Generierung von XSL-Visualisierungen für TEI-Dokumente in Drupal

Diplomarbeit
im Diplomstudiengang Informatik am Fachbereich 16 Elektrotechnik/Informatik,
vorgelegt am 6. Oktober 2010 von

Roman Kominek

Erstgutachter: Prof. Dr. Lutz Wegner
Zweitgutachter: Prof. Dr. Albert Zündorf

Erklärung

Ich versichere hiermit, dass ich die Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Kassel, den

_____

Vorwort

Das Ziel dieser Diplomarbeit ist es, die Handhabung von TEI-Dokumenten, in Verbindung mit einem Content-Management-System zu vereinfachen. Als Grundlage dienen Ergebnisse, die in einem vorangegangen Projekt, welches die Konvertierung und Darstellung von TEI-Dokumenten mit Hilfe von XSL-Transformation zum Thema hatte, erarbeitet wurden.

Der Schwerpunkt liegt auf dem Publizieren der Dokumente und deren Darstellung. Das Thema ergab sich aus dem Feedback zum beendeten Projekt, das von Dr. des. Christof Schöch geleitet und von Prof. Dr. Lutz Wegner und Dipl.-Inform. Dipl.-Math. Sebastian Pape betreut wurde, bei denen ich mich an dieser Stelle für die Rückmeldungen, Anregungen und die Betreuung bedanken möchte.

Roman Kominek
Kassel, im Oktober 2010

Inhaltsverzeichnis

1 Einleitung

1.1 Vorarbeit

Das Diplomthema dieser Arbeit ging aus einem vorangegangenen, interdisziplinären Projekt mit dem Fachbereich 2 der Universität Kassel hervor. Das Projekt „Visualisierung mit XSL für den Bereich Textedition“, wurde betreut von Prof. Dr. Lutz Wegner und Dipl.-Inform. Dipl.-Math. Sebastian Pape vom Fachbereich 16 Praktische Informatik, dessen Projektleiter Dr. des. Christof Schöch, wissenschaftlicher Mitarbeiter am Institut für Romanistik der Universität Kassel war. Die Aufgabe bestand darin, ein XSL-Stylesheet zu programmieren, welches die Darstellung von TEI-XML-Dokumenten in einem Content-Management-System ermöglichen sollte. Genauer, ging es um ein Editionsprojekt von Christoph Schöch, in dessen Rahmen das Buch von François-Joseph Bérardier de Bataut, „Essai sur le récit“, von 1776, in elektronischer Form zugänglich gemacht werden sollte. Dabei sollten die Vorteile einer digitalen Publikation, unter Beachtung von Layout- und Visualisierungsanforderungen, genutzt werden. Das Buch lag kapitelweise im TEI-XML-Format vor. Das TEI-Format, dient der Kodierung von gedruckten Werken. Im Gegensatz zu proprietären Textverarbeitungssystemen und deren unzähligen Dateiformaten, bietet es zahlreiche Vorteile. Sprach- und Literaturwissenschaftler sehen sich mit dem Problem konfrontiert, dass es zwar viele Programme zur Textedition gibt, aber auch viele davon, von Version zu Version inkompatibel werden oder der Hersteller der Software, diese nicht mehr weiter entwickeln will oder kann. Da die Arbeit von Philologen darin besteht, einen möglichst authentischen Text herzustellen, ist es wichtig, dass das verwendete Format diese Möglichkeiten auch widerspiegelt, da sonst entscheidende Texteigenschaften verloren gehen. Die Darstellung der Dokumente, sollte in einem Content-Management-System (CMS) erfolgen. Entschieden hat man sich im Laufe des Projekts für die Darstellung im CMS „Drupal“, welches die nötige XML-Unterstützung mitbrachte, beziehungsweise durch ein Zusatzmodul, um diese Fähigkeit erweitert werden konnte. Das Ergebnis dieses Projekts waren TEI-Dokumente, dargestellt im Browser mit Hilfe von Drupal, erweitert um optische Hervorhebungen und dynamische Inhalte. Die dynamischen Inhalte wurden in JavaScript umgesetzt. Sie ergänzten die Anzeige zum Beispiel durch eine kleine Index-Funktion, die es erlaubte Seiten eines Kapitels direkt anzuspringen. Auch gab es die Möglichkeit Randnotizen platzsparend darzustellen, indem sie anfangs „zugeklappt“ dargestellt und erst durch ein Klicken auf entsprechende Marker im Text aufgeklappt wurden. So war es dadurch möglich, Notizen und Anmerkungen an den betreffenden Stellen zu platzieren, ohne den Textfluss zu stören, aber auch ohne gezwungen zu sein, zu einer anderen Stelle, wie zum Beispiel einer Fußnote zu navigieren, um eine Anmerkung zu lesen. Bestimmte Textpassagen wie Gedichte, Zitate oder Blockzitate wurden entsprechend optisch aufbereitet. Zusätzlich gab es noch eine Umschaltfunktion für verschiedene Textversionen, wobei zwischen Originaltext und einer korrigierten Fassung umgeschaltet werden konnte. Das Thema dieser Diplomarbeit, die Generierung und damit, sowohl die Automatisierung der Erzeugung, als auch die Manipulation der Darstellung, der XSL-Visualisierungen, schließt somit nahtlos an das Projekt an und ist eine Fortführung dessen.

1.2 Aufgabenstellung

Die im Projekt erreichten Ziele erfüllten zwar ihren Zweck, waren in der Handhabung nicht sehr intuitiv. Besonders in Anbetracht der Tatsache, dass die Zielgruppe des Endprodukts im Administrieren von Content-Management-Systemen eher unerfahren ist. Vor allem Anpassungen per Hand, erfordern Kenntnis diverser Auszeichnungs- und teilweise sogar Skriptsprachen. So kommt es zum Beispiel auch vor, dass nach einer Aktualisierung von Drupal einige Teile wie die Seitensuchfunktion, nicht mehr funktioniert. Das liegt an der Art, wie die externen Dateien für die Darstellung ins Dokument eingebunden werden. Für einen unerfahrener Benutzer, ist das Identifizieren des Problems nicht einfach und dessen Lösung nicht auf Anhieb erkennbar.

Es galt nun also zu entscheiden, wie die Ergebnisse des Vorprojekts in ein Komplettpaket zusammengeschnürt werden konnten, ohne dass der Anwender mühselig einzelne Stylesheet-Dateien, Zusatzmodule und ähnliche Komponenten zusammensuchen muss. Das Einstellen von Dokumenten sollte so einfach wie möglich ablaufen und auch die Anpassung der Darstellung sollte ohne Eingriffe in die Drupal-Installation von statten gehen. Daraus entstand die Idee, die Funktionalität in einem Modul zusammenzufassen, so dass das Publizieren von Dokumenten in einem Schritt erfolgen und die Anpassung der Darstellung, über eine simple Oberfläche stattfinden kann. Das Ziel dieser Diplomarbeit, soll die Automatisierung der im Vorprojekt erreichten Ziele sein und die Implementierung von Ergänzungen, die aus Rückmeldungen und Erfahrungen resultierten.

Im Folgenden wird die Anwendung erläutert und die Arbeitsschritte und Abläufe näher erklärt. Danach folgt ein Überblick über den Entwicklungsprozess und die Implementierungsphase, die Vorgehensweise und eventuelle Probleme, die bei der Umsetzung der ausgearbeiteten Konzepte aufgetreten sind. Zum Schluss wird noch ein Resümee gezogen und die erreichten Ziele bewertet.

2 Grundlagen und Konzept

2.1 Die Komponenten

Eine zentrale Komponente stellen die TEI-Dokumente dar. Das TEI-Format basiert auf XML, der Extensible Markup Language. Dabei steht TEI für Text Encoding Initiative [29] und ist der Name einer 1987 entstandenen Organisation. Das gleichnamige Format wurde 1988 vorgestellt. Das Format basierte zunächst auf SGML der „Standard Generalized Markup Language“, bis zur Version P4 („proposal 4“) im Jahr 2002, welche dann XML als Grundlage verwendete und 2007 durch P5 nochmals aktualisiert wurde [3].

TEI richtet sich an Literaturwissenschaftler, Linguisten und andere Personen die Texte bearbeiten, um diese dann in einem Format zu speichern, das nicht an ein bestimmtes Programm oder einen bestimmten Hersteller gebunden ist und dabei die Vorteile elektronischer Textbearbeitung bietet. So lassen sich elektronische Texte komfortabel durchsuchen und die Hervorhebung bestimmter Abschnitte und Passagen ist auch noch im Nachhinein möglich. Durch bestimmte Auszeichnungen im Text lassen sich unter Anderem verschiedene Textvarianten speichern, wie zum Beispiel im vorliegenden Fall, indem eine Originalfassung und eine bearbeitete korrigierte Fassung im selben Dokument zur Verfügung steht.

Die in dieser Diplomarbeit behandelten Dokumente verwenden eine Teilmenge an Elementen des TEI-Lite-Standards. TEI-Lite entstand zeitgleich mit Version P4 des TEI-Standards und wurde auf ein Subset dessen reduziert.

Eine Weitere Komponente ist das Content-Management-System oder kurz CMS. Ein CMS ist ein Inhaltsverwaltungssystem, das es Benutzern erlaubt Inhalte im Internet zu publizieren, ohne sich mit der zugrundeliegenden Technik, wie Programmiersprachen oder Webservern, auseinandersetzen zu müssen.

Die wohl bekanntesten Systeme sind Drupal, TYPO3 und Joomla [28]. Bei dem hier verwendeten CMS handelt es sich um Drupal. Drupal wurde in der Programmiersprache PHP geschrieben und im Jahr 2001 veröffentlicht. Es wurde vom belgischen Informatiker Dries Buytaert ins Leben gerufen. Drupal ist quelloffen und unter der GNU General Public License (GPL) veröffentlicht worden, ist somit frei verfügbar und darf unter Beachtung der Lizenz weiterentwickelt und abgeändert werden. Als Unterbau werden verschiedene Datenbanken unterstützt, darunter das weitverbreitete MySQL und PostgreSql aber auch kommerzielle Produkte von Herstellern wie Oracle oder leichtgewichtige Datenbanken wie SQLite. Es ist modular aufgebaut und somit fast beliebig erweiterbar [6]. Durch seinen Bekanntheitsgrad und seine Verbreitung ist es auch robust und sicher. Es findet zum Beispiel in angepasster Form Verwendung auf den Internetauftritten der Regierung der U.S.A [30] und selbst die britische Regierung steuert quelloffene Module bei [7].

„Drupal is a popular free and open source publishing platform, powering high profile sites such as The White House, The New York Observer and Amnesty International“. [8]

Die letzte Komponente ist die Darstellung, welche mit Hilfe der bekannten Web-Techniken wie HTML, CSS und JavaScript umgesetzt wird. Bestimmte TEI-Elemente werden in HTML-Tags transformiert und entsprechend markiert. Textbereiche werden zu Klassen

zusammengefasst und im Code ausgezeichnet. Mit Hilfe von CSS wird dann die Darstellung, wie zum Beispiel Hervorhebungen, vorgenommen. Die dynamischen Inhalte werden durch JavaScript gesteuert und erlauben die Interaktion mit dem Benutzer, der dann Randnotizen aufklappen und zwischen verschiedenen Textvarianten umschalten kann ohne das Dokument neuzuladen.

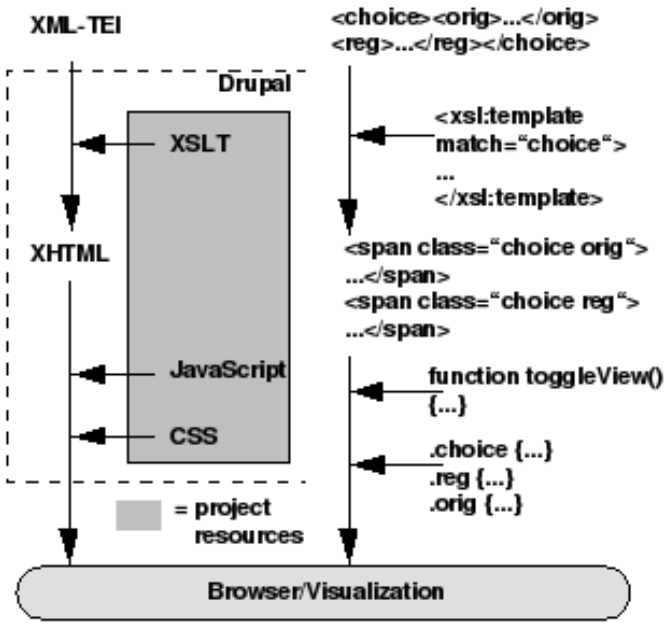

Abb. 2-1 Schema [46]

Das Zusammenwirken wird in Abbildung 2-1 verdeutlicht. Der Ablauf vollzieht sich beim Aufruf im Browser. Die in Drupal gespeicherten Dokumente liegen dort als sogenannte Nodes oder Knoten vor. Der XML-Inhalt der TEI-Dokumente wird vom XSLT-Prozessor interpretiert und anhand des XSL-Stylesheets in HTML beziehungsweise XHTML übersetzt. Das XSL-Stylesheet konvertiert etwa 20 TEI-Tags aus dem TEI-Lite-Standard. So beinhalten zum Beispiel, die im TEI-Dokument definierten Choice-Tags die verschiedenen Textversionen, zwischen denen ein Umschalten ermöglicht wird. Diese wiederum enthalten Elemente, welche den Text als regularisiert und korrigiert oder original und „falsch im Original“ (sic oder auch „wirklich so“ [31]) kennzeichnen. Daneben enthält der Text auch Anmerkungen, welche für die Darstellung am Rand besonders gekennzeichnet werden, wobei zwischen Anmerkungen des Editors und des Autors unterschieden wird.

Weiterhin werden Elemente zur Textformatierung wie Absätze, Verse aber auch Zitate, Einrückungen und die Seitenzahlen aus dem Originalbuch berücksichtigt. Der erzeugte Code enthält Verweise zu den CSS-Anweisungen für die optische Aufbereitung und zu JavaScript-Dateien, die für die dynamischen Inhalte zuständig sind. Der verwendete XSLT-Prozessor war im Projekt noch ein Drupal-Modul namens XML-Content, der mit Hilfe der PHP-XML-Erweiterung und des im Projekt erstellten XSL-Stylesheets in der Lage war die Dokumente zu transformieren. Diese wurden mit Hilfe von Drupal als HTML-Dokumente im Browser angezeigt.

2.2 Designentscheidungen

Im vorliegendem Endprodukt, musste der Benutzer also zuerst das CMS Drupal installieren, dessen Installation an sich, für weniger versierte Anwender schon nicht trivial ist. Danach war das XML-Content-Modul zu besorgen, dessen Installation in einer kleinen Textdatei, die dem Archiv beiliegt beschrieben ist. Sobald das erledigt war, musste man das Modul in der Administrations-Oberfläche von Drupal aktivieren und die notwendigen Schritte aus der Installationsanleitung nachvollziehen. Dazu gehört das Erstellen eines eigenen Dokumententyps und das Einstellen des Pfades zur Stylesheet-Datei, welche die Konvertierung der Dokumente vornehmen sollte.

Die Darstellung der eingestellten Inhalte, wird in Drupal durch den Input-Type gesteuert. Jede Node, also ein Knoten mit Inhalt, verfügt über einen Node-Type und einen Input-Type. Standardmäßig stehen Typen wie Plain-Text, HTML-Filtered und Full-HTML zur Auswahl. Dabei wird einfacher Text (plain), eine Teilmenge von HTML-Tags(filtered) oder voller HTML-Quellcode (full) verwendet. Wollte der Benutzer jetzt neue Inhalte online stellen, so musste er dazu eine neue Node erstellen, dieser einen selbst angelegten Input-Type zuweisen und den Inhalt des TEI-Dokuments in ein Textfeld einfügen. Bei Änderungen an der Darstellung, musste man CSS-Dateien per Hand editieren und auch beim Einfügen neuer Knoten, musste die entsprechende JavaScript-Datei um diese Information erweitert werden.

Die Idee war also, ein Modul zu erstellen, welches diese Schritte in einem Zug erledigen, das Einstellen neuer Inhalte weitestgehend automatisiert vornehmen und dem Benutzer dabei trotzdem freie Hand bei diversen Einstellungen lassen sollte, ohne dass er Dateien manuell editieren muss.

Die Entscheidung fiel auf „Drupal“ als Grundsystem, aufgrund der Tatsache, dass es schon im XSL-Projekt Verwendung fand und sich in der Handhabung bewährt hatte. Da nach Beendigung des Projekts die Veröffentlichung von Drupal 7 absehbar war, lag die Entscheidung nahe, das zu entwickelnde Modul, gleich an diese Version anzupassen.

Beim verwendeten Drupal-Modul XML-Content fiel auf, dass es seit längerer Zeit nicht mehr gepflegt wurde, und dass auch der letzte Versionssprung von Drupal nur skriptgesteuert, also ohne Menschliche Eingriffe, stattfand. Da der Code unter GPL, also einer Open-Source-Lizenz stand, sprach nichts dagegen den alten Quelltext zu verwenden und anzupassen [12]. Da sich die Verwendung des Moduls auf nur ein Stylesheet zur TEI-Darstellung beschränkte und diverse Schnittstellenänderungen, beim Wechsel auf Version 7 von Drupal, sowieso größere Änderungen am Code zur Folge gehabt hätten, wurde beschlossen das Modul zu integrieren und entsprechend anzupassen. Das hat zudem den Vorteil, dass eine Abhängigkeit vom Modul wegfällt und man auf dessen Wartung nicht mehr angewiesen ist.

Die Rückmeldung nach Fertigstellung des Projekts ergab auch, dass es von Vorteil wäre, wenn man die Schriftarten für die dargestellten Texte einstellen könnte. Dazu bietet sich die Fähigkeit vieler moderner Browser an, eigene Schriftarten per CSS einzubinden. Natürlich müsste auch hier, die Konfiguration entsprechend unkompliziert von statten gehen, ohne dass

der Benutzter von Hand eingreifen, Font-Dateien ins System kopieren und CSS-Dateien editieren muss. Zum Zeitpunkt der Erstellung dieser Diplomarbeit gibt es im wesentlichen 3 Formate, die teilweise von weitverbreiteten Webbrowsern unterstützt werden. Dabei handelt es sich um TTF (TrueType-Font), OTF (OpenType-Font) und WOFF (Web Open Font Format). Wobei letzteres, ein Container-Format für die davor erwähnten Schriftformate darstellt, welches Schriftdateien kapselt, mit Metainformationen versieht und komprimiert [9].

Die Verwendung des Book-Moduls, hat sich bei der Benutzung, der aus dem Projekt hervorgegangenen Endprodukts bewährt. Das Modul bietet eine gute Möglichkeit TEI-Dokumente, die als Kapitel eines Buchs vorliegen, zu gruppieren und zu einer logischen Einheit zusammen zu fassen. Da dieses Plugin weiterhin kontinuierlich gepflegt wird und sich auch im „Kern“-Paket von Drupal wiederfindet, also mit dem CMS ausgeliefert wird, musste nur berücksichtigt werden, dass beim Erstellen neuer Einträge, die Interaktion mit dem Book-Plugin ermöglicht wird. Sollte zum Beispiel das Plugin aktiviert sein, so sollten auch entsprechende Optionen zur Verfügung gestellt werden, die eine direkte Deklaration des Dokuments als „Book Page“, also zu einem als Buch definiertem Node-Verbund zugehörig, ermöglichen.

Beim Testen und Probieren in der Anfangsphase der Entwicklung brauchte das Modul einen Namen. Verwendet wurde zunächst das Wortspiel „TEIChi“, welches das Kürzel TEI enthielt und gleichzeitig nach chinesischer Philosophie klang, wobei der echte Name Taiji so etwas wie „das sehr große Äußerste“ bedeutet, was zu einem Komplettpaket ganz gut passt. Später wurde dieser Name, auf Anregung von Prof. Wegner, um die Bedeutung „Text Encoding Initiative Computer Human Interface“ erweitert.

Da die Ziele und der Umfang jetzt klar definiert waren, konnte mit der Implementierung begonnen werden.

3 Verwendung

3.1 Voraussetzungen

Im Folgenden wird die Verwendung des Moduls nach erfolgreicher Aktivierung erklärt. Die Implementierung der hier vorgestellten Funktionalität, wird in darauffolgenden Kapiteln beschrieben. Was für die Verwendung benötigt wird, sind Dokumente im TEI-XML-Format, bei mehreren Büchern das Book-Plugin und je nach Bedarf Schriftdateien, die man anstatt der Standard-Font-Klassen verwenden möchte.

3.2 Aktivierung

Wurden die Informationen aus der, im Modul enthaltenen, Info-Datei richtig eingelesen, bietet Drupal in der Modulauflistung an das Modul zu aktivieren. Hierzu muss das Modul zunächst in das Installationsverzeichnis von Drupal extrahiert werden. Der vorgesehene Pfad für Module ist *<Drupal>/sites/all/modules*, wo es in einen Ordner mit dem Modulnamen als Bezeichnung entpackt werden muss. Alternativ ist auch eine Installation über ein tar-Archiv möglich. Dazu wird Drupal angewiesen das Modul über eine URL runterzuladen und an den dafür vorgesehenen Ort zu platzieren. Sind diese Aktionen durchgeführt und alle Prüfungen erfolgreich und ohne Fehlermeldungen durchgelaufen, ist das Modul betriebsbereit. In der Menüleiste des Administrationsbereichs, sollte jetzt ein zusätzlicher Eintrag „TEIChi Preferences“ zu sehen sein (Abb. 3-1). Über diesen gelangt man zur Dokumentübersicht des TEIChi-Moduls.

Abb. 3-1 Menü

3.3 Hochladen von Dokumenten

Zum Hochladen von Inhalten, ist das Anmelden im Administrationsbereich von Drupal erforderlich. Dort wird, entweder über die Menüleiste, die in der Standardeinstellung oben angezeigt wird oder durch einen Klick auf das Zahlrad-Symbol in der Modulauflistung, hinter dem Modulnamen, zum TEIChi-Einstellungsmenü navigiert. So gelangt man zur Hauptseite der Modulkonfiguration und hat im oberen Bereich die drei Reiter „Display“, „Documents“ und „Fonts“ zur Auswahl. Nach einem Klick auf „Documents“, erscheint das Formular, in dem die Datei bestimmt wird, die hochgeladen werden soll.

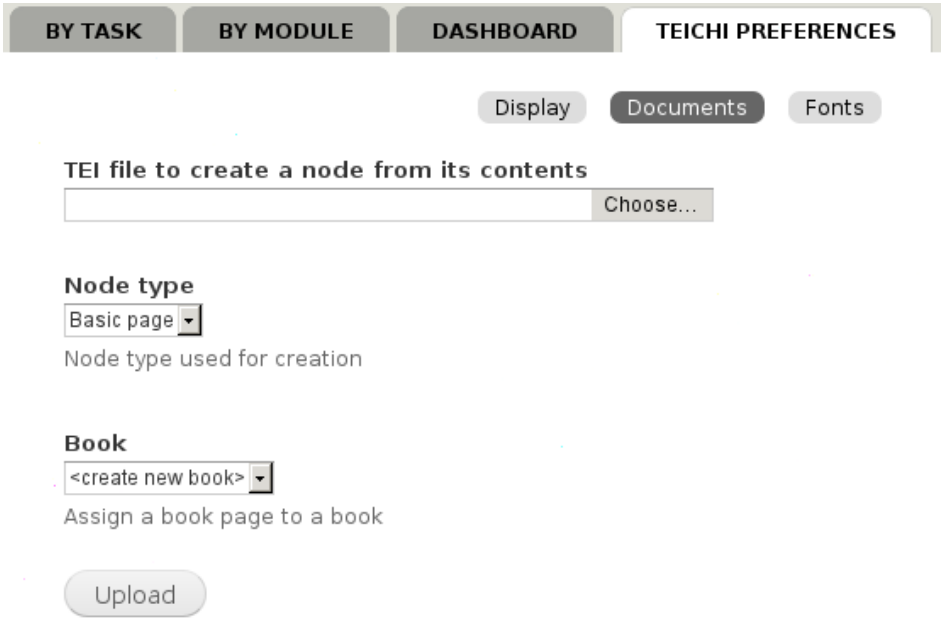

Abb. 3-2 TEI-Upload

Dort sieht man ein Eingabefeld mit einem Knopf daneben der mit „Auswählen“ oder ähnlichem beschriftet ist (die Beschriftung ist abhängig vom Browser und der eingestellten Systemsprache). Ein Druck auf diesen Knopf, bringt einen Dateiauswahldialog zum Vorschein, mit dem man dann zur TEI-Datei auf dem lokalen Dateisystem navigiert, dort die gewünschte Datei auswählt und bestätigt. Zu beachten ist, dass nur Dateien mit der Endung xml, tei oder txt erlaubt sind. Versuche eine Datei mit einer anderen Endung hochzuladen, werden mit einer entsprechenden Fehlermeldung quittiert.

Als nächstes sucht man den Knotentypen aus, den das Dokument dann haben soll. Dazu wählt man einen der in der Dropdown-Liste zur Verfügung stehenden Typen aus. Das sollten in der Regel mindestens „Article“ und „Basic Page“ sein. Ist zusätzlich noch das Book-Plugin aktiviert steht auch „Book-Page“ zur Auswahl .

Um nun zum Beispiel ein Dokument vom Typ „Article“ zu erstellen, wählt man diesen Typ aus und drückt den unteren Knopf mit der Aufschrift „Upload“. War der Vorgang erfolgreich, wird Drupal eine Nachricht ausgeben, die den Titel, eventuell erkannte Seiten und die Node-Nummer des neu erstellten Knotens anzeigen.

Damit Seitenzahlen im Text des Dokuments erkannt werden können, müssen diese nach einem speziellen Schema kodiert worden sein, auf das man sich beim vorangegangenen Projekt geeinigt hat. Seiten die im TEI-Dokument mit einem <pb>-Tag codiert sind, sollten ein zusätzliches ID-Attribut enthalten. Dieses muss die Form „pX“ haben, wobei X für eine beliebige, im Dokument eindeutige, positive, ganze Zahl steht. Das heißt würde man zum Beispiel Seite 3 markieren, wäre sie folgendermaßen im TEI-Dokument notiert:

```
<pb xml:id=“p3“>[…]  Seiteninhalt […] </pb>
```

Der neu erstellte Inhalt wird dann im Hauptmenü des Moduls aufgelistet, inklusive Links zum Anzeigen und Löschen, sowie der enthaltenen Seiteninformationen. Auch im Content-Bereich von Drupal, wo alle Dokumente verwaltet werden, wird das Dokument jetzt gelistet. Dort stehen die üblichen Funktionen wie editieren und löschen zur Verfügung.

Hat man vor mehrere Bücher einzustellen, ist es von Vorteil einen Knoten schon vorher als „Book-Page“ zu deklarieren. Dazu muss man zunächst sicherstellen, dass das Book-Plugin aktiviert ist. Das lässt sich leicht feststellen, indem man im Administrationsbereich von Drupal, im Menüpunkt „Modules“ die Modulauflistung überprüft. Dort sollte ein Haken in der ersten Spalte „Enabled“ erscheinen. Sollte das nicht der Fall sein, kann es wie das TEIChi Modul über das Kontrollkästchen und einen Klick auf „Save configuration“ nachgeholt werden. Nun begibt man sich in den Content-Bereich im Administrationsmenü von Drupal und klickt auf „Add new content“. Dort wird jetzt unter anderem auch „Book page“ als Typ für neue Inhalte aufgeführt. Diesen wählt man aus und gelangt dann zu einem Formular, wo Titel des Buchs und der Inhalt des Knotens eingegeben werden können. Zusätzlich kann man sich, je nach Inhalt für ein Textformat entscheiden. Im unteren Bereich wählt man dann den Punkt „Book outline“. Daraufhin erscheint daneben eine Dropdown-Liste, in der man ein Buch auswählen beziehungsweise erstellen kann. Um ein neues Buch zu erstellen wählt man dort „<create new book>“ und bestätigt das Ganze mit einem Druck auf „Save“. Wenn Drupal die Erstellung mit einer positiven Nachricht quittiert, kann man jetzt wieder in das TEIChi-Konfigurationsmenü wechseln. Dort navigiert man wieder per Klick auf „Documents“ zum Dokument-Upload und wählt wie gehabt die Datei aus, die man hochladen möchte. Als nächstes wählt man bei „Node type“ in der Auswahlbox, den Typ „Book page“. Die Dropdown-Liste darunter bietet jetzt die Möglichkeit diese Buchseite direkt, einem als Buch definierten Knoten zuzuweisen. Wenn man dieses Drop-Down-Menü jetzt aufklappt, sollte der zuvor erstellte Buchknoten aufgelistet sein. Diesen Wählt man nun aus und betätigt den „upload“-Knopf. Somit wird ein Dokument hochgeladen und gleichzeitig einem Buch zugewiesen.

Eine weitere Möglichkeit gibt es, wenn man ein TEI-Dokument vorliegen hat, dass als Buchknoten fungieren soll. So kann man einfach wieder in das TEIChi-Menü wechseln und dieses, wie zuvor unter „Documents“ in das Eingabefeld eintragen. Als Node-Type wählt man dann wieder „Book page“, in der Auswahlbox für den Buchknoten wählt man den Eintrag „<create new book>“, statt eines vorhandenen Buches. Nach dem Betätigen des „upload“-Knopfes wird dann ein Knoten mit den enthaltenen Informationen erstellt und zusätzlich ein gleichnamiger Buchknoten erzeugt, dem dann weitere Kapitel beziehungsweise Seiten zugewiesen werden können.

3.4 Hochladen eigener Schriftarten

Hat man lokal Schriftarten vorliegen, die in den TEI-Dokumenten zur Darstellung verwendet werden sollen, so bietet TEIChi die Möglichkeit diese hochzuladen, damit sie beim Seitenabruf in die Dokumente eingebunden werden können. Das kann man im Menü des Moduls erledigen, indem man mit dem Reiter „Fonts“ zum entsprechenden Formular navigiert.

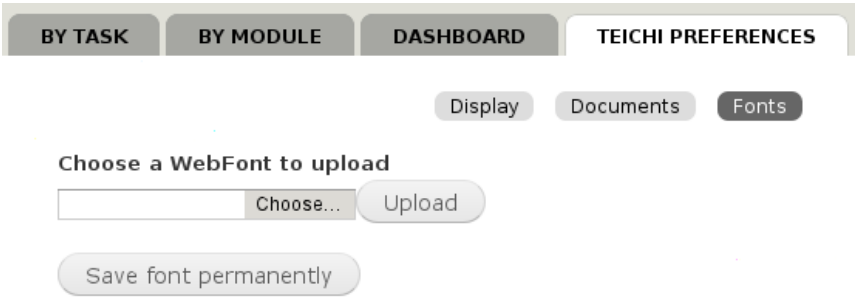

Abb. 3-3 Hochladen von Schriften

Dort gibt es ein Eingabefeld und zwei Knöpfe. Zunächst wählt man, wie schon bei den Dokumenten, die Datei auf dem lokalen Speicher, die man hochladen möchte. Im Unterschied zu den Dokumenten wird nach dem Bestätigen der Datei, der „Upload“-Knopf neben dem Eingabefeld betätigt. Nachdem man diesen gedrückt hat, sollte ein Ladebalken erscheinen, der den Fortschritt des „Uploads“ anzeigt. Das die Font-Dateien in der Regel nicht größer als wenige hundert Kilobyte groß sind, dürfte der Ladevorgang nicht lange dauern. War das Hochladen erfolgreich, wird die Datei mit einem Icon und dem Dateinamen, anstatt des Eingabefelds, auf der Seite angezeigt. Zugelassen sind auch hier nur Dateien mit einer bestimmten Dateiendung. Hier sind es otf, ttf und woff die erlaubt sind, ansonsten schlägt das Hochladen fehl. Laut „Webfonts.info“ [25] werden die Typen OTF und TTF von allen modernen Browsern unterstützt, deswegen sollte diese Auswahl den größten Teil der darstellbaren Formate abdecken. Im Detail kann man die unterstützen Formate der folgenden Tabelle entnehmen:

Tab. 3-1 Unterstützung von Schriftarten in gängigen Browsern [40]

Name:	**Version:**	**Format:**
Opera	Opera 10	TrueType/OpenType TT (.ttf) OpenType PS (.otf) SVG (.svg)
Mozilla/Firefox	Firefox 3.5	TrueType/OpenType TT (.ttf) OpenType PS (.otf) WOFF (seit Firefox 3.6)
Google Chrome	Chrome 4.0 stable release	TrueType/OpenType TT (.ttf) OpenType PS (.otf) WOFF seit Version 6
Webkit/Safari	Version 3.1	TrueType/OpenType TT (.ttf) OpenType PS (.otf) SafariMobile (iPhone/iPad): nur SVG Fonts
Internet Explorer	IE 4	Embedded OpenType (EOT) erzeugt mit Microsofts WEFT- Tool WOFF seit IE9
Netscape	Version 4, nicht fortgeführt nach Netscape Navigator 6	Portable Font Resource (.pfr)

Auch nach dem Hochladen kann man die Auswahl noch korrigieren. Falls man zum Beispiel aus Versehen die falsche Datei hochgeladen hat, hat man die Möglichkeit sie wieder zu entfernen, in dem man den neben dem Dateinamen eingeblendeten Knopf „remove“ betätigt. Auch wenn man an diesem Punkt abbricht, ohne die Datei zu löschen oder zu bestätigen, löscht Drupal die Datei nach einer gewissen Zeit automatisch. Standardmäßig läuft alle 6 Stunden ein Hintergrundprozess durch und löscht alle Dateien, die nicht als permanent markiert sind.

3.5 Darstellung anpassen

Das Ändern der Darstellung kann vielerlei Gründe haben. Da Drupal sehr modular aufgebaut ist, ist auch das Aussehen stark veränderbar. So gibt es zum Beispiel die Möglichkeit andere, sogenannte Themes zu definieren, diese können dann eine ganz andere Farbgebung und Anordnung der Elemente zur Folge haben, weswegen es wünschenswert sein kann, Farben und andere Eigenschaften der XSL-Visualisierungen anzupassen. Weitere Gründe sind, dass sich zum Beispiel Menschen mit Sehschwäche, bei der Arbeit mit den Dokumenten höher Kontraste wünschen oder einfach eine kleine Anpassung der Darstellung gewünscht wird, ohne zu tief ins Drupal-System eingreifen zu müssen. Das TEIChi-Modul bietet im Untermenü „Display“ der Konfigurationsmenüs die Möglichkeit Farben, Größen, Schriftarten und Texte für bestimmte Elemente zu definieren. Die einzelnen Einstellungen sind gruppiert, damit sie nicht zu viel Platz einnehmen und man die Übersicht nicht verliert. Ganz oben steht „Colors“ zur Auswahl. Nach dem Aufklappen dieses Menüs kommen Eingabefelder und grafische Farbauswahl-Elemente, sogenannte „Colorpicker“ zum Vorschein.

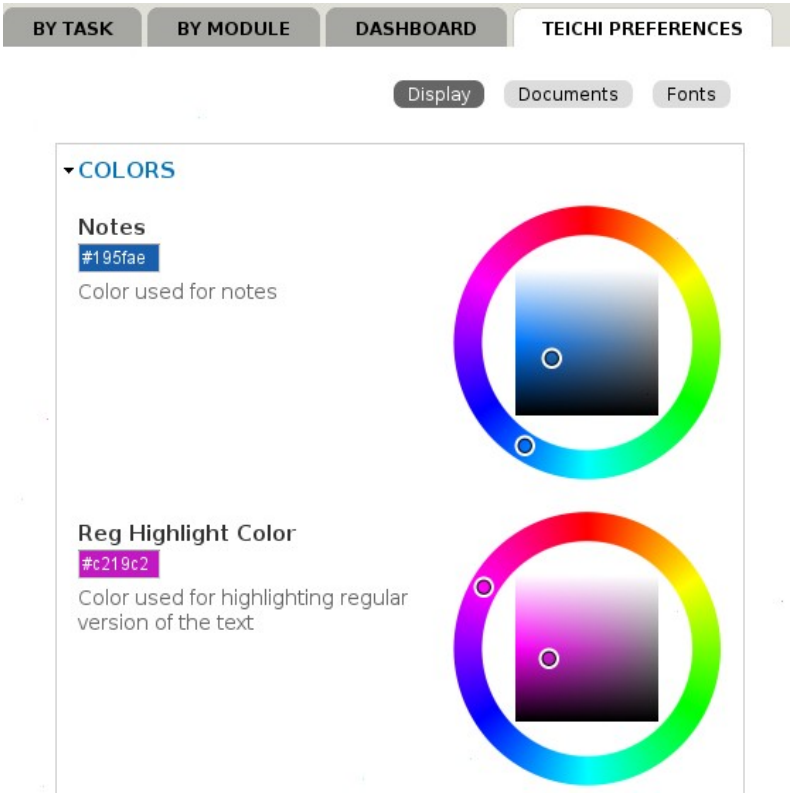

Abb. 3-4 Farbauswahl

In die Felder kann man nun einen, wie in HTML üblich, einen hexadezimalen Farbwert eintragen, wobei sich der Wert aus den drei Grundfarben Rot, Grün und Blau zusammensetzt. Dabei kann jede Farbe einen Wert zwischen 0 und 255 annehmen, also FF in der hexadezimalen Schreibweise. Eingeleitet wird der Wert mit einem Raute-Symbol #. So ist der Wert für Weiß zum Beispiel #FFFFFF, für Schwarz #000000 und für Rot #FF0000. Möglich sind auch die aus dem CSS-Standard [27] bekannten Farbnamen wie „black“, „magenta“ und so weiter. Um dem Benutzer die Eingabe hexadezimaler Farbwerte zu ersparen, gibt es daneben die oben erwähnten Farbauswahl-Elemente, mit denen man sich grafisch und per Maus eine Farbe zusammenmischen kann. Dabei bestimmt man über den äußeren Ring die Farbe und mit dem Quadrat in der Mitte ihre Intensität. Die ausgewählte Farbe, beziehungsweise ihr hexadezimaler Farbwert, wird dann in das dafür vorgesehene Eingabefeld eingetragen. Farben kann man festlegen für die Notizen am Rand, für die verschiedenen Textfassungen und für den Hintergrund der Menüleiste des Moduls. Alles weitere sollte sich über diverse Themen, beziehungsweise in deren Anzeigeeinstellungen regeln lassen.

Als nächster Menüpunkt ist „Sizes“ aufgelistet. Dort kann man verschiedene Größen und deren Einheiten festlegen. Einheiten können je nach Anlass und Sinnhaftigkeit, Prozentwerte (%), Punktangaben (pt) oder auch Pixelwerte (px) darstellen.

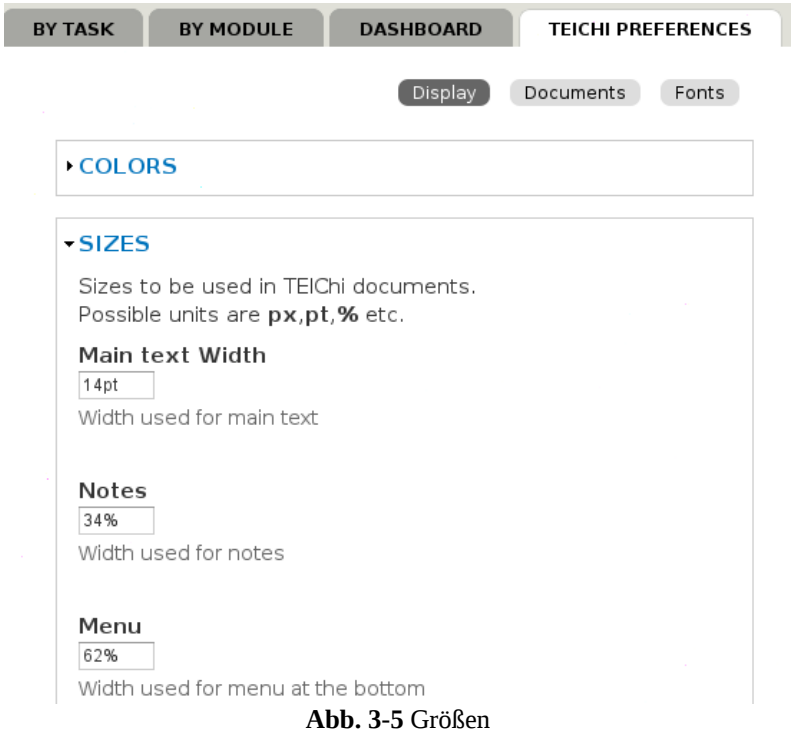

Abb. 3-5 Größen

Weitere Einheiten sind relative Einheiten wie „em“ oder auch Größen wie „cm“ und „mm“. Eine Auflistung aller Einheiten findet man beim World Wide Web Consortium (w3c) im CSS-Standard [41]. Es wäre auch möglich gewesen, eine Auswahl sinnvoller Einheiten, als sogenannte „Radiobuttons“ hinter das Feld mit den Werten zu platzieren. Doch dabei besteht die Gefahr, dass man Einheiten weglässt, die jemand persönlich als wichtig erachtet oder, dass ein Benutzter eine spezielle Kombination aus Einheiten benötigt, die dann womöglich nicht

einstellbar ist. Es ist auch vorstellbar, dass neue Einheiten hinzu kommen oder in diversen Browsern anders oder gar nicht interpretiert werden können.

Auch für die an der Seite eingeblendeten Anmerkungen ist eine Breite wählbar, dort empfehlen sich prozentuale Werte aber auch Pixelwerte und andere Einheiten sind möglich. Weiterhin ist auch für die unten eingeblendete Menüleiste ist eine Breite wählbar, dabei sollte man beachten, dass noch genug Platz, für die enthaltenen Elemente übrig bleibt, da sonst Darstellungsprobleme durch automatische Zeilenumbrüche oder verdeckte Navigationselemente auftreten können.

Die nächste Einstellungsmöglichkeit ist das festlegen von Schriftarten. Nach dem Aufklappen dieses Menüpunkts, kommen Dropdown-Listen zum Vorschein.

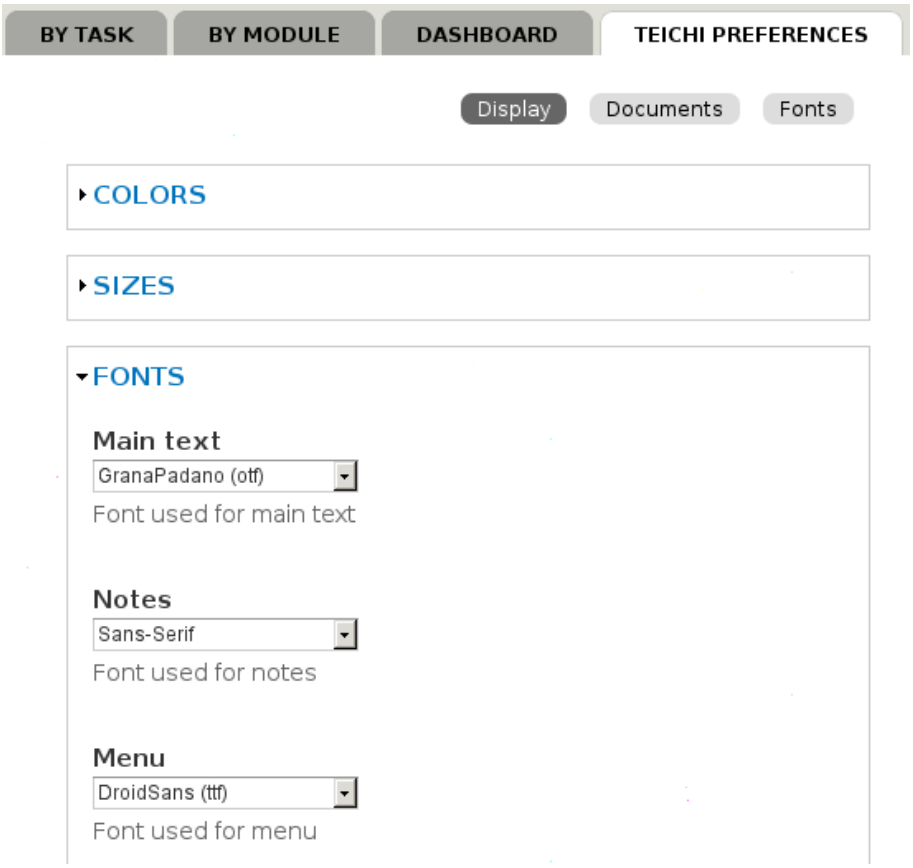

Abb. 3-6 Schriftartenauswahl

Dort stehen zunächst Font-Klassen zur Auswahl, sofern man schon Schriftarten hochgeladen hat, sollten auch diese dort aufgelistet sein. Die zur Auswahl stehenden Klassen sind „Sans-Serif“, „Serif“ und „Monospaced“, wobei „Sans-Serif“ für eine Serifenlose Schriftart steht, für die dann im Browser eine systemeigene Standardschriftart wie Arial, Verdana oder Helvetica herangezogen wird. Bei „Serif “ wählt der Browser beim Anzeigen eine Schrift wie „Times New Roman“ oder andere bekannte serifenbehaftete Schriftarten. „Monospace“ wird auf vielen Systemen dargestellt, als nichtproportionale Schriftart, also mit gleichen Buchstabenbreiten, wie zum Beispiel „Courier“ und ähnliche. Sind weitere Schriftarten vorhanden, werden sie mit dem Namen, der aus dem Dateinamen extrahiert wurde und dem Dateiformat, der durch die Dateiendung angeben ist angezeigt. Wird explizit eine dieser Schriftarten zur Darstellung ausgewählt, so wird genau diese Schriftart, für diesen Texttyp verwendet. Was bei diesem Ansatz leider nicht möglich ist, ist das festlegen von CSS-

typischen Reihenfolgen, die abgearbeitet werden, falls eine Schriftart nicht vorhanden ist. Zwar ist das Vorhanden sein der Schriftart Voraussetzung, für die Auflistung in der Auswahl. Es kann aber nicht gewährleistet werden, dass der Browser der Besuchers, auf den jeweiligen Seiten, über die Fähigkeiten verfügt diese Art von Schriftart darzustellen. Das kann zum einen, die fehlende Fähigkeiten des Browsers sein diese CSS-Eigenschaft zu interpretieren oder auch die Einschränkung auf bestimmte Dateitypen in der Font-Darstellung. Möglich ist auch eine Änderung der CSS-Eigenschaft an sich, da dieser Bereich sich immer noch im Entwurfsstadium befindet, was dann eine Änderung am Modul-Code erforderlich machen würde. Letztendlich muss der Administrator der Seite für sich entscheiden, welche Zielgruppe er ansprechen will und welcher Kompromiss für ihn der beste ist.

Die letzte Einstellungsoption betrifft Texte, die bei bestimmten Ereignissen angezeigt werden.

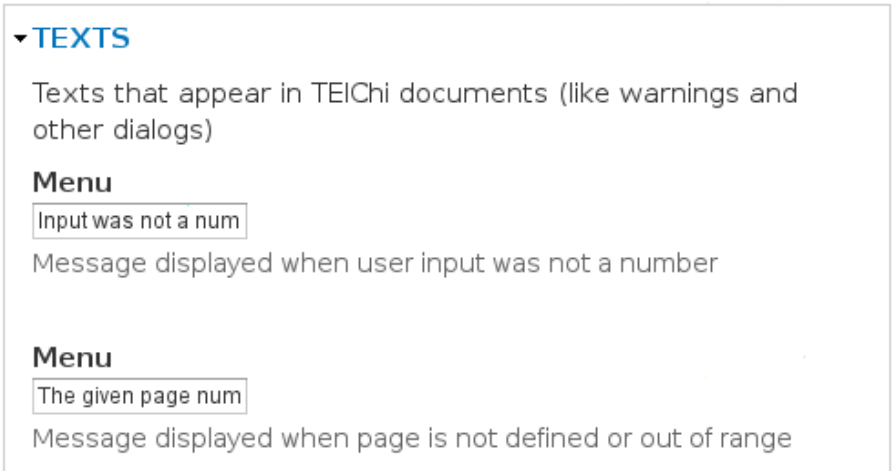

Abb. 3-7 Texteinstellung

Sie werden vom JavaScript-Teil als Meldungen aufgerufen und können nicht ohne weiteres von Drupals Übersetzungsfunktionen erfasst werden, außerdem bietet dieser Ansatz auch den Vorteil, dass man dargestellte Systemmeldungen des Moduls, unabhängig von der gewählten Sprache der Drupal-Installation definieren kann. So kann zum Beispiel ein auf englisch installiertes Drupal, Hinweise und Fehlermeldungen auf französisch ausgeben. Auch der Text in der Menüleiste kann frei gewählt werden, so dass er schnell und leicht anpassbar ist, ohne sich auf Übersetzungserweiterungen verlassen zu müssen. Die Einstellung gilt aber modulweit nicht etwa pro Buch und ist somit als Kompromiss zu sehen, zu einer Variante mit fest codierten Zeichenketten.
Sind alle gewünschten Einstellungen vorgenommen, muss noch der „Save configuration“ Knopf betätigt werden. Dieser veranlasst das Modul, die betreffenden Stylesheet-Dateien und Texte neu zu schreiben. Dabei ist zu beachten, dass es durch den Caching-Mechanismus des Browsers durchaus passieren kann, dass die vorgenommenen Änderungen nicht auf Anhieb sichtbar sind. Ist das der Fall, sollte man ein Neuladen der Seite im Browser anstoßen. Das geschieht in der Regel über ein Symbol in der Browseroberfläche oder eine Taste beziehungsweise Tastenkombination wie F5 oder Steuerung-Taste und r.

3.6 Darstellung der Dokumente

Über das TEIChi-Konfigurationsmenü oder über Drupals Content-Bereich gelangt man zur Darstellung der eingestellt Dokumente, wo sie durch Transformation visualisiert werden. Im Vergleich zum Ausgangsprojekt, hat sich optisch nicht viel verändert. Im Hintergrund jedoch, stößt jetzt das TEIChi-Modul die Transformation der Visualisierungen an und bindet die nötigen Komponenten wie, CSS-Dateien und Skripte ein. Entsprechende TEI-Tags, wie Verse und Absätze, werden in der HTML-Anzeige hervorgehoben formatiert.

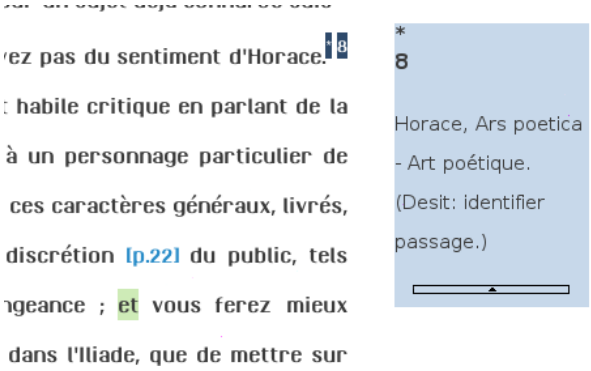

Abb. 3-8 Anmerkungen

Anmerkungen, vom Autor und Editor, werden an den dafür vorgesehenen Textstellen im Fließtext, mit einem Marker versehen und auf gleicher Höhe mit der eingestellten Breite zugeklappt an gezeigt. Dabei werden Notizen vom Editor mit Zahlen markiert und Anmerkungen des Autors mit einem „*“. Ein Klick auf diese Marker, öffnet die betreffende Notiz an der Seite (siehe Abb. 3-8). Auch das Klicken auf die Notiz selbst ist jetzt möglich. Ein weiterer Klick auf die Marker im Text oder den oberen Bereich der Notiz schließt geöffnete Randbemerkungen wieder.

Die Textvarianten werden in der, im Konfigurationsmenü festgelegten, Farbe dargestellt. Dabei sind jeweils die Textvarianten „reg“, „corr“ und „orig“, „sic“ zusammengefasst und nur in dieser Konstellation umschaltbar. Dabei zeichnen die reg- und corr-TEI-Tags, den beinhalteten Text als regularisiert beziehungsweise korrigiert aus und im Gegensatz dazu wird Text in orig- und sic-TEI-Tags als Originaltext oder im original falscher Text ausgezeichnet. Welche Version angezeigt wird, lässt sich in der, im unteren Bereich eingeblendeten Leiste, bestimmen.

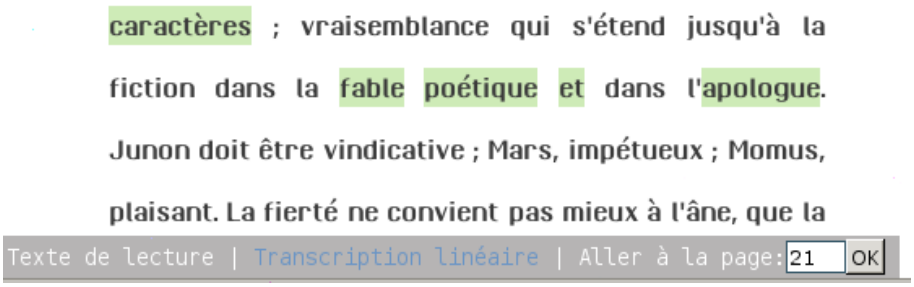

Abb. 3-9 Textumschaltung

Die Schalter dafür tragen die im Konfigurationsmenü gewählte Bezeichnung. Ganz links ist ein Knopf zum Einblenden der Seitenindexfunktion. Dort sind alle im aktuellen Kapitel

enthaltenen Seiten aufgeführt.

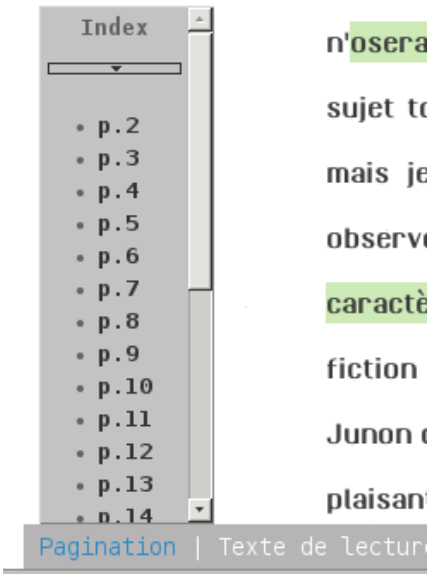

Abb. 3-10 Seitenindex

Das Anklicken einer dieser Seiten, führt zum Wechseln des sichtbaren Bildbereichs zu der Seite. Am rechten Rand der Menüleiste befindet sich die Seitensuchfunktion, die aus einem Eingabefeld und einem Knopf zum bestätigen besteht.

Abb. 3-11 Seitensuche

Diese funktioniert jetzt Buchweise und quittiert im aktuellen Buch nicht vorhandene Seiten oder nicht numerische Eingaben mit der jeweiligen Fehlermeldung, dessen Text in der Konfiguration des Moduls bestimmt werden kann.

4 Implementierung

4.1 Modulaufbau

4.1.1 .info-Dateien

Ein Drupal-Modul besteht im Wesentlichen aus einem Verzeichnis, welches den Namen des Moduls in Kleinbuchstaben trägt und bestimmte Dateien enthält die Details zum Modulaufbau enthalten. Die Konvention sieht vor, dass jedes Modulverzeichnis eine Info-Datei enthält. Diese Datei trägt den Namen des Moduls und die Endung .info. Im vorliegenden Fall also teichi.info. Der Aufbau ist, wie in der Dokumentation [11] beschrieben, relativ simpel.

Am Anfang jeder Info-Datei ist es typisch eine CVS[1] Id-Variable zu platzieren. Danach folgen die zwingenden Angaben wie Name, Beschreibung und der verwendete Kern, also die Drupal-Version zu der das Modul kompatibel ist, sowie weitere zum Modul zugehörige Dateien in Form eines file[]-Eintrags. Optionale Angaben sind zum Beispiel „package“, welches das Modul in der Administrationsoberfläche entsprechend einordnet und ein Konfigurationspfad, der die URL zum Einstellungsmenü darstellt.

```
; $Id$
name = "TEIChi"
description = "Lets you upload TEI/XML files as nodes into
Drupal"
core = 7.x

package = "Content Filters"
files[] = teichi.install
files[] = teichi.module
files[] = conf.module
files[] = teichi.cssfile.inc

configure = admin/settings/teichi
```

Abhängigkeiten zwischen Modulen sind ebenfalls möglich. Diese können in dieser Datei definiert werden und zwar mit dem Eintrag `dependencies[]` der, wie `files[]`, auch mehrfach vorkommen kann. Würde man zum Beispiel einen Eintrag `dependencies[] =book` hinzufügen, ließe sich das Modul nur beim Vorhandensein eines bereits aktivierten Book-Moduls einschalten.

1 Concurrent Versions System (Versionskontrollsystem)

4.1.2 .module-Dateien

Die Dateien mit der Endung .module enthalten den eigentlichen Code. Der Name setzt sich zusammen aus dem Modulnamen und der .module-Endung. Im vorliegenden Fall also teichi.module. Der darin enthaltene PHP-Code wird von Drupal geladen wenn das Modul aktiviert wurde. Aus dieser Datei heraus können auch andere Quelldateien inkludiert werden. Das hilft die Übersicht zu behalten und den Code zu organisieren.

4.1.3 Hook-Funktionen

Die Kommunikation zwischen Drupal und seinen Modulen geschieht über sogenannte Hook-Funktionen. Wie in [1, S. 333] beschrieben, dient das der Performance. Es werden also nur Hook-Funktionen aufgerufen, wenn das jeweilige Modul aktiviert ist. Drupal versucht auch soviel wie möglich zu „cachen“, also zwischen zu speichern, was bei der Entwicklung manchmal etwas hinderlich sein kann. Die Namenskonvention folgt dabei dem Schema hook_<Funktion>. Wobei „hook“ durch den eindeutigen Namen des Moduls zu ersetzen ist und <Funktion> ein vorgegebener, meistens aussagekräftiger Funktionsname.
Das heißt also, möchte man, dass das eigene Modul auf das Löschen von Nodes reagiert und von Drupal darüber in Kenntnis gesetzt wird, so implementiert man eine Funktion im eigenen Modul die im Falle des „teichi“ benannten Moduls `teichi_node_delete($node)` heissen muss. Welche Parameter übergeben werden, muss man der Dokumentation entnehmen.

4.1.4 .install-Dateien

Die Informationen zu dem was bei der Aktivierung beziehungsweise noch davor geschehen soll, steckt in den .install-Dateien. Diese enthalten eigentlich nur „Hooks“ die von Drupal bei der Installation aufgerufen werden. Somit wird sichergestellt, dass zum Beispiel Aufgaben, die noch bevor das Modul aktiviert wird erledigt werden müssen, auch abgearbeitet werden. Prädestiniert sind dafür zum Beispiel Aktionen wie das Anlegen von Datenbanktabellen und Ähnliches. Diese „Hooks“ können eigentlich auch in der .module-Datei abgelegt werden, der Übersicht halber ist es aber von Vorteil, sie in eine eigene Datei auszulagern.

4.1.5 Andere Dateien

Das Modulverzeichnis enthält außerdem auch andere Dateien, wie zum Beispiel JavaScript-Dateien, Bilder und Ähnliches, die dann über den Modulpfad, der sich von Installation zu Installation ändern und von Drupal zur Laufzeit abgefragt wird, leichter eingebunden werden können. Außerdem kann das Modul dort selber Dateien anlegen und ändern, solange die erforderlichen Rechte vorhanden sind. Die Struktur des Moduls entspricht der folgenden

Darstellung:

```
.
|-- conf.module
|-- css
|   `-- teichi.css
|-- img
|   |-- closenote.png
|   `-- index.png
|-- js
|   |-- drp_main.js
|   `-- drp_teichi.js
|-- teichi.cssfile.inc
|-- teichi.info
|-- teichi.install
|-- teichi.module
|-- teichi.xslfile.inc
`-- xsl
    `-- drp_style.xsl
```

Einige der dargestellten Dateien, wie .css oder .js Dateien, werden erst beim Aktivieren des Moduls erzeugt, da sie variable Inhalte speichern, wie zum Beispiel den Pfad zum Modul innerhalb einer Installation und quasi „on the fly“ erzeugt werden.

4.2 Hintergrundablauf und Schreibrechte

Beim Aktivieren des Moduls werden zuerst die _install-Hooks und anschließend die _enable-Hooks ausgeführt. Die _install-Hooks erstellen die Tabelle für die Zusatzinformationen des Moduls. Beim _enable-Hook, in dem Fall `teichi_enable`, wird zunächst sichergestellt, dass in dem vorhandenen PHP, die XML-Unterstützung aktiviert ist. Das geschieht mit `_teichi_check_xml_support()`, welches aus XML-Content übernommen wurde. Es liefert einen boolschen Wert zurück. Bei einem „false“ wird eine Fehlermeldung ausgegeben. Danach wird `_teichi_xsl_file` ausgeführt. Diese Funktion dient zum Schreiben des XSL-Stylesheets. Sie ist wegen ihrer Größe und der Übersicht halber, in eine eigene Datei ausgelagert worden. Das Schreiben des Stylesheets per PHP anstatt es mit auszuliefern, war notwendig, da im Stylesheet Bilder verlinkt sind, deren Pfad erst zum Zeitpunkt der Installation bestimmt werden kann. Für diese Aktion sind Schreibrechte im Modul-Verzeichnis erforderlich und zwar für den Prozess, der das PHP-Skript ausführt, ansonsten kommt es zu einer Fehlermeldung und als weitere Folge kommt hinzu, dass TEI-Dokumente nicht dargestellt werden können, da kein Stylesheet zum Transformieren vorhanden ist. Da Schreibrechte auch für andere Funktionen des Moduls benötigt werden, ist das Schreibrecht Voraussetzung für die Benutzung dieses Moduls. Die Funktion speichert zunächst den Modulpfad mit Hilfe von:

```
$teichi_path = drupal_get_path('module','teichi');
```

Und setzt daraus dann den Pfad für die Bilddateien zusammen:

```
$img_path = $teichi_path."/img";
```

Danach wird das eigentliche Dokument, ähnlich wie das CSS-Stylesheet, per „here Document“, als String in einer Variablen abgelegt. Bei den Bildern werden dann die Pfade mit den entsprechenden Werten ergänzt. Zum Schluss wird der Variableninhalt mit der folgenden Zeile geschrieben:

```
file_put_contents($xslPath,$xsl);
```

Hiernach wird geprüft, ob die Installation XSLT beherrscht, was mit der Funktion `_teichi_check_xsl_available` passiert, welche auch einen boolschen Wert zurückliefert, der bei „false“ zu einer Fehlermeldung führt. Da das Modul seinen eigenen Input-Type benötigt, um die TEI-Dokumente darzustellen, muss dieser beim ersten Aktivieren erzeugt werden oder bei erneuter Aktivierung sein Vorhandensein sichergestellt werden. Die Prüfung wird in der Funktion `_teichi_check_input_format` ausgeführt. Diese fragt mit `filter_formats` eine Liste aller vorhandenen Formate ab und vergleicht sie in einer Schleife, mit allen Werten, mit dem in der Konstanten `TEI_INPUT_FORMAT_NAME` gespeicherten Namen. Sollte eine Übereinstimmung gefunden werden, bricht die Schleife sofort ab und die Ausführung der Funktion wird mit einem `return` beendet. Bleibt die Suche erfolglos, wird die Funktion `_teichi_add_input_format` aufgerufen. Dort wird zuerst eine Liste von Rollen aus der Datenbank geholt. Die Rechteverwaltung von Drupal wird über Rollen gesteuert. So darf nur derjenige Inhalte diesen Typs erstellen, der über die entsprechende Rollen verfügt. Hier wurde der weniger restriktive Ansatz gewählt, indem zuerst alle vorhandenen Rollen, in der Lage sind Inhalte, des erstellten Input-Types zu erstellen. Eine weitere Einschränkung kann der Administrator im Drupal-Menü vornehmen. Die Erstellung des Formats erfolgt in den folgenden Zeilen:

```
  $format = new stdClass();
  $format->name = TEI_INPUT_FORMAT_NAME;
  $format->roles = ',' . implode(',', $roles);

  // assign teichi as filter for this format
  $format->filters = array(
    'teichi' => array(
      'status' => 1,
    ),
  );
[...]
 filter_format_save($format);
```

Es wird ein Format erzeugt und diesem alle gefundenen Rollen zugewiesen. Danach wird es abgespeichert und somit dem System bekannt gemacht, wodurch es bei der Erstellung von Dokumenten zur Verfügung steht.

Ist auch der Input-Type vorhanden wird geprüft, ob vielleicht schon TEI-Dokumente vorhanden sind. Dazu wird die Funktion `_teichi_update_node_infos` aufgerufen, welche dann die Zusatzinformationen aktualisiert.

4.3 XSL-Transformation

Da das XML-Content-Modul aufgrund von API-Änderungen in Drupal 7 nicht mehr funktionierte, galt es zunächst dessen Funktionen oder besser gesagt die daraus benötigte Teilmenge an Funktionalität nachzubilden. Auf Einstellmöglichkeiten, die die Handhabung eher verkomplizierten wurde verzichtet. Dazu gehört zum Beispiel die Auswahl einer Stylesheet-Datei. Auf diese kann verzichtet werden, da eine fertige XSL-Datei zur TEI Konvertierung mitgeliefert wird. Auch die verschiedenen Validierungsmöglichkeiten ließen sich auf wohlgeformtes XML reduzieren, da bei groben Syntaxfehlern sowieso der PHP-XSLT-Prozessor anschlagen würde und eine Validierung anhand DTD oder XML-Schemata lokal stattfinden kann, weil zur Darstellung nur eine Teilmenge des TEI-Lite-Standards verwendet wurde.
In Drupal 7 wurden grundlegende Funktionen zur Node-Darstellung umgeschrieben oder umbenannt. So gab es in Drupal 6 zum Beispiel eine Funktion wie `hook_nodeapi()`, die mehrere Aufgaben übernommen hat und je nach Parameter entschied, welche Aktion ausgeführt wird. Um aber nicht unnötig Code zu laden und eine bessere Trennung zu erreichen, wurde die Funktionalität aufgetrennt [10]. Hooks, die sich im XML-Content-Modul bisher um die Transformation kümmerten, mussten also umgeschrieben werden. Die Hooks `hook_filter()` und `hook_filter_tips()`, die Informationen enthielten ob ein Modul einen Filtertyp anbietet wurden ersetzt und in `hook_filter_info()` zusammengefasst [19]. Der Rückgabewert des implementierten `teichi_filter_info()` ist ein assoziatives Array, welches an dem Index mit dem Namen des Filters ein weiteres Array enthält. Zur Verdeutlichung:

```
function teichi_filter_info() {
  $filters['teichi'] = array(
         'title' => t('XSL filter for TEIChi module'),
         'process callback' => '_teichi_transform',
         'tips callback'  => '_teichi_filter_tips',
         'cache' => FALSE,
 );
```

Pro Filter-Index wird ein Titel, eine Callback-Funktion und eine Tips-Callbackfunktion angegeben. Der Titel wird in einem Drop-Down-Menü, bei der Erstellung neuer Inhalte angezeigt und die Tips-Callback-Funktion liefert eine erweiterte Beschreibung zur Tätigkeit des Filters und wird hier nicht näher beschrieben. Zusätzlich kann noch angegeben werden, ob die verarbeiteten Inhalte von Drupal zwischengespeichert werden sollen. Die wichtigste Angabe ist aber die „process callback“-Option, die beim Zugriff auf einen Knoten aufgerufen wird, sofern er den Filter verwendet, dessen Name mit dem Index dieses assoziativen Arrays übereinstimmt.
Angegeben ist hier die Funktion `_teichi_transform`. Diese wurde zum großen Teil aus

dem XML-Content-Modul übernommen, aber auch um einige spezifische Funktionen erweitert, deren Zusammenhang erst später klar wird, weswegen weiter unten auf sie eingegangen wird.

4.4 Das Konfigurationsmenü

4.4.1 Hilfeseiten

Die wohl am meisten verwendeten Hooks, sind `hook_help` und `hook_menu` [1, Kapitel 16]. Ein Help-Hook bietet die Möglichkeit, eine kurze Beschreibung des Moduls anzugeben und eigene oder bestehende Seiten mit Hilfestellungen, in Form von Meldungen einzublenden. Erreichbar ist das Ganze dann über den „Help“-Bereich von Drupal, in der Administrationsoberfläche.
Da die eigentlichen Einstellungen selbsterklärend oder ausreichend beschriftet sind, hat man sich hier auf das Wesentliche beschränkt. Möglich wäre natürlich auch eine ausführliche Schritt für Schritt Anleitung, um die man das Modul später ergänzen kann. Das Ergebnis sieht folgendermaßen aus:

```
function teichi_help($path, $arg) {
  $output = '';
  switch ($path) {
    case "admin/help#teichi":
      $output = '<p>'.  t("TEI/XML to Drupal
converter") .'</p>';
      break;
  }
  return $output;
}
```

Der Funktion werden ein Pfad und optional auch zusätzliche Argumente übergeben. Fass der Pfad mit dem aus der Switch-Anweisung übereinstimmt, liefert die Funktion den Text, der von Drupal beim Aufruf durch Links zu weiteren Einstellungsmöglichkeiten ergänzt wird. Der zu prüfende Pfad entspricht dabei dem Schema *admin/help#modulename* , wie in Dokumentation [19] beschrieben.

4.4.2 TEI-Upload

Der wichtigste Teil des Moduls, ist das Konfigurationsmenü. Es soll dem Benutzer den

Zugang zu den zahlreichen Einstellmöglichkeiten bieten. Dort wird die TEI-Dokumente betreffende Funktionalität gebündelt. So wird dort der Datei-Upload von Dokumenten und Schriftarten abgewickelt, sowie die Darstellungsoptionen angeboten. Im folgenden Abschnitt werden das Formular, das Hochladen von Dokumenten und deren Verarbeitung näher beschrieben.

Wie so oft, gibt es eine spezielle Hook-Funktion in Drupal, die es erlaubt sich in die vorhandene Menüstruktur einzuklinken. Realisiert wird die Funktionalität also in der Funktion `teichi_menu`. Als Rückgabewert erwartet Drupal ein Array, welches die Menü-Items enthält, die das Konfigurationsmenü des Moduls zur Verfügung stellt. Die Implementierung des Hooks hat also folgende Form:

```
function teichi_menu(){
        $items = array();

        $items['admin/settings/teichi'] = array(
                'title' => t('TEIChi Preferences'),
                'description' => t('Extended TEIChi
configuration'),
                'access arguments' => array('administer site
configuration'),
                'page callback' => '_teichi_main_pref_screen',
                'type' => MENU_NORMAL_ITEM |
MENU_DEFAULT_LOCAL_TASK,
        );

[…]

        return $items;
}
```

Dabei ist das Vorgehen ähnlich dem im Filter-Info-Hook, dass nämlich ein assoziatives Array zurückgeliefert wird, wobei jeder Index einen Menüeintrag symbolisiert. Jeder von ihnen hat auch hier einen Titel und eine Callback-Funktion. Zusätzlich kommen noch eine Beschreibung, Zugriffsrechte und ein Typ hinzu. Der Punkt `'access arguments'` regelt wer diesen Menüpunkt aufrufen darf. Der Typ in der Option „type“ definiert die Art der Darstellung dieses Menüpunktes, wie in der Dokumentation [21] beschrieben. So kann man festlegen ob ein Punkt ein Menü, ein Untermenü oder als Reiter dargestellt werden soll. Angegeben sind vier Menüpunkte, wobei wir uns zunächst den wohl wichtigsten anschauen werden und zwar `'admin/settings/teichi/docs'`. Durch dessen Callback-Argument wird die Funktion `_teichi_doc_upload` aufgerufen, welche wiederum, mit Hilfe von `drupal_get_form` die Funktion `teichi_submit_tei_doc_form` aufruft.

Formular

Die Hauptaufgabe der drupaleigenen Funktion `drupal_get_form` ist es, aus den übergebenen Werten ein HTML-Formular zu erzeugen und zurück zu liefern. Dieses Formular wird in `teichi_submit_tei_doc_form` erzeugt. Dort wird auch der ganze Vorgang des Hochladens und Abspeicherns, sowie das Anlegen neuer Drupal-Nodes vorgenommen. Die Konvention sieht es vor, dass Funktionsnamen von Funktionen, die Formulare erzeugen mit

„_form“ enden. Jede dieser Funktionen bekommt als Parameter `$form` sowie `$form_state` übergeben. Diese enthalten das eigentliche Formular, falls es schon erzeugt wurde, die Variablen mit Werten und den Zustand, der angibt ob zum Beispiel ein Eingabefehler vorlag oder das Formular abgesendet wurde. Auch in diesem Fall ist der Rückgabewert ein assoziatives Array, dessen Indizes die einzelnen Formularfelder darstellen. Um einen Upload von Dateien zu ermöglichen, wurde ein Datei-Element benötigt. Zusätzlich sollte es möglich sein, direkt beim Hochladen den Typ des eingestellten Inhalts zu bestimmen. Die Knotentypen sind bei der Grundinstallation „Basic Page“ und „Article“. Hat der Benutzer ein Modul aktiviert, das weitere Typen definiert, wie zum Beispiel das Book-Modul, sollten auch diese zur Auswahl stehen. Das Holen dieser Information geschieht mit der Funktion `_teichi_get_node_types()`. Diese fragt vorhandene Typen von Drupal ab und formatiert sie in ein Array, das an das '#options'-Feld des entsprechenden Formulars übergeben wird. Dieses erzeugt ein Drop-Down-Menü, welches dem Benutzer diese Typen zur Auswahl stellt. Das besondere hier ist, dass wenn als Node-Typ 'Book page' ausgewählt wird, auch ein Book-Drop-Down zur verfügung steht. Das heißt, dass das neu erstellte Dokument direkt einem vorhandenen „Book“ zugewiesen werden kann. Sind andere Typen als „Book page“ ausgewählt, wird dieser Eintrag ignoriert.
Die betreffenden Stellen im Code sind also zunächst in der Funktion `teichi_menu` beim Item „`admin/settings/teichi/docs`“:

```
$items['admin/settings/teichi/docs'] = array(
  'title' => t('Documents'),
  'description' => t('TEIChi documents'),
  'access arguments' => array('administer site
configuration'),
  'page callback' => '_teichi_doc_upload',
  'type' =>  MENU_NORMAL_ITEM | MENU_LOCAL_TASK |
MENU_DEFAULT_LOCAL_TASK,
);
```

Die Callback-Option in diesem Teil sorgt dafür, dass beim Aufruf des entsprechenden Pfades `_teichi_doc_upload` die Formularfunktion `teichi_submit_tei_doc_form` aufruft. In dieser wird das Formular erzeugt nach dessen Abschicken die Funktion `teichi_submit_tei_doc_form_submit` angestoßen wird. Die Namensgebung entspricht auch hier der Drupal-Konvention.

Upload

Alle Formularfunktionen können das Verhalten nach dem Absenden der Formulardaten, durch Angabe einer Funktion in der Form: <Modulname>_<Formularname>_form_submit() steuern. Die Hauptaufgabe von `teichi_submit_tei_doc_form_submit` ist also, das Überprüfen der Nutzereingaben auf Richtigkeit und eine entsprechende Reaktion darauf. Im vorliegenden Fall, prüft die Funktion zunächst, ob die übergebene Datei eine der erlaubten Dateiendungen hat. Das geschieht indem man eine Validators-Variable definiert und die zugelassenen Endungen in einem Array übergibt. Ausgewählt wurden xml, tei und txt. Wobei diese Prüfung keine Sicherheitsabfrage darstellt, sondern eine reine „Convenience-Funktion“ gegen das versehentliche Hochladen falscher Dateien darstellt, da nicht der Dateiinhalt, sondern nur die Dateiendung entscheidend ist. Stellt Drupal fest, dass eine andere Dateiendung vorliegt, bricht es ab und weist den Nutzer auf das Formular zurück, in dem eine

entsprechende Warnung ausgegeben wird. Ist die Eingabe korrekt, wird versucht aus den übergebenen Daten ein File-Objekt zu erstellen. Das geschieht mit Hilfe der Funktion `file_save_upload` die entweder NULL, FALSE oder ein valides Datei-Objekt zurückliefert. Ist der Rückgabewert NULL, hat das Formular keine Datei erhalten. Der Nutzer wird also zum Formular zurückgeschickt mit dem Hinweis, dass die Datei zur Dokumenterstellung fehlt. Ist das File-Objekt ungleich NULL und auch nicht FALSE kann mit der Verarbeitung des Inhalts fortgefahren werden. Sollte die Prüfung FALSE ergeben, so lag ein Fehler vor und es wird auch hier die entsprechende Fehlermeldung ausgegeben. Da eigentlich nur mit dem Dateiinhalt gearbeitet wird und nicht mit der Datei an sich, braucht man nur den Pfad zur temporären Datei, die Drupal in einem dafür vorgesehenen Verzeichnis ablegt. Solche Dateien werden normalerweise nach 6 Stunden gelöscht.[22].

Verarbeitung

Die Verarbeitung der Datei übernimmt die Funktion `_teichi_tei_node_create`. Diese erwartet als Parameter den Pfad zur Datei, den Knotentyp, der beim Hochladen angegeben wurde und als optionalen Parameter ein „Book“, falls es sich beim zu erstellenden Inhalt um eine Buchseite handelt, die direkt zugewiesen werden soll. Geholt wird zunächst eine Liste der in Drupal vorhandenen Filterformate, welche zwischengespeichert wird. Danach wird der gesamte Dateiinhalt der hochgeladenen Datei in einer Variable abgelegt. Die Variable wird an `_teichi_get_tei_node_info` übergeben, wo der XML-Inhalt auf seine Richtigkeit überprüft wird. Falls es sich beim Inhalt um valides XML handelt, wird versucht den TEI-Header, also den Dokumentteil mit Metainformationen und alle vorhandenen Seitenzahlen einzulesen. Das „title“-Element aus dem Header wird zwischengespeichert und das Format der Seitenzahlen wird auf die richtige Form hin überprüft. Erwartet wird eine XML-ID die sich aus dem Buchstaben „p“, gefolgt von einer Zahl, zusammensetzt. Da für das Modul nur die erste und letzte Seite von Bedeutung sind, werden nur diese geprüft. Andere Fehler wie zum Beispiel doppelt vergebene XML-IDs werden vom XML-Parser mit einer Warnung quittiert, was aber keine weiteren Folgen hat [28]. Wurde das Dokument erfolgreich analysiert, wird ein Array zurückgeliefert, welches die Information, ob das Parsen erfolgreich war, Anfangs- und Endseite, sowie den Titel des Dokuments enthält. Diese Informationen gehen an `_teichi_tei_node_create` zurück. Dort wird zusätzlich, der aktuell eingeloggte Benutzter und die ID des Filterformats ermittelt. Sind diese Daten vorhanden, kann das Modul ein Node-Objekt erzeugen, welches dann mit `node_submit($node)` Drupal bekannt gemacht wird. Damit wird Drupal angewiesen einen neuen Knoten in der Datenbank abzulegen. Der Rückgabewert dieses Aufrufs ist die Node-ID über die der Knoten in der Datenbank eindeutig identifiziert werden kann. Da so ein Knoten keine Felder für Informationen wie Start- und Endseite besitzt, werden diese Daten in eine dafür eigens angelegte Tabelle gespeichert. Das Erstellen dieser Tabelle geschieht in der teichi.install-Datei. Dort sind die Hooks hinterlegt, die beim installieren die Tabelle `teichi_node_infos` erstellen. Die Tabelle hat folgendes Schema:

```
+-----------+---------+------+-----+---------+-------+
| Field     | Type    | Null | Key | Default | Extra |
+-----------+---------+------+-----+---------+-------+
| nodenr    | int(11) | NO   | PRI | 0       |       |
| startpage | int(11) | NO   |     | 0       |       |
| endpage   | int(11) | NO   |     | 0       |       |
+-----------+---------+------+-----+---------+-------+
```

Jeder Eintrag ist somit über die eindeutige Node-ID, hier als „nodenr“, welche den Primärschlüssel darstellt, identifizierbar und speichert die Seitennummern, welche für die Index-Funktion benutzt werden (vgl. 3.6). Versucht wurde hier auch die Drupal Field-API [44] zu benutzen. Das ist eine mit Drupal 7 eingeführte Schnittstelle, die es erlaubt einzelne Datenfelder an einen Knoten, mit bestimmtem Inhalt, Sprache oder Revision anzuheften. Diese Idee wurde jedoch verworfen, da der Aufwand für die Pflege dieser Informationen ziemlich hoch ist und die Komplexität diesen Aufwand nicht rechtfertigt. Deswegen erwies sich der Ansatz mit einer eigenen Tabelle, als die elegantere und effizientere Lösung und es wurde stattdessen auf die Datenbank-API [45] gesetzt. Drupal bietet einen komfortablen Datenbank-Layer, der die Benutzung verschiedener Datenbank-Backends für den Programmierer abstrahiert. Das Schreiben in die Datenbank erfolgt in der Funktion `_teichi_save_new_node_info`, welche als Parameter die Node-Nummer, die Start- und die Endseite erwartet. Diese prüft zunächst, ob es sich bei den übergebenen Werten auch wirklich um Zahlen handelt. Anschließend werden die Daten in die Tabelle geschrieben. Das geschieht mit dem Aufruf der Datenbank-Funktion `db_merge`, diese hat die Besonderheit, dass je nachdem, ob der Wert für einen bestimmten Primärschlüssel existiert oder nicht, ein Tabelleneintrag anlegt oder aktualisiert wird [1, S.363].

Was danach passiert, betrifft den JavaScript-Teil des Moduls, auf den später eingegangen wird. Es wird eine Datei erzeugt, welche die neu abgelegten Informationen als ein Array von JavaScript-Objekten speichert. Diese Daten werden dann zum Anspringen von Seiten verwendet. Schlägt das Anlegen der Datei fehl, wird eine Warnung ausgegeben, dass die Schreibrechte auf dem Server womöglich angepasst werden müssen. War der Durchlauf erfolgreich, ist die Submit-Funktion abgearbeitet und gibt eine Meldung aus, die angibt was für ein Knoten erzeugt wurde und ob und wenn ja, welche Seitenzahlen gefunden wurden.

4.4.3 Font-Upload

Eine weitere Einstellung, sollte es ermöglichen eigene Schriftarten zu verwenden. Dazu war auch hier ein Datei-Upload notwendig, um die Font-Dateien im entsprechenden Dialog auswählbar zu machen. Anders als beim Upload von TEI-Dokumenten, wo nur deren Inhalt benötigt wurde, war es hier notwendig, die gesamte Datei permanent auf dem Server, in einem für das Modul zugänglichen Ort, zu speichern.

In den Dialog gelangt man wenn der Menü-Hook von Drupal auf den einstellten Pfad zutrifft. In dem Fall „admin/settings/teichi/fonts“. Dieses Menü-Item hat den Page-Callback auf die Funktion `_teichi_font_upload` gesetzt, welche mit Hilfe von `teichi_upload_font_form` das Formular für den Upload erstellt. Wie bei Formularfunktionen üblich werden dieser die Variablen `$form` und `$form_state` übergeben. Auch hier wird eine Validators-Variable benutzt, um die erlaubten Dateiendungen zu bestimmen. Die Variable wurde auf die, zur Zeit der Erstellung des Moduls bekannten Endungen „ttf“, „otf“ und „woff“ gesetzt, die in einer Konstante abgelegt wurden, die im ganzen Modul abgerufen werden kann. Das Formular ist recht simpel aufgebaut. Es besitzt ein Dateifeld und einen Knopf zum Abschicken. Das besondere am Dateifeld ist, dass es sich hierbei um ein in Drupal 7 eingeführtes Element handelt. Das Managed-File-Feld [22]

erweitert das einfache aus HTML-Formularen bekannte Feld um Ajax[2]-Fähigkeiten. Das heißt, dass die Ausgewählte Datei asynchron hochgeladen werden kann, ohne dass das ganze Dokument neu geladen werden muss. Um die Verarbeitung der Daten nach dem Abschicken kümmert sich auch hier die Submit-Funktion, die hier `teichi_upload_font_form_submit` heißt. Hat Drupal die Font-Datei erfolgreich hochgeladen, wird ein File-Objekt angelegt und die `$form_state-Variable`, beziehungsweise dessen Referenz, an die Funktion übergeben. Dort wird unter dem Index „value“, in einem assoziativen Array eine File-ID abgelegt. Ist diese ungleich 0, war der Upload erfolgreich. Die Datei kann permanent gespeichert werden, indem ihr Status, beziehungsweise der des File-Objekts, welches über die ID geholt wurde, auf permanent gesetzt wird.

```
$file = file_load($form_state['values']['teichi_new_font']
['fid']);
$file->status = FILE_STATUS_PERMANENT;
file_save($file);
```

Im anderen Fall wurde die Datei wieder entfernt oder bei der Übertragung ist ein Fehler aufgetreten. Es wird also, falls vorhanden, das File-Objekt wieder entfernt und eine Meldung ausgegeben.

4.4.4 Display-Einstellungen

Das Einstellungsmenü für die Anzeige, soll es dem Benutzer ermöglichen die Darstellung der TEI-Dokumente zu einem gewissen Grad zu beeinflussen und an seine Drupal-Installation anzupassen. Auch diese Einstellungen bestehen aus einem Formular. Um Platz zu sparen sind die Werte in sogenannte „Fieldsets“ organisiert. Diese fassen semantisch zusammengehörige Variablen optisch zu einer Gruppe zusammen. Durch die, dank JavaScript, dynamische Drupal-Oberfläche sind die Fieldsets auch zusammenklappbar, was bei der Menge an Optionen hilft den Überblick zu behalten. Die Funktion `teichi_display_prefs_admin_settings` wird, wie die anderen Menüfunktionen, aus dem Menü-Hook über die Callback-Option aufgerufen. Eine Besonderheit hierbei ist, dass dieses Formular keine Submit-Funktion benötigt. Am Ende wird das Formular einfach an `system_settings_form` übergeben, welches die weitere Verarbeitung übernimmt. Bei `system_settings_form` handelt es sich um eine Drupal-Funktion, die automatisch Variablen für das Modul anlegt und speichert [1, S.354]. Jede im Formular definierte Variable, kann also in Drupal abgelegt werden und ist dann systemweit verfügbar. Ist die Variable noch nicht gesetzt, kann das beim Holen der Werte per `variable_get`-Funktion berücksichtigt werden, indem man einen Standardwert mitgibt, welcher dann stattdessen verwendet wird. Für dieses Formular wurden vorher definierte Konstanten benutzt, die zu Beginn der Moduldatei definiert sind. Das erleichtert die Lesbarkeit und die Wartung des Moduls, da man alle vordefinierten Werte wie Zeichenketten, an einer zentralen Stelle versammelt vorfindet. Die Definition des Feldes für die Schriftart des Haupttextes zum Beispiel sieht folgendermaßen aus:

2 Asynchronous JavaScript and XML

```
$form['fonts']['teichi_maintext_font'] = array(
  '#title' => t('Main text'),
  '#type' => 'select',
  '#options' =>
drupal_map_assoc($fonts,"_teichi_font_extension"),
  '#default_value' =>
variable_get('teichi_maintext_font',TEICHI_DEFAULT_FONT),
  '#description' => t('Font used for main text')
);
```

Hierbei handelt es sich um das aus HTML bekannte Select-Element, welches im Browser ein Drop-Down-Menü erzeugt und vorgegebene Werte zur Auswahl stellt. Der anfangs ausgewählte Wert in der Liste, wird über die „default-value“-Option definiert. Diese wird mit der oben beschriebenen „variable_get“-Funktion gesetzt, die den aktuell gespeicherten Wert als ausgewählt markiert. Sollte noch kein Wert vorhanden sein, wird die Konstante `TEICHI_DEFAULT_FONT` benutzt, welche im oberen Bereich des Moduls durch:

```
define ("TEICHI_DEFAULT_FONT","Sans-Serif");
```

festgelegt wird.
Eine weitere Besonderheit, ist die `drupal_map_assoc`-Funktion. Da die Werte im Select-Element aus Key-Value-Paaren aufgebaut sein können, bietet diese Funktion die Möglichkeit sie entsprechend zu formatieren und an die Option weiter zu geben. Als Parameter erwartet wird ein Array mit Werten und der Name der Funktion, die für jeden Wert, mit diesem Parameter ausgeführt werden soll. In diesem Fall ist es „`_teichi_font_extension`“, diese ist recht simpel aufgebaut:

```
function _teichi_font_extension($value,$append_type = TRUE){

  if(isset($value) && trim($value) !== "")
  {
    $needle = strrpos($value,'.');
    if($needle)
    {
      return substr($value,0,$needle).($append_type?"
(".substr($value,$needle+1).")":"")  ;
    }else{
      return $value;
    }
  }
}
```

Der Funktion wird als Parameter der Dateiname einer Font-Datei übergeben. Ein weiterer optionaler Parameter gibt an, ob die Endung beim Rückgabewert mit angegeben werden soll und ist standardmäßig TRUE, falls kein expliziter Wert gesetzt wird. Was dann passiert ist eine Zerlegung des Dateinamens in zwei Teile, den Namen und die Dateiendung. Der Rückgabewert ist dann das Ergebnis dieser Aufteilung als String. Das Array welches die

Dateinamen enthält, wird im Formular erstellt. Es enthält zunächst nur die Font-Klassen Sans-Serif, Serif und Monospace, für den Fall dass noch keine Font-Dateien hochgeladen wurden. Weitere Schriftarten werden über die Funktion `_teichi_get_font_list()` geholt. Dort wird durch den vorher definierten Upload-Ordner iteriert. Hat eine enthaltene Datei die gewünschte Endung, wird die in ein Array abgelegt, dass dann zurückgeliefert wird. Dieses bildet dann, zusammen mit den Font-Klassen, die Liste der Auswahlmöglichkeiten in der Drop-Down-Box, nach dem die Endungen entfernt und in Klammern hinter den Name angehängt wurden. Nach dem gleichen Prinzip geschieht der Vorgang auch für die Werte der anderen Font-Variablen wie Notiz- und Menütext. Außer den Drop-Down-Menüs gibt es noch ganz normale Textfelder, die für Werte wie vordefinierte Texte und Größenangaben mit Einheit (zum Beispiel px, em etc.) benutzt werden. Auch hier kann auf Standardwerte aus vordefinierten Konstanten zurückgegriffen werden, falls die Variablen noch nicht gesetzt wurden. Eine besondere Herausforderung waren die Felder für die Farbwerte. Da man es dem weniger technik-affinen Benutzern ersparen will hexadezimale Farbwerte einzugeben, wurde ein sogenannter Colorpicker verwendet. Der Colorpicker ist eine Graphische Eingabehilfe für Farbwerte, die einfach mit der Maus, aus einem mit Hilfe von JavaScript dargestellten Farbrad, ausgewählt werden können. Die ausgewählte Farbe wird dann als der übliche hexadezimale RGB-Farbwert in das dafür vorgesehene Feld eingesetzt, so dass eine manuelle Anpassung immer noch möglich ist und die Farbauswahl auch bei deaktiviertem JavaScript funktioniert, wenn auch nur eingeschränkt. Benutzt wurde hierfür die „Farbtastic“ JavaScript-Bibliothek, die bei Drupal 7 im Kernpaket enthalten ist. Die Verwendung ist relativ unkompliziert und funktioniert, indem man JavaScript-Code und ein zusätzliches Div-Element an das gewünschte Textfeld anhängt. Der betreffende Code sieht im Modul also wie folgt aus:

```
function teichi_display_prefs_admin_settings($form,
&$form_state){
[…]
drupal_add_library('system', 'farbtastic');
[…]
// Fieldset
        $form['colors'] = array(
                '#title' => t('Colors'),
                '#type' => 'fieldset',
                '#collapsible' => TRUE,
                '#collapsed' => TRUE
        );
// Texteld für Farbe
        $form['colors']['teichi_note_color'] = array(
                '#title' => t('Notes'),
                '#type' => 'textfield',
                '#size' => 8,
                '#maxlength' => 7,
                '#default_value' =>
variable_get('teichi_note_color','#e3e9f0'),
                '#description' => t('Color used for notes'),
                '#prefix' => '<div class="teichi-form-
separator"><div id="custom-note-color-picker"
```

```
style="float:right;"></div>',
                '#suffix' => '</div>',
        );
// angehängter Code
        $form['colors']['teichi_note_color']['#attached']
['library'][] = array(
                'system',
                'farbtastic',
        );

        $form['colors']['teichi_note_color']['#attached']
['js'][] = array(
           'data' => '(function($) { $(document
).ready(function() { $("#custom-note-color-
picker").farbtastic("#edit-teichi-note-color"); }); }) (jQuery
);',
           'type' => 'inline',
        );
```

In der Funktion `teichi_display_prefs_admin_settings` wird zuerst die benötigte Bibliothek eingebunden. Später folgt ein Fieldset, das die Farbauswahlfelder gruppiert, da sie sonst zu viel Platz in Anspruch nehmen würden. Die Variable `$form['colors']['teichi_note_color']` speichert den Wert für die Farbe der dargestellten Notizen, am Rand des Dokuments. Der Typ ist als „textfield“ definiert. Zusätzlich wird durch Angabe der Optionen „#prefix“ und „#suffix“ ein HTML-Div eingefügt in den der JavaScript-Code den Colorpicker platziert und ein weiteres Div-Element um das Eingabefeld und die Farbauswahl, auf der angezeigten Menüseite, nebeneinander darzustellen. Dahinter wird eine an das Textfeld angehängte Variable definiert, die den Code enthält um einen Colorpicker zu initialisieren. Somit wird veranlasst, dass beim Laden der Seite das Farbauswahl-Widget in das dafür vorgesehene Div-Element geladen wird.

Was noch zu erwähnen bleibt, sind die einstellbaren Texte, die man frei wählen kann. Diese finden Verwendung im JavaScript-Teil. So kann man eigene Texte definieren, die zum Beispiel bei Warnungen ausgegeben werden, unabhängig von der in Drupal eingestellten Sprache, sowie anderen Themen- und Übersetzungsdateien, für die man tiefer ins System eingreifen muss.

Nach dem Durchlaufen der Funktion wird `_teichi_css_file` aufgerufen. Diese wurde in die Datei „teichi.cssfile.inc“, im Modulverzeichnis, ausgelagert. Die Hauptaufgabe dieser Funktion, die quasi die ganze Datei einnimmt, ist es eine CSS-Datei zu erzeugen, die später beim Seitenaufruf eingebunden wird und somit die eingestellten Darstellungsanpassungen auf die TEI-Dokumente anwendet. Zunächst werden die Werte in Variablen geladen:

```
  $note_color = variable_get('teichi_note_color','lime');
  $menubar_color =
variable_get('teichi_menubar_color','magenta');
  $reg_highlight =
variable_get('teichi_reg_highlight_color','magenta');
  $corr_highlight =
```

```
variable_get('teichi_corr_highlight_color','magenta');
```

Dazu werden sie über die schon geschriebene `variable_get` Funktion abgerufen und falls noch nicht gesetzt mit Standardwerten belegt. Dann wird das Verzeichnis in dem die Schriftarten abgelegt werden ermitteln. Das geschieht mit der Funktion `file_create_url` mit Hilfe der voreingestellten Konstanten `TEICHI_FONT_DIR`, die als Parameter übergeben wird. Der Rückgabewert, ist eine über das Web erreichbare Adresse, die notwendig ist, damit man dort den Namen einer Schriftartendatei anhängen kann, um diese im CSS-Dokument korrekt zu verlinken. Das passiert im nächsten Schritt, nachdem geprüft wurde, ob es sich bei den einstellten Schriftarten nicht um die Standardfontklassen wie Sans-Serif oder ähnliche handelt, die nicht extra eingebunden werden müssen. Sollten also externe Schriftarten vorkommen, müssen diese per im CSS3-Entwurf definierten „@font-face“ eingebunden und dem Dokument bekannt gemacht werden [15]. Eine Font-Face-Anweisung definiert eine „font-family“, die Quelldatei und das Format einer Ressource, die im weiteren CSS-Dokument, als Schriftart für diverse HTML-Tags und Klassen definiert werden kann, wie man es von Font-Klassen oder Schriftarten kennt, die auf nahezu jedem System vorhanden sind, wie Times-New-Roman, Arial und ähnliche [16].

Sind die Schriftarten eingelesen, wird das komplette CSS-Dokument in einer Variablen abgelegt. Benutzt wird hierfür die „here Document“-Notation. Dabei wird der String-Anfang mit „<<<“, einem Terminatorzeichen markiert und mit dem Terminatorzeichen allein, in einer Zeile ohne Leerzeichen mit einem Semikolon dahinter, abgeschlossen [18].

```
$css=<<<CSS
{$teichi_maintext_font_def}
{$teichi_notes_font_def}
{$teichi_menu_font_def}

[...]

#inner-content,#center{font-family:
{$teichi_maintext_font_family},serif;}
.maincontent{font-family:
{$teichi_maintext_font_family},serif;}
.maincontent{font-size:{$main_text_size};margin-
left:10%;width:64%;text-align:justify;line-height: 2;}
[...]
CSS;
```

In dem String werden Variablen an die Stellen gesetzt, an denen ein benutzerdefinierter Wert eingefügt werden soll. Am Ende der Funktion wird mit:

```
$cssPath =
drupal_get_path('module','teichi')."/css/teichi.css";
file_put_contents($cssPath,$css);
```

der Pfad der CSS-Datei bestimmt und mit `file_put_contents` der Inhalt der Variablen dorthin geschrieben.

4.4.5 Hauptmenü

Das Hauptmenü bietet eine simple Auflistung vorhandener Dokumente. Jedes TEI-Dokument wird mit seinem Titel aufgeführt. Hinter dem Titel befinden sich Verweise zum Editieren oder Löschen der Nodes und dahinter, falls vorhanden, die Anfangs- und Endseiten. Das Hauptmenü kommt zum Vorschein, wenn in der Menüfunktion beim Aufruf, der Pfad „admin/settings/teichi“ übergeben wird. Als Callback-Funktion wird dann `_teichi_main_pref_screen`“ aufgerufen. Dort wird zunächst ein Array aus Node-Objekten, die TEI-Dokumente enthalten, zusammengebaut. Das geschieht mit der Funktion `teichi_get_teichi_nodes`, welche eine Datenbankabfrage ausführt, die alle Nodes mit dem TEIChi-Filter aus der Node-Tabelle sammelt. Zuvor wird die ID des Formats ermittelt, indem mit `filter_formats`, eine Liste vorhandener Formate geholt wird und nach dem Input-Type mit dem Namen, der in `TEI_INPUT_FORMAT_NAME` gespeichert ist, gesucht wird. In einer foreach-Schleife, wird dann durch die zurückgelieferten Nodes iteriert und die entsprechenden Informationen an den Ausgabestring angehängt. Dabei werden die Links zu den Anzeige- und Löschfunktionen mit Hilfe von `url` erzeugt. Dies ist eine Drupal eigene Funktion zum Erstellen von Verweisen zu internen und extern URLs. Als Parameter wird der Pfad erwartet, zum Beispiel in der Form „node/34“, was zu einem Link als Rückgabewert führt der, wie hier, in einem HTML-Link verwendet werden kann.
Drupal erkennt dann anhand der URL, dass es sich um einen Node-Link handelt, extrahiert daraus die Nummer und, falls vorhanden, auch die Aktion die durchgeführt werden soll, wie zum Beispiel das Löschen. Die gesamte Zeichenkette wird dann an die Menüfunktion zurückgeliefert und als Node-Auflistung dargestellt.

4.5 Darstellung

4.5.1 Input-Type

Der den TEI-Knoten zugewiesene Input-Type heißt „teichinode“ und wird jeder Node, beim Erstellen von Dokumenten, zugewiesen. Das veranlasst Drupal bei jedem Aufruf dieser Nodes die zugewiesene Transform-Funktion auszuführen. In diesem Fall ist es die schon erwähnte und teilweise aus dem XML-Content-Modul stammende `_teichi_transform`, welche in der Filterinformationen als Callback festgelegt wurde. Sie kriegt als Parameter unter anderem den XML-Code übergeben, der wie schon besprochen, vom XSLT-Prozessor verarbeitet wird. Daneben werden aber auch zahlreiche Dateien eingebunden, die an der Darstellung mitwirken. Als wichtigste Datei wird das Stylesheet eingebunden per:

```
$path_to_xslt =
```

```
drupal_get_path('module','teichi').'/xsl/drp_style.xsl';
```

Dieses Stylesheet ist im Gegensatz zum XML-Content-Modul fest im TEIChi-Modul verankert und deswegen nicht ohne weiteres änderbar. Auch die JavaScript- und CSS-Dateien werden auf ähnliche Weise ermittelt und eingebunden, wie man später sehen wird (vgl. 4.5.2).

Zunächst wird das, an die Transform-Funktion übergebene, XML-Dokument eingelesen und validiert:

```
  $dom = new DomDocument('1.0', 'UTF-8');
  @$valid = $dom->loadXML($xml);
  if (!$valid) {
      watchdog('teichi', "Invalid XML/TEI Content", array(),
WATCHDOG_WARNING);
    return $xml;
  }
```

Daraufhin wird eine neue Instanz der XSLT-Prozessors angelegt:

```
  $xsl = new DomDocument('1.0', 'UTF-8');
  $xsl->load($path_to_xslt);

  // Create the XSLT processor
  $proc = new XsltProcessor();
  $xsl = $proc->importStylesheet($xsl);
```

Ist beides erfolgreich durchgelaufen, wird das Stylesheet auf das Dokument angewendet:

```
$newdom = $proc->transformToDoc($dom);
$out = $newdom->saveXML();
return $out;
```

Das transformierte Dokument wird dann in einer Variablen gespeichert und an Drupal zurückgeliefert, wo es angezeigt oder wahlweise von anderen Filtern durchlaufen werden kann.

4.5.2 JavaScript/jQuery

Der alte Ansatz des JavaScript-Teils, war eher statisch und vieles war hartcodiert. Das heißt, dass bei Änderungen Eingriffe in den Quellcode per Hand notwendig waren. Auch war die Verwendung mehrerer Bücher, mit Hilfe des Book-Plugins nicht vorgesehen. Das bedeutet, wenn dort eine Seitennummer mehrfach vorkam, auch wenn es sich um verschiedene Bücher handelte, wurde der erste Treffer angesprungen. In einer externen JavaScript-Datei, wurde ein Array von Kapitel-Objekten angelegt. Also ein Objekt, bestehend aus Anfangs- und Endseite und die zugehörige Node-Nummer pro Kapitel. Dieser Ansatz musste nun um die Information des Buchs erweitert werden. Dazu wurde das Kapitel-Objekt um ein Feld „book“ erweitert,

sodass das Navigieren zu einer bestimmten Buchseite, jetzt noch die Information benötigt, in welchem Buch es sich befindet. In JavaScript sieht das folgendermaßen aus:

```
function Chapter(node,start,end,book)
{
        this.start = start;
        this.end = end;
        this.node = node;
        this.book = book;
}
```

Das Anlegen eines neuen Kapitels funktioniert dann durch Aufruf des new-Operators. Möchte man also Kapitel-Objekte in einem Array ablegen, sieht die Implementierung zum Beispiel so aus:

```
var chapters = new Array( new Chapter(2,28,68,2),
 new Chapter(3,670,685,0),
 new Chapter(4,536,578,0),
 new Chapter(5,2,26,2),
 new Chapter(6,536,578,2)
);
```

Das Book-Modul bietet die Möglichkeit mehrere Nodes zu einem Buch zusammen zu fassen, welche dann zu einem übergeordneten Knoten gehören. Die Informationen hierzu, legt das Modul in einer eigenen Tabelle ab die folgende Struktur hat:

```
+-------+------------------+------+-----+---------+-------+
| Field | Type             | Null | Key | Default | Extra |
+-------+------------------+------+-----+---------+-------+
| mlid  | int(10) unsigned | NO   | PRI | 0       |       |
| nid   | int(10) unsigned | NO   | UNI | 0       |       |
| bid   | int(10) unsigned | NO   | MUL | 0       |       |
+-------+------------------+------+-----+---------+-------+
```

Wichtig hierbei sind die Spalten „nid“ und „bid“, wobei „nid“ die Node-Nummer und „bid“ die Book-Id darstellen. Sind „nid“ und „bid“ gleich, so handelt es sich um einen übergeordneten Knoten, also ein Buch, dem andere Nodes zugewiesen werden können.
Da man beim Aufruf einer Seite deren Node-Nummer über die URL-Parameter kennt, kann hierzu die zugehörige Book-Id aus der Datenbank ermittelt werden. Das passiert in der `_teichi_transform`-Funktion des Filters, indem:

```
if (arg(0)=='node' && is_numeric(arg(1)))
{
  $book = _teichi_get_book_for_node(arg(1));
}
```

aufrufen wird. Wobei zuerst geprüft wird ob das erste Argument gleich „node“ ist und ob das zweite eine Zahl ist. Die Drupal-Funktion `arg` bietet den Zugriff auf die URL-Parameter,

wobei die Zählung bei null anfängt. Die Funktion zum Holen der Buchinformation sieht nun folgendermaßen aus:

```
function _teichi_get_book_for_node($node){

  if(!module_exists('book') || !is_numeric($node) || $node < 1
)
  {
    return 0;
  }else{
    $result = db_query("select n.nid,b.bid,n.title from {book}
b inner join {node} n on n.nid = b.nid where b.nid = :node"
,array(':node'=>$node));

    $res = $result->fetch();
    return (isset($res->bid)?$res->bid:0);
  }
}
```

An `_teichi_get_book_for_node` wird die Node-Nummer als Parameter übergeben. Danach wird geprüft ob das Book-Modul überhaupt aktiv ist, also auch ob die benötigte Tabelle vorhanden ist und ob der übergebene Parameter eine gültige Zahl größer 1 ist. Sollte nichts davon zutreffen, wird 0 als Buch-ID zurückgeliefert. Da die Node-Zählung bei 1 anfängt kann man die Null ohne Kollisionen oder Mehrdeutigkeiten als ID für „keinem Buch zugehörig“ verwenden. Handelt es sich beim Parameter um eine gültige Node-Nummer, wird eine Datenbankabfrage erstellt. Diese Anweisung stellt einen Join zwischen den beiden Tabellen „book“' und „node“ über die Spalte „nid“ dar, die in beiden vorhanden ist. Die Where-Bedingung schränkt das Ergebnis dann noch auf die übergebene Node-Nummer ein. Die zurückgelieferte Zeile wird dann per fetch-Funktion der Datenbank-API geholt, gefolgt von einer return-Anweisung, die je nachdem ob eine Buch-ID gesetzt ist oder nicht, eben diese oder null zurückliefert.

Das zurückgelieferte Ergebnis wird direkt in den HTML-Code des Knotens als JavaScript-Variable geschrieben. Das geschieht mit:

```
drupal_add_js("var currentBook = ".($book!=NULL?$book:0).";",
'inline');
```

indem man den Code als „inline“ definiert, anstatt eine externe Datei anzugeben. Auf diese Weise, ist die Information zu welchem Buch ein Kapitel gehört, im Kapitel selbst vorhanden und die Seitensuchfunktion kann die Suchanfrage entsprechend filtern.
Die restlichen JavaScript-Funktionen werden per

```
drupal_add_js(drupal_get_path('module','teichi').'/js/drp_teic
hi.js');
drupal_add_js(drupal_get_path('module','teichi').'/js/drp_main
.js');
```

aus externen Dateien inkludiert. Die Datei „drp_main.js“ enthält die Logik und „drp_teichi.js“ wird automatisch vom Modul erzeugt. Sie enthält die Struktur eines Kapitel-Objekts und die aktuellen Informationen zu den einzelnen Kapiteln als Array.

Die Buchkapitel werden in der Funktion `_teichi_save_new_node_info` gespeichert. Diese kriegt als Parameter die Node-Nummer, die Start- und die Endseite übergeben. Nachdem diese Informationen in der Modul eigenen Tabelle abgelegt wurden, wird der Pfad für die zu schreibende JavaScript-Datei ermittelt:

```
$nodeInfoFileDrpPath = drupal_get_path('module','teichi');
$nodeInfoFile = drupal_realpath($nodeInfoFileDrpPath);
$js_file = $nodeInfoFile."/js/drp_teichi.js";
```

dann werden die Daten, mit Hilfe eines Joins mit der Book-Tabelle, ausgelesen. Dazu wird folgende Funktion aufgerufen:

```
function teichi_get_node_infos()
{

  $query = "";
  if(module_exists('book'))
  {
     $query = "SELECT * FROM {teichi_node_infos} t LEFT OUTER
JOIN {book} b ON t.nodenr = b.nid";
  }else{
    $query = "SELECT * FROM {teichi_node_infos}";
  }
  $db_dump = db_query($query);

  return $db_dump->fetchAllAssoc('nodenr');

}
```

Es wird zunächst wieder geprüft, ob das Book-Modul aktiv ist, also ob zusätzliche Informationen zu einer Buchzugehörigkeit und somit ein Join überhaupt notwendig ist. Sollte das nicht der Fall sein, findet eine simple Select-Anfrage statt. Ansonsten wird ein Left-outer-join durchgeführt und zwar anhand der Spalten „nodenr“ und „nid“ aus den jeweiligen Tabellen. Somit wird sichergestellt, dass alle Nodes, auch ohne zugewiesenes Buch, im Ergebnis der Abfrage enthalten sind. Der Rückgabewert wird dann in einer foreach-Schleife durchlaufen und in den vorher ermittelten Dateipfad geschrieben.
Aktualisiert werden die Informationen dazu noch beim (Re-)Aktivieren des Modul, durch den Aufruf von `_teichi_update_node_infos`, sowie beim Löschen von Nodes durch die Hook-Funktion `teichi_node_delete`. In `_teichi_update_node_infos` wird eine Liste von TEI-Dokumenten ermittelt und in einer Schleife wird für jedes Dokument geprüft, ob Informationen zu Start- und Endseite vorhanden sind. Sind die Seiteninformationen gegeben, wird `_teichi_save_new_node_info` ausgeführt, welches zum Schreiben in die Datenbank, die db_merge-Funktion benutzt. Das heißt, je nachdem ob ein Knoten mit der angegebenen Nummer vorhanden ist oder nicht, wird ein Befehl zum Aktualisieren (update) oder zum Einfügen (insert) ausgeführt, was doppelte Einträge oder Fehler, durch verletzte Datenbankrichtlinien verhindert. Beim Löschen von Knoten werden diverse Hook-Funktionen aufgerufen, darunter auch `hook_node_delete`, dessen Implementierung wie folgt aussieht :

```
function teichi_node_delete($node){

  $count = db_delete('teichi_node_infos')
    ->condition('nodenr', ($node->nid))
    ->execute();

  if($count > 0)
  {
    _teichi_update_node_infos();
  }
}
```

Der gelöschte Knoten wird als Parameter übergeben. Dieser wird in der db_delete-Funktion benutzt um dessen ID als Bedingung zu verwenden. Eine Prüfung ob die Node-Nummer in der Tabelle existiert, findet nicht statt. Sollte ein Knoten mit dieser Nummer nicht enthalten sein, ist der Rückgabewert der Löschfunktion gleich 0, was die Anzahl der gelöschten Zeilen angibt . Ist die Anzahl der gelöschten Zeilen größer 0, wurde tatsächlich ein TEI-Dokument gelöscht und die gespeicherten Informationen müssen aktualisiert werden, wozu `_teichi_update_node_infos` aufgerufen wird. Die Seiteninformationen sollten dann wieder auf dem aktuellen Stand sein und können in der Seitensuch- und der Indexfunktion verwendet werden können.

Die Funktionalität hierfür steckt in der externen JavaScript-Datei „drp_main.js“ im „js“ Unterverzeichnis des Moduls. Durch Änderungen in der Drupal JavaScript-API ist es zwingend, dass der enthaltene Code in

```
(function ($) {
  // Original JavaScript code.
})(jQuery);
```

„verpackt“ wird [20]. Somit befindet er sich im Namensraum, der von Drupal verwendeten JavaScript-Bibliothek jQuery. jQuery entstand 2006 und wurde von John Resig entwickelt. Es ermöglicht den Eingriff in das Document Object Model (DOM) von HTML-Dokumenten mit Hilfe von JavaScript und auch dessen Manipulation. Dabei stellt es Funktionen zur Verfügung, die im Browser vielleicht (noch) nicht implementiert sind, wie zum Beispiel die Funktion `getElementsByClassName` aus der Web Applications 1.0 (HTML 5) Spezifikation [23], die es erlaubt alle Elemente einer Klasse zu selektieren und erleichtert so das browserübergreifende Entwickeln von JavaScript-Code. Es verfügt zudem über schnelle Selector-Funktionen, die weitestgehend denen im fertigen CSS3-Entwurf gleichen. Sie stammen aus der Sizzle-Engine [33], eine Entwicklung in Zusammenarbeit mit anderen JavaScript-Bibliotheken wie Dojo [34] und Prototype [35]. jQuery bietet auch Funktionen, wie vereinfachtes Iterieren durch Listen mit „each“, welches sich an moderne (Skript-) Sprachen anlehnt oder auch die Animation und optische Transformation von dargestellten Objekten, die mit dem CSS3-Entwurf als Transitions eingeführt werden [24]. Diese werden derzeit von nur wenigen aktuellen Browsern und auch nur unvollständig unterstützt. Weitere Vorteile von jQuery sind, die geringe Größe der Bibliothek, die quelloffene Lizenz und eine weite Verbreitung und somit auch eine große Entwicklergemeinde.

Eine weitere Anpassung war für das Abfangen von Events, also Ereignissen wie das

Klicken mit der Maus im Dokument oder das Aktivieren eines Links, notwendig. Das bedeutet zum Beispiel, dass die Funktion zum Abfangen von Klick-Ereignissen, welche schon die jQuery-Syntax benutzte, nochmals angepasst werden musste. In der früheren Projektversion sah die Funktion wie folgt aus:

```
$(document).ready(function(){
   $("a").click(function(event){
     if(this.href.indexOf('#p') >= 0)
     {
         event.preventDefault();
                  dispatchPageLink(this.href.substr( (this.href
.indexOf('#p')+2)) );
     }
   });
 });
```

durch die Anpassungen sah die Implementierung folgendermaßen aus:

```
Drupal.behaviors.teichi = {
        attach:function (context, settings){
        $('a',context).click(function(){

              if(this.href.indexOf('#p') >= 0)
              {
                  event.preventDefault();
                  Drupal.teichidispatchPageLink(this.href.subs
tr( (this.href.indexOf('#p')+2)) );
              }
            })
    }

 };
```

Die Änderungen waren notwendig um der Konvention zu entsprechen und um auch hier Namensraumkollisionen zu vermeiden.

An der grundlegenden Funktionalität ändert sich nichts. Es werden Klicks auf Verweise im Dokument abgefangen und untersucht ob sie die spezielle Form, eines Seitenlinks des Moduls, aufweisen. Ist im Link eine Seiteninformation enthalten, wird die entsprechende Suchfunktion aufgerufen, die die Node-Nummer zur Seite sucht. Die hierfür zuständige Funktion hieß in der Projektversion `dispatchPageLink` und heißt jetzt `teichidispatchPageLink`:

```
Drupal.teichidispatchPageLink = function (page)
{
  if(isNaN(page))
  {
    alert(tNotANumber +": ("+ page +")");
    return;
  }else{
```

```
    for(i=0; i < chapters.length; i++)
    {
       if(page >= chapters[i].start && page <= chapters[i].end
&& chapters[i].book == currentBook)
      {
          window.location.href = window.location.pathname + "?
q=node/"+chapters[i].node+"#p"+page;
        return;
      }
    }
    alert(tPageNotAssigned+" ("+page+")");
  }
}
```

Sie wurde um die Information des Buchs erweitert. Es wird also auch geprüft, ob sich die gesuchte Seite im selben Buch, wie die aufrufende Seite befindet. Wobei die Variable `currentBook` gleich 0 ist, wenn kein Buch definiert ist. Sollte eine Seite nicht gefunden werden, wird jetzt eine über das Modul-Menü konfigurierbare Meldung ausgegeben. Das ist zum Beispiel der Fall, wenn eine Seite nicht definiert wurde, beziehungsweise noch nicht hochgeladen wurde oder wenn der übergebene Wert keine Zahl war. Das kann vorkommen, wenn die Seite per Eingabefeld in der Menüleiste des Dokuments gesucht wurde. Die Menüleiste, die bei jedem TEI-Dokument am unteren Rand des Bildausschnitts eingeblendet wird, ist nun nicht mehr statisch im XSL-Stylesheet definiert, sondern wird per JavaScript beim Seitenaufruf erstellt. Hierfür wird die Funktion `createTeichiMenuBar` aufgerufen, welche die in der Menüleiste enthaltenen Elemente erzeugt, zusammenfügt und dann direkt an das Body-Element des HTML-Dokuments anhängt. Das verhindert Darstellungsprobleme mit der Menüleiste, die dadurch zu Stande kamen, dass die Leiste vorher, im Code, in einem DIV-Element abgelegt war. Zusätzlich wurde die Position per CSS als „fixed“ definiert, damit die Leiste beim hoch und runter Bewegen der Seite ihre Position beibehält, als wäre sie angeheftet. Zudem ist es im Konfigurationsmenü des Moduls möglich, eine prozentuale Breite einzustellen, wie zum Beispiel „80%“. Dabei ist dann nicht mehr gewährleistet, dass alle Browser auf gleiche Weise interpretieren von welchem übergeordneten Element die 80% berechnet werden. So erschien es logisch, die Menüleiste direkt an das Wurzelelement im Dokument anzuhängen, da so Artefakte beim Navigieren vermieden werden, die in als „fixed“ definierten „inline“-Objekten entstehen können und die Breite so eindeutig bestimmt werden kann, da sie sich immer auf das ganze Dokument bezieht. Das Menü selbst bietet 3 mögliche Aktionen. Der enthaltene Seitenindex wird jetzt durch die folgende Funktion (de-)aktiviert:

```
Drupal.teichiTogglePageIndex = function()
{

        var pageindex = document.getElementById('pageindex');
        if(pageindex != null)
        {
                if(pageindex.style.display == 'none' ||
pageindex.style.display == '')
                {
                        pageindex.style.display = 'inline';
```

```
                        pageindex.style.position = 'fixed';
                        pageindex.style.left = $
("a#teichiPageIndex").position().left;
                }else{
                        pageindex.style.display = 'none';
                }

        }
}
```

Sie blendet einen Seitenindex, der alle Seiten des aktuellen Kapitels, als anklickbare Links auflistet ein oder, falls schon angezeigt, wieder aus. Sie wurde um die Möglichkeit erweitert die Position des Verweises zu ermitteln, der den Index zur Anzeige bringt. Somit ist gewährleistet, dass die Seitenauflistung unmittelbar über dem dafür vorgesehenen Link erscheint, auch wenn es durch verschiedene Breiten zu Verschiebungen kommen kann.

Die Schalter zum Umschalten von Textversionen in der Mitte der Leiste, rufen die Funktion `Drupal.toggleView` auf. Diese ermöglicht das Ausblenden bestimmter Tags, die mit Hilfe von CSS-Klassen gruppiert wurden und das Einblenden anderer Klassen, die eine andere Textversion darstellen. Im Gegensatz zur alten Implementierung im Projekt, benötigt diese Funktion keine Hilfsfunktion zum Selektieren gesamter Klassen in einem Document-Object-Model-Baum (DOM Tree). Das geschieht jetzt mit einem jQuery-Selector. Auch das Ein- und Ausblenden ganzer Klassen geschieht über jQuery Funktionen, somit wird der Code erheblich reduziert und sieht wie folgt aus:

```
Drupal.toggleView = function (how)
{

        var toggle1 = document.getElementById('toggle1');
        var toggle2 = document.getElementById('toggle2');
        if(toggle1 && toggle2 && (how == 'orig' || how ==
'sic'))
        {
            $('.orig').show();
            $('.sic').show();
            $('.reg').hide();
            $('.corr').hide();
            toggle1.innerHTML = " <span
id='activetextmode'>"+tRegText+"</span>";
            toggle2.innerHTML = " <span
id='passivetextmode'>"+tOrigText+"</span>";
        }else{
            $('.orig').hide();
            $('.sic').hide();
            $('.reg').show();
            $('.corr').show();
            toggle1.innerHTML = " <span
id='passivetextmode'>"+tRegText+"</span>";
            toggle2.innerHTML = " <span
id='activetextmode'>"+tOrigText+"</span>";
```

```
        }

}
```

Durch die jQuery-Funktionen entfallen auch Iterationen über Listen von Klassen-Elementen beziehungsweise werden durch optimierte Funktionen ersetzt, um die Sichtbarkeit umzuschalten, was nicht nur Code-Zeilen spart, sondern auch schneller ist. Es wird jeweils eine Textversion ein- und die andere ausgeblendet, zusätzlich werden noch die Links zum Umschalten und deren Inhalt angepasst.

Die Funktion `Drupal.teichidispatchPageLink`, die sich um Eingaben in der Seitensuchbox kümmert, ist weitestgehend gleich geblieben, nur der Name wurde angepasst, damit sie sich vom JavaScript, in dem aus dem XSL-Stylesheet erzeugtem HTML-Code, ansprechbar ist. Was aber im Laufe der Implementierung aufgefallen ist, war die Tatsache, dass das Navigieren zu Seiten per Suchbox nicht in jedem Browser funktionierte. Es stellte sich heraus, dass die betroffenen Browser auf Google Chrome [36], Apple Safari [37] und Internet Explorer 8 [38]eingegrenzt werden konnten. Eine nähere Untersuchung mit JavaScript-Debuggern wie „Opera Dragonfly“ [39] und den, im Chrome-Browser eingebauten, „Entwicklertools“ ergab, dass der Fehler in der Funktion `createTeichiMenuBar` auftrat. Die Betrachtung des DOM-Baumes in den jeweiligen Werkzeugen zeigte auf, dass das Attribut „onsubmit“, welches für das Abschicken des Formulars mit der Seitenzahl zuständig war, nicht gesetzt wurde. Dadurch wusste der Browser nicht, wie er die übergebenen Formulardaten interpretieren soll und versuchte eine Übergabe der Parameter, durch das Anhängen an die URL des aktuellen Dokuments, also einen sogenannten „GET-Request“. Nach mehreren Testläufen konnte das Problem auf die jQuery-Funktion `attr` eingegrenzt werden. Die erste Vermutung, es könnte an auf der „WebKit“-Engine basierenden Browsern, wie Chrome und Safari liegen, bestätigte sich nicht, da der Fehler auch im Internet Explorer auftrat. Es muss sich also um einen Fehler in der `attr`-Funktion der jQuery-Bibliothek handeln. Das Problem wurde schließlich umgangen, indem das Formular zunächst als normaler Element-Knoten erzeugt wurde und dann darauf die Standard-JavaScript-Methode `setAttribute` angewandt wurde. Die betroffene Stelle im Quellcode stellte sich wie folgt dar:

```
var teichiPagesearch = $( document.createElement('form') );
teichiPagesearch.attr('onsubmit','javascript:Drupal.teichidisp
atchPageLink(this.pnr.value);return false;');
```

Nach der Anpassung sah sie folgendermaßen aus:

```
var tFormCon = document.createElement('form');
tFormCon.setAttribute('onsubmit','javascript:Drupal.teichidisp
atchPageLink(this.pnr.value);return false;');
var teichiPagesearch = $( tFormCon );
```

Hiernach wurde der neue Knoten, der das Formular darstellte, in ein jQuery-Objekt verpackt, so dass man fortfahren konnte wie gehabt, ohne weitere Änderungen am Code vornehmen zu müssen. Durch diese Änderungen, funktionierte die Seitensuche in allen Browsern wie erwartet.

Weitere JavaScript-Teile sind direkt im XSL-Stylesheet enthalten. So wird zum

Beispiel der Seitenindex pro Kapitel, noch immer per XSL-Transformation erzeugt. Dieser Index enthält auch einen Link zum Schließen, der die Toggle-Funktion zum Ausblenden benutzt. Eine zusätzliche Erweiterung die das Modul erfahren hat ist, dass Anmerkungen, die an der Seite zugeklappt eingeblendet werden, sich jetzt auch durch direktes Anklicken öffnen lassen, statt nur durch spezielle Marker im Text. Das passiert durch Aufruf der Funktion `Drupal.teichiShowNote2`, welcher die ID einer Notiz als Parameter übergeben wird. Die Implementierung wurde um kleine jQuery-Anpassungen erweitert:

```
Drupal.teichiShowNote2 = function (nr)
{
    var note = document.getElementById('note'+nr);
    if(note)
    {
        if(note.style.display == "block")
        {
            Drupal.teichiHideNote(nr);
            return;
        }

        note.style.display = "block";

        if(note.parentNode)
        {
            $('.note').css('z-index','0');
            note.parentNode.style.zIndex = "10";
            note.parentNode.style.overflow = "auto";
        }
    }

}
```

Es wird versucht die Anmerkung mit der übergebenen ID in der Variable `note` abzulegen. Dann wird zunächst geprüft, ob eine solche existiert und wenn ja, ob sie nicht schon angezeigt wird. Wird sie schon angezeigt, wird das Ausblenden veranlasst und die Funktion wieder verlassen, ansonsten wird die Notiz sichtbar gemacht und deren Eltern-Element in den den Vordergrund geholt über die CSS-Eigenschaft z-Index, wobei andere Elemente mit der Klasse „note“, in den Hintergrund verschoben werden. Das Verhindert eine Überlappung bei Notizen die eng beieinander liegen oder durch deren Höhe beim Aufklappen in andere Notizen hineinreichen. Hinzugekommen ist auch ein Befehl, der die Overflow-Eigenschaft der Notizen regelt. Diese ist zuständig für Elementbereiche mit übergroßem Inhalt. Bei Browsern wie Chrome oder Safari, wurden bei zugeklappten Notizen unnötigerweise kaum sichtbare Scrollbalken eingeblendet. Das wird nun verhindert, indem sie zunächst explizit ausgestellt sind und erst beim Aufklappen und auch nur bei Bedarf eingeschaltet werden.

Über die simple Funktion `teichiHideNote` werden Notizen wieder ausgeblendet.

```
Drupal.teichiHideNote = function (nr)
{
    var note = document.getElementById('note'+nr);
```

```
    if(note)
    {
        note.style.display = "none";
        if(note.parentNode)
        {
            note.parentNode.style.zIndex = "0";
            note.parentNode.style.overflow = "hidden";
        }
    }
}
```

Auch hier wird zunächst die Notiz mit der entsprechenden Nummer gesucht und bei Erfolg deren Sichtbarkeit wieder ausgestellt und der übergeordnete Container in den Hintergrund geschickt. Die Overflow-Eigenschaft wird wieder entsprechend angepasst, indem zu große Elemente ausgeblendet werden um Darstellungsfehler zu verhindern.

5 Zusammenfassung

5.1 Erreichte Ziele

Zusammenfassend kann man sagen, dass die gesetzten Ziele erreicht wurden. Das Modul bietet die Funktionalität, des vorangegangenen Projekts, wobei die Vorgänge und Arbeitsschritte vereinfacht und weitestgehend automatisiert wurden. Es gibt eine komfortable Konfigurationsoberfläche, die die Einstellungsmöglichkeiten an einer zentralen Stelle zugänglich macht und keine Eingriffe ins Drupal- oder das Dateisystem erfordert. Der Unterbau wurde optimiert und einige Vorgänge, wie das Umschalten von Textvarianten, effizienter umgesetzt. Erweitert wurde auch die Seitensuch- und Indexfunktion des Moduls, um die Unterstützung mehrerer Bücher, die mit Hilfe des Book-Moduls angelegt werden können. Die Funktionalität des XML-Content-Moduls wurde auf das nötigste reduziert und in das TEIChi-Modul integriert. Somit entfällt die Abhängigkeit, sowohl zum Modul, als auch zur Fortführung der Pflege und Anpassung des XML-Content-Moduls an neue Drupal-Versionen. Die Darstellung und Handhabung wurden an einigen Stellen verbessert. So lassen sich Notizen am Rand jetzt auch durch einen Klick auf die Notizen selbst öffnen, anstatt den zugehörigen Marker im Text suchen zu müssen. Die Menüleiste, bei Darstellung der Visualisierungen am unteren Rand, wurde vom im XSL-Stylesheet erzeugten Code entkoppelt und wird jetzt dynamisch beim Seitenaufruf erzeugt, was sich positiv auf die Darstellung und die Konfigurationsmöglichkeiten, zum Beispiel die einstellbare Breite, auswirkt und auch deren Zentrierung, die vorher mit einem Seitenabstand gelöst wurde, wurde mit der entsprechenden CSS-Eigenschaft und einem zusätzlichen, übergeordnetem Elternelement umgesetzt, das sicherstellt, dass die Leiste bei verschiedenen Monitorauflösungen mittig erscheint.

5.2 Resümee und Ausblick

An einigen Stellen der Implementierungsphase, zeigten sich Möglichkeiten auf, wie man einige Entwürfe hätte anders umsetzen können. So ist die Idee mit den Schriftarten relativ spät ins Konzept aufgenommen worden. Ein Hochladen der Schriftarten, schien zunächst als der einfachste und sicherste Weg diese einzubinden. Die Recherche ergab, dass es zusätzlich noch die Möglichkeit gibt, externe Schriftarten von entfernten Rechnern nachzuladen. Diese Methode, konnte aus zeitlichen Gründen nicht mehr umgesetzt werden, weil nicht nur der Vorhandene Ansatz, hätte umstrukturiert werden müssen, sondern zum Beispiel auch zusätzliche Prüfungen der Quelldateien notwendig geworden wären. Auch hätte man noch Fallback-Reihenfolgen für Schriftarten anbieten können, also das Definieren von Ausweichschriftarten, falls eine angegebene Schriftartdatei nicht gefunden wird. Dieser Ansatz hätte aber aufgrund der vielen Kombinationsmöglichkeiten schnell zum unnötigen Verkomplizieren dieser Option führen können. Zu den Überlegungen kommt hinzu, dass das Einbinden von Schriftarten, noch kein bindender und vor allem kein fertiger Standard ist. Besonders was die unterstützten Formate angeht, hat sich in letzter Zeit viel getan. So haben sich einige Browserhersteller auf das Woff-Format geeinigt, das in den neuesten Versionen

von Browsern wie Mozilla Firefox oder Google Chrome Einzug gehalten hat. Es ist auch absehbar, das sich neue Dienste in dieser Richtung etablieren werden, wie zum Beispiel The Google Font Directory [42] oder Linotype Webfonts [43], um nur zwei Beispiele zu nennen. Es bleibt also abzuwarten, wie sich die Verwendung von Webfonts in naher Zukunft entwickelt, wobei die im Modul implementierte Version auf absehbare Zeit, einen guten Kompromiss darstellt.

Die Verwendung von jQuery, als JavaScript-Bibliothek, hat sich als hilfreich erwiesen, da sie dem Entwickler Arbeit, in Bezug auf Browserunabhängigkeit abnimmt, indem sie Funktionen ergänzt oder einfach effizienter umsetzt. Wobei bei der Implementierung auch Probleme aufgezeigt wurden, wie zum Beispiel das Setzen einiger Attribute in bestimmten Browsern. Der Vorteil hierbei ist aber, dass das jQuery-Paket in der Drupal-Installation mitgeliefert wird und somit die Aktualisierung und Kompatibilität weitestgehend gesichert sind.

Was die Wahl von Drupal 7 als Zielversion angeht, kann man sagen, dass bei Fertigstellung dieses Dokuments noch immer keine stabile Version 7 zur Verfügung stand. Wie so oft bei Software, hat die Entwicklung länger in Anspruch genommen als ursprünglich erwartet. So befand sich Drupal 7 bei der Implementierung immer im Alpha-Stadium (Alpha 6 und Alpha 7). Das wirkte sich nicht nur negativ auf manche Code-Teile aus, die noch nicht richtig implementiert waren, sondern auch auf Schnittstellenänderungen und vor allem die kaum vorhandene Dokumentation der neuen Funktionen. Manche Abläufe und Techniken mussten erst mühsam recherchiert werden, so mussten zum Beispiel Informationen zur Handhabung von „managed-file-uploads“, also dem asynchronen JavaScript-Upload von Dateien, aus Entwicklerkommentaren zu Quellcodeänderungen zusammengetragen werden. Auch konnte durch den Versionssprung, nicht immer das im Projekt mit Drupal 6 erlangte Wissen angewandt werden.

Bei der Implementierung kamen auch neue Ideen und Verbesserungen auf, die aus Zeitgründen nicht mehr umgesetzt werden konnten. Zum einen waren das natürlich mehr Einstellungsoptionen, um die Darstellung noch flexibler zu gestalten oder auch die Möglichkeit die Menüleiste auszublenden zum Beispiel bei TEI-Dokumenten, die keine Seitenzahlen enthalten. Eventuell auch automatisch, falls keine Seitenzahlen erkannt werden oder womöglich als Option im Konfigurationsmenü. Auch das mit dem Modul mitgelieferte XSL-Stylesheet ist ausbaufähig, so könnte man es um weitere Elemente aus dem TEI-Lite-Standard erweitern. Da dieses Modul, wie XML-Content, unter der GNU Public License zur Verfügung gestellt wird, kann es als Grundstein dienen und in Zukunft nach Belieben angepasst und erweitert werden.

Literaturverzeichnis

[1] Jacob Redding. *Beginning Drupal*, John Wiley & Sons, 2010

[2] Cody Lindley. *jQuery Cookbook: Solutions & Examples for jQuery Developers*, O'Reilly Media; 1 edition (November 19, 2009) S.2

Webseiten (Letzter Zugriff am 5.10.2010)

[3] Wikipedia. Text Encoding Initiative
http://de.wikipedia.org/wiki/Text_Encoding_Initiative

[4] Drupal. About Drupal
http://drupal.org/about

[5] Bérardier de Bataut, Essai sur le récit
http://www.berardier.org

[6] Wikipedia. Drupal
http://de.wikipedia.org/wiki/Drupal

[7] Golem. Britische Regierung veröffentlicht quelloffenes Modul
http://www.golem.de/1008/77507.html

[8] Owen Barton. Drupal 7 - faster than ever
http://googlecode.blogspot.com/2010/09/drupal-7-faster-than-ever.html

[9] Wikipedia. WOFF
http://en.wikipedia.org/wiki/WOFF

[10] Drupal. Converting 6.x modules to 7.x
http://drupal.org/update/modules/6/7#remove_op

[11] Drupal. Writing .info files (Drupal 7.x)
http://drupal.org/node/542202

[12] Drupal. XML Content
http://drupal.org/project/xmlcontent

[13] John Resig. The jQuery Project,
http://jquery.org/about

[14] Wikipedia. jquery
http://de.wikipedia.org/wiki/JQuery

[15] W3C. CSS Fonts Module Level 3. Working Draft 18 June 2009 The (@font-face rule)
http://www.w3.org/TR/css3-fonts/#the-font-face-rule

[16] Code Style. Windows font survey results
http://www.codestyle.org/css/font-family/sampler-WindowsResults.shtml

[17] Drupal. Converting 6.x modules to 7.xm hook_filter() and hook_filter_tips() replaced by hook_filter_info()
http://drupal.org/update/modules/6/7#hook_filter_info

[18] PHP. Strings, Heredoc

http://de.php.net/manual/en/language.types.string.php#language.types.string.syntax.heredoc

[19] Drupal. „hook_help“
http://api.drupal.org/api/function/hook_help/7

[20] Drupal. JavaScript should be compatible with other libraries than jQuery
http://drupal.org/update/modules/6/7#javascript_compatibility

[21] Drupal. „hook_menu“
http://api.drupal.org/api/function/hook_menu/7

[22] Drupal. New #managed_file element is undocumented
http://drupal.org/node/750190

[23] WHATWG. HTML5 (including next generation additions still in development)
http://www.whatwg.org/specs/web-apps/current-work/#getelementsbyclassname

[24] W3C. CSS Transitions Module Level 3
http://www.w3.org/TR/css3-transitions/

[25] Webfonts.info. @font-face browser support
http://www.webfonts.info/wiki/index.php?title=%40font-face_browser_support

[26] W3C. xml:id Version 1.0 (6. Errors)
http://www.w3.org/TR/xml-id/#errors

[27] W3C. CSS Color Module Level 3 (4.1 HTML4 color keywords)
http://www.w3.org/TR/css3-color/#html4

[28] Wikipedia. Content-Management-System
http://de.wikipedia.org/wiki/Content-Management-System

[29] Text Encoding Initiative
http://www.tei-c.org/

[30] The White House
http://www.whitehouse.gov

[31] Sic
http://de.wikipedia.org/wiki/Sic

[32] API-Dokumentation Drupal7
http://api.drupal.org/api/7

[33] Sizzle-Engine
http://blog.jquery.com/2009/01/14/jquery-13-and-the-jquery-foundation/

[34] Dojo
http://dojofoundation.org/

[35] Prototype
http://prototypejs.org/

[36] Google Chrome
http://www.google.com/chrome

[37] Apple Safari
http://www.apple.com/safari/

[38] Internet Explorer
http://www.microsoft.com/ie

[39] Opera Dragonfly
http://www.opera.com/dragonfly/

[40] Unterstützung von Schriftartformaten
http://www.webfonts.info/wiki/index.php?title=%40font-face_browser_support

[41] CSS3 Values and Units
http://www.w3.org/TR/css3-values/

[42] Google Webfonts
http://code.google.com/webfonts

[43] Heise.de: Linotype
http://www.heise.de/newsticker/meldung/Linotype-mischt-bei-den-Webfonts-mit-1078988.html

[44] Field-API
http://drupal.org/node/443540

[45] Datenbank-API
http://drupal.org/node/310069

Bilder

[46] Schema: Transformation. Erstellt von Prof. Dr. Lutz Wegner