl:template match="* | @*">
    <xsl:copy>
      <xsl:apply-templates select="@*, node()"/>
    </xsl:copy>
  </xsl:template>

  <xsl:function name="kn:link-key" as="xs:string">
    <xsl:param name="link" as="element(*)*"/>
    <xsl:variable name="sorted" as="xs:string+">
      <xsl:perform-sort select="$link/@source, $link/@target">
```

```
        <xsl:sort select="string(.)"/>
      </xsl:perform-sort>
    </xsl:variable>
    <xsl:sequence select="string-join($sorted, '__________')"/>
  </xsl:function>

  <xsl:variable name="root" select="/"/>
  <xsl:variable name="lang" select="/book/@xml:lang"/>

  <!-- NODES GENERIEREN -->

  <xsl:template match="preface" mode="nodes">
    <node type="home"
          url="{concat('../', $lang, '/index.html')}"
          name="{title}"
          keywords="{letex:generate-keywords(@xml:id, 'home')}">
      <xsl:apply-templates select="." mode="id"/>
    </node>
  </xsl:template>

  <xsl:template match="preface" mode="id">
    <xsl:attribute name="id" select="'index'"/>
  </xsl:template>

  <xsl:template match="author | othercredit" mode="nodes">
    <node type="{letex:node-type(.)}"
          url="{concat('../', $lang, '/',
letex:vertex-id(@xml:id), '.html')}"
          name="{concat(firstname, ' ', surname)}"
          keywords="{letex:generate-keywords(@xml:id,
letex:node-type(.))}">
      <xsl:apply-templates select="." mode="id"/>
    </node>
  </xsl:template>

  <xsl:template match="part | chapter" mode="nodes">
    <node type="{letex:node-type(.)}"
          url="{concat('../', $lang, '/',
letex:vertex-id(@xml:id), '.html')}"
          name="{title}"
          keywords="{letex:generate-keywords(@xml:id,
letex:node-type(.))}">
```

```
      <xsl:apply-templates select="." mode="id"/>
    </node>
  </xsl:template>

  <xsl:template match="text()" mode="nodes"/>

  <xsl:template match="author | othercredit | part | chapter"
mode="id">
    <xsl:attribute name="id" select="letex:vertex-id(@xml:id)"/>
  </xsl:template>

  <xsl:function name="letex:node-type" as="xs:string">
    <xsl:param name="content-item" as="element(*)" />
    <xsl:choose>
      <xsl:when test="$content-item/self::part">
        <xsl:value-of select="'category'"/>
      </xsl:when>
      <xsl:when test="$content-item/parent::part/@xml:id eq
'technology'">
        <xsl:value-of select="'technology'"/>
      </xsl:when>
      <xsl:when test="$content-item/parent::part/@xml:id eq
'products'">
        <xsl:value-of select="'products'"/>
      </xsl:when>
      <xsl:when test="$content-item/parent::part/@xml:id eq
'services'">
        <xsl:value-of select="'services'"/>
      </xsl:when>
      <xsl:when test="$content-item/parent::part/@xml:id eq
'case-studies'">
        <xsl:value-of select="'story'"/>
      </xsl:when>
      <xsl:otherwise>
        <xsl:value-of select="'generic'"/>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:function>

  <xsl:function name="letex:vertex-id" as="xs:string">
    <xsl:param name="id" as="xs:string" />
    <xsl:sequence select="replace($id, '^(Author|Othercredit)-',
```

```
'')" />
  </xsl:function>

  <!-- KEYWORDS FILTERN -->

  <xsl:function name="letex:generate-keywords" as="xs:string">
    <xsl:param name="id" as="xs:string"/>
    <xsl:param name="type" as="xs:string"/>
    <xsl:value-of
      select="distinct-values(tokenize(
                letex:jsonify(normalize-space(string-join(
                  letex:remove-fillwords(
                    if($type eq 'category') then
                      $root//*[@xml:id eq
$id]//text()[not(ancestor::title or ancestor::chapter)]
                    else
                      $root//*[@xml:id eq
$id]//text()[not(ancestor::title)]
                    union
                      $root//title[../@xml:id = $id]//text() ),
                ' '))),
              '\s+'))"/>

  </xsl:function>

  <xsl:function name="letex:jsonify" as="xs:string">
    <xsl:param name="text-to-jsonify"/>
    <xsl:value-of select="letex:replaces( $text-to-jsonify, (
      '\\','\\\\',
      '/', '\\/',
      '&quot;|&#8222;|&#8220;', '\\&quot;',
      '&#xA;','\\n',
      '&#xD;','\\r',
      '&#x9;','\\t',
      '\n','\\n',
      '\r','\\r',
      '\t','\\t',
      ',\s*$','',
      '\s+$','',
      '^\s+','',
      ':$',''
      ) )"/>
```

```
  </xsl:function>

  <xsl:function name="letex:normalize-language-code"
as="xs:string">
    <xsl:param name="lang" as="xs:string"/>
    <xsl:sequence select="replace($lang, '^(\p{Ll}{2,3})(.+)?$',
'$1')"/>
  </xsl:function>

  <xsl:function name="letex:remove-fillwords" as="xs:string*">
    <xsl:param name="with-fillwords" as="node()*"/>
    <xsl:variable name="fillwords-doc" as="document-node()"
     select="doc('fillwords.xml')"/>
    <xsl:sequence
     select="for $i in $with-fillwords
       return tokenize(string(lower-case(replace($i,
'[.,;:!?\(\)]', ''))),'\s+')
              [not(matches(.,
                           concat('^(\d+|[ivx]+|',
                                  string-join($fillwords-doc//w

[parent::fillwords/@lang =
letex:normalize-language-code($lang)],'|'),
                                  ')$'),
                           'x'))]"/>
              <!-- bei normalize-language-code urspr.
($with-fillwords/ancestor::*[@xml:lang][1]/@xml:lang) statt
($lang)-->
  </xsl:function>

  <xsl:function name="letex:replaces">
    <xsl:param name="replace-text" as="xs:string+"/>
    <xsl:param name="search-and-replacement" as="item()+"/>
    <xsl:variable name="replaced" select="replace( $replace-text,
     $search-and-replacement[1],
     $search-and-replacement[2] )"/>
    <xsl:choose>
      <xsl:when test="count( $search-and-replacement ) lt 3">
        <xsl:sequence select="$replaced"/>
      </xsl:when>
      <xsl:otherwise>
        <xsl:sequence select="letex:replaces( $replaced,
```

```
$search-and-replacement[ position() gt 2 ])"/>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:function>

  <!-- LINKS AUFBEREITEN -->

  <xsl:template match="preface" mode="links">
    <xsl:apply-templates select="//part | //chapter | //author |
//othercredit" mode="#current"/>
  </xsl:template>

  <xsl:template match="xref[@linkend]" mode="links">
    <link origin="xref">
      <xsl:attribute name="source"
select="letex:vertex-id(ancestor::*[@xml:id][1]/@xml:id)" />
      <xsl:attribute name="target"
select="letex:vertex-id(@linkend)" />
    </link>
  </xsl:template>

  <xsl:template
match="link[@linkend][not(ancestor::sect1[@xml:id='buchmesse-arc
hive'])]" mode="links">
    <link origin="link">
      <xsl:attribute name="source"
select="letex:vertex-id(ancestor::*[@xml:id][1]/@xml:id)" />
      <xsl:attribute name="target"
select="letex:vertex-id(@linkend)" />
    </link>
  </xsl:template>

  <xsl:template match="text()" mode="links"/>

</xsl:stylesheet>
```

LITERATURVERZEICHNIS

Anderson, Robert D.; Elovirta, Jarno; Fienhold Sheen, Roger W.: DITA Open Toolkit. o. J..
URL: http://www.dita-ot.org/ (Zugriff am 26.01.2017).
[Anderson, Elovirta, Fienhold Sheen: DITA Open Toolkit]

atom (Hrsg.): atom. GitHub-Projekt, Januar 2017.
URL: https://github.com/atom/atom/releases (Zugriff am 04.01.2017).
[atom (Hrsg.): atom]

Atom.io (Hrsg.): Atom. A hackable text editor, o. J..
URL: https://atom.io/ (Zugriff am 04.01.2017).
[Atom.io (Hrsg.): Atom]

Atom.io (Hrsg.): Flight Manual. o. J..
URL: http://flight-manual.atom.io/ (Zugriff am 04.01.2017).
[Atom.io (Hrsg.): Flight Manual]

Baddeley, Alan D.: Word Length and the Structure of Short-Term Memory, in: Journal of Verbal Learning and Verbal Behaviour. Dezember 1975.
URL: https://msu.edu/~ema/802/Ch6-Memory/1/BaddeleyEtAl75.pdf (Zugriff am 20.12.2016).
[Baddeley: Word Length and the Structure of Short-Term Memory]

Beck, Sascha (Hrsg.): Wiki der Informationswissenschaft. Januar 2009.
URL: http://wiki.infowiss.net/Klassifikation (Zugriff am 26.01.2017).
[Beck (Hrsg.): Wiki der Informationswissenschaft]

Bergert, Sven (Hrsg.): Warum DITA sich als Standard durchsetzen wird. Februar 2016.
URL: http://www.tanner.de/infoportal/984/warum-sich-dita-als-standard-durchsetzen-wird/ (Zugriff am 25.01.2017).
[Bergert (Hrsg.) : Warum DITA sich als Standard durchsetzen wird]

Bostock, Mike: Changes in D3 4.0. Juli 2016.
URL: https://github.com/d3/d3/blob/master/CHANGES.md (Zugriff am 22.01.2017).
[Bostock: Changes in D3 4.0]

Bostock, Mike: Data-Driven Documents. Januar 2017.
URL: https://d3js.org/ (Zugriff am 22.01.2017).
[Bostock: Data-Driven Documents]

Bostock, Mike: Force-Directed Graph. Januar 2017.
URL: http://bl.ocks.org/mbostock/f584aa36df54c451c94a9d0798caed35 (Zugriff am 22.01.2017).
[Bostock: Force-Directed Graph]

Cygwin (Hrsg.): Cygwin. Get that Linux Feeling - on Windows, o.J.
URL: https://www.cygwin.com/ (Zugriff am 04.01.2017).
[Cygwin (Hrsg.): Cygwin]

data2type (Hrsg.): Das Konzept von XProc. 2016.
URL: https://www.data2type.de/xml-xslt-xslfo/xproc/xproc-einfuehrung/ (Zugriff am 23.01.2017).
[data2type (Hrsg.): Das Konzept von XProc]

data2type GmbH (Hrsg.): Das Konzept von XProc. 2016.
URL: https://www.data2type.de/xml-xslt-xslfo/xproc/xproc-einfuehrung/ (Zugriff am 04.01.2017).
[data2type (Hrsg): Das Konzept von XProc]

Day, Don; Priestley, Michael; Schell, David: Introduction to the Darwin Information Typing Architecture. Toward portable technical information, September 2005.
URL: http://www.ibm.com/developerworks/xml/library/x-dita1/x-dita1-pdf.pdf (Zugriff am 25.01.2017).
[Day, Priestley, Schell: Introduction to the Darwin Information Typing Architecture]

DITA XML.org (Hrsg.): History of DITA. Oktober 2015.
URL: http://dita.xml.org/book/history-of-dita (Zugriff am 25.01.2017).
[DITA XML.org (Hrsg.) : History of DITA]

DITA XML.org (Hrsg.): What is a DITA map?. September 2007.
URL: http://dita.xml.org/what-dita-map (Zugriff am 25.01.2017).
[DITA XML.org (Hrsg.) : What is a DITA map?]

Eckey, Hans-Friedrich; Kosfeld, Reinhold; Türck, Matthias: Deskriptive Statistik. Grundlagen - Methoden - Beispiele, 5. Aufl., Wiesbaden: Gabler | GWV Fachverlage, 2008.
[Eckey, Kosfeld, Türck: Deskriptive Statistik]

Ecma international (Hrsg.): Standard ECMA-404. Oktober 2013.
URL: http://www.ecma-international.org/publications/standards/Ecma-404.htm (Zugriff am 23.01.2017).
[JSON Spezifikation ECMA-404]

electron (Hrsg.): electron. Dezember 2016.
URL: https://github.com/electron/electron (Zugriff am 04.01.2017).
[electron (Hrsg.): electron]

Elovirta, Jarno; Anderson, Robert D.: DITA Open Toolkit. April 2006.
URL: https://sourceforge.net/projects/dita-ot/files/DITA-OT%20Stable%20Release/DITA%20OT%201.2.1/ (Zugriff am 13.12.2016).
[Elovirta, Anderson: DITA Open Toolkit]

Flanagan, David: JavaScript. The Definitve Guide, 6. Aufl., Sebastopol: O'Reilly Media, 2011.
[Flanagan: JavaScript]

Gauchat, Juan Diego: HTML5, CSS3 & JavaScript. Die neuen Funktionen verstehen und sicher anwenden, Weinheim: WILEY-VCH Verlag, 2013.
[Gauchat: HTML5, CSS3 & JavaScript]

GNU (Hrsg.): GNU Bash. Mai 2016.
URL: http://www.gnu.org/software/bash/bash.html (Zugriff am 04.01.2017).
[GNU (Hrsg.): GNU Bash]

Google (Hrsg.): AngularJS. Januar 2017.
URL: https://angularjs.org/ (Zugriff am 14.01.2017).
[Google (Hrsg.): AngularJS]

Halpern, Mark; Miller, George A.: Citation for your disclaimer. o. J..
URL: http://members.shaw.ca/philip.sharman/miller.txt (Zugriff am 16.12.2016).
[Halpern, Miller: Citation for your disclaimer]

Hessel, Silvia: Lernen mit Medien, in: Niegemann, Helmut M. et al: Kompendium multimediales Lernen. Berlin, Heidelberg: Springer-Verlag, 2008.
[Hessel: Lernen mit Medien]

Hughes-Croucher, Tom; Wilson, Mike: Node. Up and Running, 2012.
URL: http://chimera.labs.oreilly.com/books/1234000001808/index.html (Zugriff am 20.01.2017).
[Hughes-Croucher, Wilson: Node]

Internet Engineering Task Force (Hrsg.): The JavaScript Object Notation (JSON) Data Interchange Format. März 2014.
URL: https://tools.ietf.org/html/rfc7159 (Zugriff am 23.01.2017).
[JSON Spezifikation IETF RFC 7159]

Jache, Alexander: Interaktive Visualisierung von Querbeziehungen in XML-Daten mit D3.js. Bachelorarbeit, März 2016.
URL: http://edoc2.bibliothek.uni-halle.de/lssaoamb/content/titleinfo/56330 (Zugriff am 13.12.2016).
[Jache: Interaktive Visualisierung von Querbeziehungen in XML-Daten mit D3.js]

Key, Michael H.: About SAXON. Version 6.5.5, Nov 2015.
URL: http://saxon.sourceforge.net/saxon6.5.5/ (Zugriff am 04.01.2017).
[Kay: About SAXON]

Krämer, Walter: Statistik für alle. Die 101 wichtigsten Begriffe anschaulich erklärt, Berlin, Heidelberg: Springer-Verlag, 2015.
[Krämer: Statistik für alle]

Langer, Marcel: Grafik und Illustration zur Kommunikation. Stiftdesign, 2015.
URL: http://www.stiftdesign.de/2014/index.html (Zugriff am 26.01.2017).
[Langer: Grafik und Illustration zur Kommunikation]

le-tex publishing services (Hrsg.): transpect. An Open Source framework for converting and checking data, 2016.
URL: http://transpect.github.io/modules-xproc-util.html (Zugriff am 23.01.2017).
[le-tex publishing services (Hrsg.): transpect]

le-tex publishing services: Content Engineering. 2016.
URL: http://www.le-tex.de/de/index.html (Zugriff am 13.12.2016).
[le-tex]

MDN (Hrsg.): Neu in JavaScript 1.8.5. April 2015.
URL: https://developer.mozilla.org/de/docs/Web/JavaScript/Neu_in_JavaScript/1.8.5 (Zugriff am 23.01.2017).
[MDN (Hrsg.): Neu in JavaScript 1.8.5]

Miller, George A.: The Magical Number Seven, Plus or Minus Two. Some Limits on Our Capacity for Processing Information, Mai 1955.
URL: https://www.google.de/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&cad=rja&uact=8&ved=0ahUKEwjV64L_rPjQAhWGPRQKHVK9CWsQFggdMAA&url=http%3A%2F%2Fwww.psych.utoronto.ca%2F~peterson%2Fpsy430s2001%2FMiller%2520GA%2520Magical%2520Seven%2520Psych%2520Review%25201955.pdf&usg=AFQjCNFc0jBTWBLspBBsq-cFpxSGYvbrpA&sig2=YIz6dtWrdNgDRIXh1qdR-w&bvm=bv.141536425,d.d24 (Zugriff am 16.12.2016).
[Miller: The Magical Number Seven, Plus or Minus Two]

Mintert, Steffen: Hilf, wenn Du kannst. Wozu JavaScript-Bibliotheken gut sind, Juli 2009.
URL: https://www.heise.de/ix/artikel/Hilf-wenn-Du-kannst-794658.html (Zugriff am 14.01.2017).
[Mintert: Hilf, wenn Du kannst]

Mueller, Matthew: Cheerio GitHub Repository. Oktober 2016.
URL: https://github.com/cheeriojs/cheerio (Zugriff am 21.01.2017).
[Mueller: Cheerio GitHub Repository]

Mueller, Matthew: Cheerio. o. J..
URL: https://cheerio.js.org/ (Zugriff am 21.01.2017).
[Mueller: Cheerio]

netinstructions (Hrsg.): How to make a web crawler in JavaScript / Node.js. November 2015.
URL: http://www.netinstructions.com/how-to-make-a-simple-web-crawler-in-javascript-and-node-js/ (Zugriff am 16.09.2016).
[netinstructions (Hrsg.): How to make a web crawler in JavaScript / Node.js]

Node.js Foundation (Hrsg.): Node.js Long-term Support Working Group. Januar 2017.
URL: https://github.com/nodejs/LTS (Zugriff am 19.01.2017).
[Node.js Foundation (Hrsg.): Node.js Long-term Support Working Group]

Node.js Foundation (Hrsg.): Node.js v6.9.4 Documentation. 2016.
URL: https://nodejs.org/dist/latest-v6.x/docs/api/ (Zugriff am 19.01.2017).
[Node.js Documentation]

Node.js Foundation (Hrsg.): Node.js. Januar 2017.
URL: https://nodejs.org/en/ (Zugriff am 19.01.2017).
[Node.js Foundation (Hrsg.): Node.js]

npm (Hrsg.): file-system. Strengthen the ability of file system, November 2016.
URL: https://www.npmjs.com/package/file-system (Zugriff am 26.01.2017).
[npm (Hrsg.): file-system]

npm (Hrsg.): npm Version 4.0.3 Documentation. Getting Started, November 2016.
URL: https://docs.npmjs.com/getting-started/using-a-package.json (Zugriff am 15.12.2016).
[npm Documentation]

npm (Hrsg.): request. Simplified HTTP request client, o. J..
URL: https://www.npmjs.com/package/request (Zugriff am 22.01.2017).
[npm (Hrsg.): request]

npm (Hrsg.): url-parse. o. J..
URL: https://www.npmjs.com/package/url-parse (Zugriff am 22.01.2017).
[npm (Hrsg.): url-parse]

o. V.: Introducing JSON. o. J..
URL: http://json.org/ (Zugriff am 23.01.2017).
[Introducing JSON]

o.V.: file-system. November 2016.
URL: https://github.com/douzi8/file-system (Zugriff am 26.01.2017).
[file-system (GitHub)]

OASIS (Hrsg.): Darwin Information Typing Architecture (DITA) Version 1.3 Part 0. Overview, Dezember 2015.
URL: http://docs.oasis-open.org/dita/dita/v1.3/os/part0-overview/dita-v1.3-os-part0-overview.html (Zugriff am 25.01.2017).
[DITA 1.3 Overview]

OASIS (Hrsg.): Darwin Information Typing Architecture (DITA) Version 1.3 Part 3. All-Inclusive Edition, Dezember 2015.
URL: http://docs.oasis-open.org/dita/dita/v1.3/os/part3-all-inclusive/dita-v1.3-os-part3-all-inclusive.html (Zugriff am 25.01.2017).
[DITA 1.3 Spezifikation]

OASIS (Hrsg.): The DocBook Schema Version 5.0. November 2009.
URL: http://docs.oasis-open.org/docbook/specs/docbook-5.0-spec-os.html (Zugriff am 25.01.2017).
[DocBook 5.0 Spezifikation]

Ramey, Chet; Fox, Brian: Bash Reference Manual. Reference Documentation for Bash Edition 4.4, September 2016.
URL: http://www.gnu.org/software/bash/manual/ (Zugriff am 04.01.2017).
[Ramey, Fox: Bash Reference Manual]

Resig, John; Bibeault, Bear: Geheimnisse eine JavaScript-Ninjas. 1. Aufl., Heidelberg: mitp Verlag, 2014.
[Resig, Bibeault: Geheimnisse eine JavaScript-Ninjas]

Rogers, Mikael: request. Dezember 2016.
URL: https://github.com/request/request (Zugriff am 22.01.2017).
[Rogers: request]

Saxonica (Hrsg.): About Saxon. o.J.
URL: http://www.saxonica.com/documentation/index.html (Zugriff am 04.01.2017).
[Saxonica (Hrsg.): About Saxon]

Saxonica (Hrsg.): About Saxon-JS. Dezember 2016.
URL: http://www.saxonica.com/saxon-js/documentation/index.html (Zugriff am 23.01.2017).
[Saxonica (Hrsg.): About Saxon-JS]

Saxonica (Hrsg.): Our Products. 2016.
URL: http://www.saxonica.com/products/products.xml (Zugriff am 04.01.2017).
[Saxonica (Hrsg.): Our Products]

Saxonica (Hrsg.): Saxon 9.7 product comparison (Feature Matrix). 2016.
URL: http://www.saxonica.com/products/feature-matrix-9-7.xml (Zugriff am 04.01.2017).
[Saxonica (Hrsg.): Saxon 9.7 product comparison (Feature Matrix)]

Saxonica (Hrsg.): Saxon-JS. Dezember 2016.
URL: http://www.saxonica.com/saxon-js/index.xml (Zugriff am 23.01.2017).
[Saxonica (Hrsg.): Saxon-JS]

Schoch, Rolf: The Magical Number Seven. Ein klassischer Artikel aus der Psychologie und seine Kosequenzen für die Methodik der Umfrageforschung, Februar 2015.
URL: http://www.vsms-asms.ch/de/medien-medienmitteilungen/bucher-artikel/magical-number-seven/ (Zugriff am 13.12.2016).
[Schoch: The Magical Number Seven]

Schraitle, Thomas: DocBook-XML. Medienneutrales und plattformunabhängiges Publizieren, 2009.
URL: https://www.data2type.de/xml-xslt-xslfo/docbook/querverweise-und-links/interne-querverweise/ (Zugriff am 25.01.2017).
[Schraitle: DocBook-XML]

SELFHTML (Hrsg.): JavaScript. Dezember 2016.
URL: https://wiki.selfhtml.org/wiki/JavaScript (Zugriff am 11.01.2017).
[SELFHTML (Hrsg.): JavaScript]

SELFHTML (Hrsg.): jQuery. November 2016.
URL: https://wiki.selfhtml.org/wiki/JQuery (Zugriff am 14.01.2017).
[SELFHTML (Hrsg.): jQuery]

SELFHTML (Hrsg.): XML. September 2016.
URL: https://wiki.selfhtml.org/wiki/XML (Zugriff am 23.01.2017).
[SELFHTML (Hrsg.): XML]

SELFHTML (Hrsg.): XML/XSL/XPath. September 2016.
URL: https://wiki.selfhtml.org/wiki/XML/XSL/XPath (Zugriff am 23.01.2017).
[SELFHTML (Hrsg.): XML/XSL/XPath]

SELFHTML (Hrsg.): XML/XSL/XSLT. September 2016.
URL: https://wiki.selfhtml.org/wiki/XML/XSL/XSLT (Zugriff am 23.01.2017).
[SELFHTML (Hrsg.): XML/XSL/XSLT]

SyncRO Soft SRL (Hrsg.): Oxygen XML Editor. 2016.
URL: https://www.oxygenxml.com/xml_editor.html (Zugriff am 04.01.2017).
[SyncRO Soft (Hrsg.): Oxygen XML Editor]

The jQuery Foundation (Hrsg.): jQuery API Documentation. 2017.
URL: http://api.jquery.com/ (Zugriff am 22.01.2017).
[The jQuery Foundation (Hrsg.): jQuery API Documentation]

unshift.io (Hrsg.): url-parse. Januar 2017.
URL: https://github.com/unshiftio/url-parse (Zugriff am 22.01.2017).
[unshift.io (Hrsg.): url-parse]

W3C (Hrsg.): Extensible Markup Language (XML) 1.0 (Fifth Edition). November 2008.
URL: https://www.w3.org/TR/xml/ (Zugriff am 23.01.2017).
[XML Spezifikation]

W3C (Hrsg.): HTML & CSS. o. J..
URL: https://www.w3.org/standards/webdesign/htmlcss (Zugriff am 25.01.2017).
[W3C (Hrsg.): HTML & CSS]

W3C (Hrsg.): HTML 5.1. November 2016.
URL: https://www.w3.org/TR/html51/ (Zugriff am 25.01.2017).
[HTML5.1 Spezifikation]

W3C (Hrsg.): HTML5. A vocabulary and associated APIs for HTML and XHTML, Oktober 2014.
URL: https://www.w3.org/TR/html5/ (Zugriff am 25.01.2017).
[HTML5 Spezifikation]

W3C (Hrsg.): Selectors Level 3. W3C Recommendation, September 2011.
URL: https://www.w3.org/TR/selectors/ (Zugriff am 19.01.2017).
[CSS3 Selektoren Spezifikation]
W3C (Hrsg.): The Extensible Stylesheet Language Family (XSL). November 2016.
URL: https://www.w3.org/Style/XSL/ (Zugriff am 23.01.2017).
[W3C (Hrsg.): XSL]
W3C (Hrsg.): What is CSS?. August 2016.
URL: https://www.w3.org/Style/CSS/ (Zugriff am 19.01.2017).
[W3C (Hrsg.): What is CSS?]
W3C (Hrsg.): XHTML™ 1.0 The Extensible HyperText Markup Language. Secon Edition, August 2002.
URL: https://www.w3.org/TR/xhtml1/#xhtml (Zugriff am 25.01.2017).
[XHTML Spezifikation]
W3C (Hrsg.): XML Path Language (XPath) 3.0. Januar 2017.
URL: https://www.w3.org/TR/xpath-30/ (Zugriff am 23.01.2017).
[XPath Spezifikation]
W3C (Hrsg.): XPath and XQuery Functions and Operators 3.0. April 2014.
URL: https://www.w3.org/TR/xpath-functions-30/ (Zugriff am 23.01.2017).
[XPath/XQuery Functinos and Operators 3.0 Spezifikation]
W3C (Hrsg.): XProc. Mai 2010.
URL: https://www.w3.org/TR/xproc/ (Zugriff am 23.01.2017).
[XProc Spezifikation]
W3C (Hrsg.): XQuery 3.0. An XML Query Language, April 2014.
URL: https://www.w3.org/TR/xquery-30/ (Zugriff am 23.01.2017).
[XQuery Spezifikation]
W3C (Hrsg.): XSL Transformations (XSLT) Version 1.0. November 1999.
URL: https://www.w3.org/TR/xslt (Zugriff am 23.01.2017).
[XSLT 1.0 Spezifikation]
W3C (Hrsg.): XSL Transformations (XSLT) Version 2.0. Januar 2007.
URL: https://www.w3.org/TR/xslt20/ (Zugriff am 23.01.2017).
[XSLT 2.0 Spezifikation]
W3C (Hrsg.): XSL Transformations (XSLT) Version 3.0. November 2015.
URL: https://www.w3.org/TR/xslt-30/ (Zugriff am 23.01.2017).
[XSLT 3.0 Spezifikationskandidat]
W3CSchools (Hrsg.): JS AJAX. o. J..
URL: http://www.w3schools.com/js/js_ajax_intro.asp (Zugriff am 22.01.2017).
[W3CSchools (Hrsg.): JS AJAX]
W3CSchools (Hrsg.): JS Tutorial. o. J..
URL: http://www.w3schools.com/js/default.asp (Zugriff am 23.01.2017).
[W3CSchools (Hrsg.): JS Tutorial]
W3CSchools (Hrsg.): XML Tutorial. o. J..
URL: http://www.w3schools.com/xml/ (Zugriff am 25.01.2017).
[W3CSchools (Hrsg.): XML Tutorial]
W3CSchools (Hrsg.): XQuery Tutorial. o. J..
URL: http://www.w3schools.com/xml/xquery_intro.asp (Zugriff am 23.01.2017).
[W3CSchools (Hrsg.): XQuery Tutorial]
Walsh, Norman: DocBook 5. The Definitive Guide, Oktober 2011.
URL: http://tdg.docbook.org/tdg/5.0/docbook.html (Zugriff am 24.01.2017).
[Walsh: DocBook 5]
Walsh, Norman: DocBook. September 2016.
URL: http://docbook.org/ (Zugriff am 24.01.2017).
[Walsh: DocBook]

Walsh, Norman: XML Calabash. Documentation, Mai 2015.
URL: http://xmlcalabash.com/docs/ (Zugriff am 04.01.2017).
[Walsh: XML Calabash]
Walsh, Norman: XML Processing. Oktober 2016.
URL: http://xproc.org/ (Zugriff am 23.01.2017).
[Walsh: XML Processing]
Weisstein, Eric (Hrsg.): Wolfram MathWorld. Februar 2016.
URL: http://mathworld.wolfram.com/ (Zugriff am 13.12.2016).
[Weisstein (Hrsg.): Wolfram MathWorld]
Wenz, Christian: JavaScript. Das umfassende Handbuch, 10. Aufl., Bonn: Galileo Press, 2010.
[Wenz: JavaScript]
Wenz, Christian: JavaScript. Grundlagen, Programmierung, Praxis, 11. Aufl., Bonn: Galileo Press, 2014.
[Wenz: JavaScript]
WHATWG (Hrsg.): HTML. Living Standard, Januar 2017.
URL: https://html.spec.whatwg.org/multipage/ (Zugriff am 25.01.2017).
[WHATWG (Hrsg.): HTML]
Wolf, Jürgen: HTML5 und CSS3. Das umfassende Handbuch, Bonn: Rheinwerk Verlag, 2015.
[Wolf: HTML5 und CSS3]
Zeitverlag Gerd Bucerius GmbH & Co. KG (Hrsg.): Marken & Produkte. o. J..
URL: http://www.zeit-verlagsgruppe.de/marken-und-produkte/ (Zugriff am 26.01.2017).
[Zeitverlag Gerd Bucerius GmbH & Co. KG (Hrsg.): Marken & Produkte]
Zitzmann, Sandra: DITA-Navigator. Documentation, September 2016.
URL: https://subversion.le-tex.de/common/sandbox/dita2navigator/doc/resource/index.html (Zugriff am 13.12.2016).
[DITA-Navigator Documentation]
Zitzmann, Sandra: DITA-Navigator. September 2016.
URL: https://subversion.le-tex.de/common/sandbox/dita2navigator/resource/index.html (Zugriff am 13.12.2016).
[DITA-Navigator]

EIDESSTATTLICHE VERSICHERUNG

Hiermit versichere ich an Eides statt, dass ich die Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe.

Sandra Zitzmann
Leipzig, den 02. Februar 2017