XSL Transformationen als Vorbereitung für die Bibliothekssuchmaschine ALBERT

Bachelorarbeit

Linda Nolte

zur Erlangung des akademischen Grades ***Bachelor of Arts***
im Studiengang ***Bibliotheksmanagement***
an der Fachhochschule Potsdam
Fachbereich Informationswissenschaften

Erster Gutachter: Prof. Dr. Günther Neher
Zweiter Gutachter: Dipl.-Dok. Tobias Höhnow

Potsdam, Januar 2012

Abstract

Die vorliegende Arbeit beschreibt eine Möglichkeit zur Anpassung von Daten, in eine Struktur im XML-Format, die die Bibliothekssuchmaschine ALBERT zur korrekten Darstellung benötigt. Diese Daten bestehen aus Angaben zu Zeitschriftenbeständen aus der Elektronischen Zeitschriftenbibliothek und zu neuen Zeitschriftenartikeln, die mit dem Dienst JournalTOCs abonniert werden.

Zunächst soll dafür erläutert werden, was eine Bibliotheksuchmaschine ist. Hierzu werden auch herkömmliche Katalogoberflächen und Gründe für eine Veränderung zu den Suchmaschinen beschrieben.
Dann werden die Werkzeuge, mit denen die Anpassung ermöglicht wird, nämlich Xpath, XSLT und Apache Ant, kurz charakterisiert. Aufbauend darauf werden im letzten Teil der Arbeit der Prozess und seine Funktionsweise dargestellt.

Anmerkung

Auf die ausdrückliche sprachliche Differenzierung der maskulinen und femininen Schreibweise wird zu Gunsten eines besseren Leseflusses verzichtet. Die maskuline Form bezieht sich, wenn nicht anders gekennzeichnet, immer auf männliche und weibliche Personen.

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

ALBERT	All Library Books, journals and Electronic Resources Telegrafenberg
CSV	Comma-Separated Values
DTD	Document Type Definition
E-ISSN	Electronic International Standard Serial Number
EZB	Elektronische Zeitschriftenbibliothek
HTML	Hypertext Markup Language
ISSN	International Standard Serial Number
KOBV	Kooperativer Bibliotheksverbund Berlin-Brandenburg
MAB	Maschinelles Austauschformat für Bibliotheken
OPML	Outline Processor Markup Language
RSS	Really Simple Syndication
W3C	World Wide Web Consortium
XHTML	Extensible Hypertext Markup Language
XML	Extensible Markup Language
XPath	XML Path Language
XSL	Extensible Stylesheet Language
XSL FO	Extensible Stylesheet Language Formatting Objects
XSLT	Extensible Stylesheet Language Transformation

1. Einleitung

Der Bibliothekskatalog ist ein Instrument, mit dem nachgewiesen werden soll, welches Wissen in einer Bibliothek vorhanden ist.[1]
Weil seine Produktion und Pflege einen elementaren Bestandteil der Geschichte, Identität und Arbeit in Bibliotheken darstellen, ist er auch immer ein wichtiger Faktor bei Diskussionen in der Bibliothekswelt.[2]
Das heißt immer dann, wenn in Bibliotheken etwas geändert wird, d.h. sie sich zum Beispiel mehr an den Bedürfnissen und Vorlieben ihrer Zielgruppen orientieren, wird auch der Katalog, in diesem Fall nutzerfreundlicher, umgestaltet.[3] Insgesamt werden Bibliothekskataloge ständig überarbeitet und mit neuen Funktionen ausgestattet, was beispielsweise die angebotenen Dienstleistungen und Suchoberflächen betrifft.[4]

Eine aktuelle Suchoberfläche sind Bibliothekssuchmaschinen, die vor allem durch das Aufkommen von Open-Source-Produkten als Konkurrenz zu den schon länger bestehenden kommerziellen Produkten in die Aufmerksamkeit von Bibliotheken gelangt sind.[5]
Diese Suchoberflächen sind auch deshalb interessant, weil sie einen gemeinsamen Zugang zu verschiedenen Informationen aus unterschiedlichen Quellen bieten und dadurch attraktiver für den Suchenden werden, da er dann nicht mehr an verschiedenen Stellen nach Informationen suchen muss. Denn schließlich bietet eine Bibliothek zwar Informationen, die nicht im Katalog nachgewiesen sind, d.h. vom Suchenden nicht über den herkömmlichen Katalog gefunden werden können, aber gleichzeitig auch nicht über das Internet frei verfügbar und nutzbar sind, wie zum Beispiel Artikel von lizenzpflichtigen Zeitschriften.[6]
Diesen Ansatz, unter einer Oberfläche verschiedene Informationsarten anzubieten, gab es bereits um 1990. Damals war die Umsetzung technisch zwar möglich, aber sie

[1] Vgl. Calhoun (2006), S. 7.

[2] Vgl. Dempsey (2006).

[3] Vgl. Dempsey (2006).

[4] Vgl. Lauber-Reymann (2010), S. 164.

[5] Vgl. Breeding (2009), S. 40.

[6] Vgl. Coyle (2007a), S. 289.

war nicht gut zu handhaben, da auch viel teurer Speicherplatz und Verhandlungen mit den Verlagen um Informationen nötig waren,[7] die heute teilweise frei zugänglich gemacht werden.

Ziele waren damals, die lokal vorhandenen Sammlungen zu vereinigen, Zugang zu relevanten, nicht lokal vorhandenen Ressourcen, zu Volltexten und zu Definitionen aus Nachschlagewerken zu bieten,[8] sowie den Katalog offener für die Nutzung der Daten durch andere Systeme zu machen.[9] Der Zugang zu Artikeln aus Nachschlagewerken sollte, so wie es heute in einigen Suchoberflächen Wikipedia-Artikel tun, dabei helfen, das Thema der eigenen Suche besser zu verstehen, indem Suchbegriffe erklärt werden.

Insgesamt sollten damals also schon so viele Informationen wie möglich unter einer gemeinsamen Oberfläche angeboten werden.[10]

Die Beschreibung, welche Funktionen so eine Bibliothekssuchmaschine außer dem gemeinsamen Zugriff auf verschiedene Quellen bieten kann, soll im ersten Teil der vorliegenden Arbeit erfolgen. Da die Beschreibung der Bibliothekssuchmaschinen auch in Abgrenzung zu den herkömmlichen Suchoberflächen von Bibliothekskatalogen geschehen soll, sollen diese Suchoberflächen als Erstes charakterisiert werden, sowie Gründe für die Veränderung der Suchoberflächen genannt werden. Zu diesen Gründen zählen jedoch nicht nur Probleme der herkömmlichen Suchoberflächen.

Die Ursachen für die Veränderung bilden gleichzeitig auch einen Teil der Basis für die Beschreibung der neuen Suchoberflächen, da die Veränderung insbesondere aufgrund des veränderten Nutzersuchverhaltens und der modifizierten Ansprüche in Bezug auf die Suchoberflächen stattfindet.

Da mit der neuen Suchoberfläche Informationen aus verschiedenen Quellen gemeinsam zugänglich gemacht werden sollen, müssen diese Informationen alle in

---

[7] Vgl. Potter (1989), S. 104.

[8] Vgl. Potter (1989), S. 100.

[9] Vgl. Potter (1989), S. 104.

[10] Vgl. Potter (1989), S. 100.

einer bestimmten gemeinsamen Struktur vorliegen, damit sie von der Suchoberfläche ausgewertet werden können.[11]
Um die unterschiedlich strukturierten Daten also für die Suchoberfläche nutzbar zu machen, müssen manche von ihnen in das benötigte Format gebracht werden.

Da dies auch bei der Bibliothekssuchmaschine ALBERT so ist, die aus einer Kooperation des KOBV (Kooperativer Bibliotheksverbund Berlin-Brandenburg) und der Bibliothek des Wissenschaftsparks Albert Einstein in Potsdam hervorgegangen ist,[12] wurde im Rahmen der vorliegenden Arbeit versucht eine Möglichkeit zur Lösung des Problems für diese Suchmaschine zu finden.
Dieser Versuch wurde exemplarisch für die elektronischen Zeitschriftenbestände der Bibliothek, die unter anderem in der Elektronischen Zeitschriftenbibliothek (EZB) verwaltet werden und für RSS-Feeds (Dateien im Format Really Simple Syndication), die die neusten Artikel aus Zeitschriften enthalten und über einen Dienst namens JournalTOCs[13] abonniert werden können, gestaltet.
Da für einen Datenaustausch häufig das Format XML (Extensible Markup Language) verwendet wird[14] und auch die Daten aus den beiden Verwaltungssystemen weitestgehend in diesem Format exportiert werden können, wurde auf Basis dieses Formats ein Lösungsweg gesucht.
Außerdem können die Anweisungen zum Bearbeiten von XML-Dokumenten relativ einfach angepasst bzw. verändert werden, insbesondere, weil keine spezielle Software für das Bearbeiten der Anweisungen nötig ist, sondern ein einfacher Texteditor dafür verwendet werden kann.[15]
Zusätzlich eignet sich die Transformation von XML-Dokumenten besonders gut für Anwendungsfälle, bei denen nicht alle ursprünglichen Daten verwendet werden sollen[16] oder wenn die ursprünglichen Daten mit weiteren Informationen

---

[11] Vgl. Tennant (2003.

[12] Vgl. Bertelmann; et al. (2007), S. 1302.

[13] http://www.journaltocs.hw.ac.uk/, [letzter Zugriff: 12.01.2012].

[14] Vgl. Yee; Beaubien (2004), S. 69.

[15] Vgl. Keith (2004), S. 125.

[16] Vgl. Jackenkroll (2003), S. 33.

angereichert werden sollen. Letzteres gilt auch für die neusten Zeitschriftenartikel, die zusätzliche Informationen von den Zeitschriften aus der EZB erhalten sollen.

Damit nun der Prozess zur Anpassung der Daten aus den beiden Verwaltungssystemen erläutert werden kann, sollen zunächst die Mittel, mit denen der Prozess durchgeführt werden kann, im zweiten Teil der Arbeit beschrieben werden.
Der Prozess, mit dem die Anpassung der Daten schließlich vorgenommen werden kann, soll im letzten Teil der Arbeit dargestellt werden.
Ziel ist es also grundsätzlich, eine Möglichkeit zur Anpassung von Daten in ein bestimmtes Schema zu erläutern, wobei zunächst die Hintergründe für die Anpassung, d.h. die Ursachen und letztlich das System, wegen derer die Anpassung der Daten nötig ist, beschrieben werden sollen.

2. Suchoberflächen

In diesem Kapitel sollen zunächst die herkömmlichen Suchoberflächen beschrieben werden. Dann folgt eine Darstellung möglicher Gründe für den Wechsel oder die Anpassung der Suchoberflächen und schließlich eine Beschreibung von Bibliotheks-suchmaschinen und ihren Funktionen, sowie von der Bibliothekssuchmaschine ALBERT, für die der Prozess, der Thema dieser Arbeit ist, erstellt wurde.

2.1. Ausgangslage und herkömmliche Suchoberflächen

Häufig besteht die Möglichkeit in einer Bibliothek etwas zu finden aus einer Anzahl von elektronischen Angeboten, in denen einzeln nach verschiedenen Informationen gesucht werden muss. Die Angebote sind meist nur auf den Bibliothekswebseiten verlinkt, aber nicht untereinander. Daher sind sie nicht richtig miteinander verknüpft. Beispiele für diese einzelnen Angebote sind die Kataloge, sowie Datenbanken mit fachwissenschaftlich relevanten Publikationen und Nachweisinstrumente für Zeitschriften, wie zum Beispiel die EZB. Also muss der Nutzer genau wissen, welche Art von Information er sucht, um zu wissen, mit welchen Instrumenten er für das Finden der Information arbeiten muss.[17]

Erschwerend kommt hinzu, dass die Angebote häufig unterschiedliche Suchoberflächen haben, sodass der Nutzer sich auch mit den Oberflächen auskennen muss, um die Instrumente effizient und effektiv nutzen zu können.[18] Sonst könnte er u.U. nicht alle relevanten und für ihn zugänglichen Informationen finden.

Neben der Tatsache, dass bestimmte Informationen nicht gefunden werden können, besteht weiterhin das Problem, dass bestimmte Angebote erst gar nicht genutzt werden, weil nicht mit ihnen umgegangen werden kann und manchmal sogar weil sie in der Menge von unterschiedlichen Angeboten nicht gefunden werden.[19]

Diese Probleme sind schon länger bekannt und man wünschte sich auch bereits um 1990, verschiedene Informationsarten und zusätzliche Informationen, wie den Volltext, in den Katalog zu integrieren, [20] um das Finden von Medien zu erleichtern.

---

[17] Vgl. Tröger (1999), S. 648.

[18] Vgl. Breeding (2010), S. 32.

[19] Vgl. Soules (2010), S. 10.

[20] Vgl. Potter (1989), S. 100.

Das wohl wichtigste Instrument, um in einer Bibliothek Informationen zu finden, ist der Katalog, der konzipiert wurde, um bei der Suche nach Informationen Zeit zu sparen und zu zeigen, welches Wissen in einer Bibliothek vorhanden ist.[21]

Die bedeutendsten Funktionen, die der Katalog bietet, sind die Möglichkeit Informationen zu verwalten und die Möglichkeit nach Informationen zu suchen.[22]

Die Informationen, die enthalten sind, sind mit - vor allem im Bereich der Sacherschließung hochwertigen[23] und mit viel Arbeit erstellten[24] - Metadaten erschlossen, die zum Teil auch normiert sind, wie zum Beispiel Personennamen oder Schlagwörter.[25] Dies kann bei der Suche insofern hilfreich sein, dass man nur einen normierten Suchbegriff nutzen kann, anstatt ebenfalls nach möglichen Synonymen suchen zu müssen, um Informationen zu einem Sachverhalt zu finden.
Ebenso hilfreich ist, dass in der Regel zusammengehörende Titelaufnahmen, beispielsweise die gedruckte und die elektronische Version eines Titels, miteinander nicht nur über die sachliche Erschließung verknüpft sind.[26]
Insgesamt kann über die Kataloge gesagt werden, dass sie in der Regel Teil eines Bibliothekssystems sind, welches viele Prozesse in der Bibliothek unterstützt, wie beispielsweise die Ausleihe oder Fernleihe.[27] Dadurch enthalten die Kataloge auch nur das, was in dem System verwaltet wird.[28] Dies ist normalerweise nur ein Teil des Bestandes über den eine Bibliothek verfügt.[29]
Allerdings muss festgestellt werden, dass es von Bibliothek zu Bibliothek variiert, was in ihrem Katalog verzeichnet wird, sodass manchmal zum Beispiel auch die

---

[21] Vgl. Calhoun (2006), S. 7.

[22] Vgl. Calhoun (2006), S. 31.

[23] Vgl. Blenkle; Ellis; Haake (2009), S. 618.

[24] Vgl. Kostädt (2008), S. 102.

[25] Vgl. Calhoun (2006), S. 31.

[26] Vgl. Tröger (1999), S. 650.

[27] Vgl. Kostädt (2008), S. 101.

[28] Vgl. Breeding (2010), S. 31.

[29] Vgl. Lewandowski (2006), S. 72.

Zeitschriften nicht mit dem Bibliothekssystem, sondern mit einem eigenen System, verwaltet werden.

Zum Teil enthalten Kataloge auch zusätzlich Informationen zu den Medien, die in der Bibliothek vorhanden sind, wie zum Beispiel Inhaltsverzeichnisse von Büchern. Diese zusätzlichen Informationen werden einerseits verfügbar gemacht, um bei einer Entscheidung über die Relevanz des Treffers in Bezug auf die Suchanfrage zu helfen. Andererseits werden sie zur Unterstützung der Suche genutzt, indem die Wörter aus den Dateien auch in den Suchraum integriert werden.[30]

Außerdem sind die Kataloge normalerweise mit anderen Anwendungen, wie zum Beispiel dem Zugriff auf das Benutzerkonto oder Dokumentenlieferfunktionen, verknüpft.[31]

Mit all den Möglichkeiten, die ein Katalog bietet, insbesondere durch die strukturierten Metadaten und die Funktionen der komplexen Suchmaske, bei der Suchanfragen bereits zu Beginn einer Suche stark präzisiert werden können, eignet er sich vor allem für eine Suche nach Standorten bzw. Zugängen zu bereits bekannten Informationen.[32] Probleme weist er daher eher bei thematischen Suchen auf, die in den Katalogen schon immer schwierig umzusetzen waren.[33]

Außerdem benötigt man, wie bereits angedeutet, eine gute Kenntnis der Möglichkeiten, die ein Katalog bietet, um mit ihm effizient und effektiv arbeiten zu können.[34] Beherrscht man jedoch diese Fähigkeiten, kann der Katalog zu einem sehr nützlichen Arbeitswerkzeug werden.

Zu den Dingen, die man beachten muss, wenn man mit einem Katalog arbeitet, und die auch kritisiert werden, gehört beispielsweise, dass die eingegebenen Suchbegriffe in der Regel durch ein logisches „UND“ verknüpft werden, sodass wenn einer der Suchbegriffe nicht gefunden wird, die ganze Suchanfrage keine Ergebnisse liefert.[35] Ähnlich verhält es sich, wenn ein Suchbegriff einen Rechtschreibfehler enthält, der meist nicht durch das System erkannt und korrigiert wird, sodass auch in diesem Fall

---

[30] Vgl. Kostädt (2008), S. 102-103.

[31] Vgl. Kostädt (2008), S. 105.

[32] Vgl. Lewandowski (2010), S. 91.

[33] Vgl. Larson (1991), S. 135.

[34] Vgl. Lewandowski (2010), S. 91.

[35] Vgl. Wiesenmüller (2008), S. 27.

keine Ergebnisse erzielt werden und die Suchanfrage erneut eingegeben werden muss.[36] Manchmal erkennt der Nutzer, wenn er keine Ergebnisse erzielt hat, auch nicht, dass die Suchanfrage einen Fehler enthält und betrachtet daher die Suchanfrage als Ganzes als nicht zielführend, weswegen er sie dann und verwirft.

2.2. Gründe für die Veränderung

Neben der Kritik an den bestehenden Suchoberflächen der Kataloge, die bereits im vorausgegangenen Teil der Arbeit angeklungen ist, sind die Änderungen des Verhaltens und der Bedürfnisse der Nutzer die wichtigsten Gründe, warum die bestehenden Katalogoberflächen momentan in manchen Bibliotheken verändert oder ersetzt werden.

Zunächst sollen jedoch weitere Kritikpunkte an den bestehenden Oberflächen genannt werden, um dann auf dieser Basis besser auf die modifizierten Nutzerbedürfnisse eingehen zu können.

Ein Ausgangspunkt des Problems mit den Suchoberflächen ist, dass immer mehr elektronische Ressourcen in die Bibliothek kommen und es schwierig ist einen Zugang zu ihnen zu schaffen, der attraktiv, angemessen und einfach ist.[37] Dies begründet sich darin, dass die Bibliothekssysteme zum Verwalten von Büchern und Zeitschriften geplant wurden und deshalb die elektronischen Informationen nur unzureichend darstellen bzw. verwalten können.[38]

Da die bestehenden Systeme nicht für die elektronischen Informationsressourcen konzipiert wurden, wurden unter anderem andere Lösungen gefunden, um den Ressourcen gerecht zu werden, wie zum Beispiel elektronische Bestände in eigenen Systemen zu verwalten.

In den 1990ern z.B. in Form der Digitalen Bibliothek NRW gab es schon Versuche die verschiedenen Systeme, die ein Nutzer zum Auffinden von Informationen verwenden kann, unter einer Oberfläche zu vereinigen. Hierbei sind sowohl die angesprochenen Angebote zum Finden von elektronischen Ressourcen wie Datenbanken, als auch Kataloge von verschiedenen Bibliotheken und

---

[36] Vgl. Wiesenmüller (2008), S. 28.

[37] Vgl. Breeding (2010), S. 32.

[38] Vgl. Casey (2007), S. 17.

Verbundkataloge gemeint.[39] Diese Versuche werden auch als Metasuchmaschinen bezeichnet.

Bei den Metasuchmaschinen merkt der Nutzer meist gar nicht, dass die erzielten Ergebnisse aus mehreren Quellen stammen.[40]

Als nachteilig wurde an solchen Systemen allerdings empfunden, dass die Suchanfrage in die Sprache der jeweiligen Informationsquelle umgesetzt werden muss, damit in ihr gesucht werden kann und dann die Treffer aus den einzelnen Quellen gesammelt werden, um schließlich unter einer gemeinsamen Oberfläche präsentiert zu werden. Diese Vorgehensweise bedeutet, dass die Auslieferungsdauer der Treffer sich nach der langsamsten Quelle richtet, d.h. die Suchgeschwindigkeit insgesamt unter Umständen als langsam empfunden wird, weil eine Quelle eine lange Reaktionszeit hat. [41]

Zusätzlich werden die Möglichkeiten eine Suchanfrage zu formulieren durch dieses Vorgehen eher begrenzt, da nur solche Möglichkeiten angeboten werden können, die in allen Quellen verwendet werden können.[42] Das bedeutet, dass nicht das volle Potential, das die Daten aus den unterschiedlichen Systemen haben, ausgeschöpft werden kann. Anders formuliert, fallen demnach vielleicht wichtige Felder für eine Quellenart bei der Suche weg, da nur Felder, die allen Quellen gemeinsam sind, durchsucht werden können.

Grundsätzlich enthält diese Art von Angeboten schon viele Charakteristika der Bibliothekssuchmaschinen, die im Verlauf der Arbeit noch beschrieben werden sollen, wie zum Beispiel, dass sie auf die persönlichen Bedürfnisse eingerichtet werden können,[43] oder dass Suchergebnisse inhaltlich oder formal gruppiert werden können.[44]

Jedoch werden diese Systeme kritisiert, wenn sie auch vom Katalog aus genutzt werden können, da in diesem Fall der Nutzer dadurch verwirrt werden könne, dass nicht klar angezeigt würde, aus welcher Quelle die jeweilige Information stamme

---

[39] Vgl. Tröger (1999), S. 649-650.

[40] Vgl. Kostädt (2008), S. 103.

[41] Vgl. Lossau (2004), S. 287.

[42] Vgl. Lossau (2004), S. 287.

[43] Vgl. Kostädt (2008), S. 104.

[44] Vgl. Lewandowski (2010), S. 90.

und der Nutzer daher auch nicht sicher sein könne, dass alle angezeigten Informationen auch verfügbar sind bzw. alle für ihn zugänglichen Informationen darstellen.[45]

Allerdings muss hierzu angemerkt werden, dass genau diese Tatsache auch etwas sein könnte, was man den Bibliothekssuchmaschinen, vorwerfen könnte, da auch in ihnen die Informationen aus verschiedenen Quellen stammen und unter Umständen nicht angezeigt wird, aus welcher Quelle die einzelne Information kommt.

An dieser Stelle muss zusätzlich darauf hingewiesen werden, dass normalerweise dann, wenn verschiedene Systeme unter einer Oberfläche angeboten werden sollen, alle Daten, die verarbeitet werden sollen, im gleichen Format vorliegen müssen, um von der Oberfläche interpretiert werden zu können.[46] Das heißt, letztlich sehen die Datensätze in der Ergebnisanzeige auch deshalb ähnlich aus, weil sie unter einer Oberfläche angezeigt werden.

Die Tatsache, dass die Daten eine ähnliche Struktur haben müssen, um unter einer Oberfläche angeboten werden zu können, ist eigentlich auch der wichtigste Grund für das Entstehen dieser Arbeit, da in ihr versucht wird, einen Weg zu finden, Daten aus zwei Systemen in eine Struktur zu bringen, die von der Bibliothekssuchmaschine ALBERT benötigt wird.

Ein weiteres Problem der herkömmlichen Katalogoberflächen ist, dass bei der Konzeption in der Regel nicht die Nutzer selbst gefragt wurden, welche Funktionen die Suchoberflächen haben sollen, obwohl die Nutzer schließlich mit den Oberflächen arbeiten sollen, sondern Bibliothekare, die andere Vorstellungen als die Nutzer haben. [47]

Der Grund für die unterschiedlichen Anforderungen liegt unter anderem sicher darin, dass die Bibliothekare die Struktur der Daten, die sich in dem jeweiligen System befinden, besser kennen und wissen, worauf sie bei einer Suche achten müssen, da sie die Daten auch in das System eingeben. Was ihnen unter diesen Voraussetzungen als nützlich und wichtig erscheint, kann von den Nutzern jedoch ebenso als überflüssig und wenig hilfreich empfunden werden.

---

[45] Vgl. Lewandowski (2010), S. 89.

[46] Vgl. Tennant (2003.

[47] Vgl. Casey (2007), S. 15.

Durch die Konkurrenz von anderen Informationsanbietern wie zum Beispiel Internetsuchmaschinen, Datenbanken und Verlagen,[48] rücken die Bedürfnisse und das Verhalten der Nutzer als Zielgruppe jedoch bei der Entwicklung neuer Oberflächen oder der Verbesserung der alten Oberflächen wieder mehr in den Fokus.

Zu den neuen Oberflächen gehören zum Beispiel die Bibliothekssuchmaschinen, die vor allem durch die Schnelligkeit der Suche, die große Anzahl an Treffern, das Ranking der Treffer nach Relevanz,[49] die intuitiven Bedienmöglichkeiten und die überschaubare und anpassungsfähige Darstellung der Ergebnisse bestechen.[50]
Diese Funktionen sind stärker an den Bedürfnissen und Vorlieben der Nutzer ausgerichtet, die sich in Bezug auf das Suchen durch die Verwendung von Internetsuchmaschinen verändert haben.[51]
Die Informationssuchenden möchten in der Regel einen unkomplizierten Weg, um möglichst schnell und erfolgreich an alle wichtigen Informationen zu gelangen.[52]

Da Internetsuchmaschinen diese Kriterien erfüllen, suchen die Informationssuchenden auch bevorzugt dort nach Informationen. Zusätzlich können die in Bibliotheken vorhandenen Informationen, wie zum Beispiel Bücher, Zeitschriftenbeiträge, Reports, Konferenzbeiträge oder andere fachwissenschaftliche Informationen mittlerweile zum Teil ebenso in den Internetsuchmaschinen gefunden werden.[53]
Da viele Informationssuchende ihr Informationsbedürfnis nicht genau spezifizieren können, wählen sie zufällige Suchterme, die ihnen geläufig sind[54] und finden so in normalen Bibliothekskatalogen wahrscheinlich ein paar passende Treffer. Da die Bibliotheken normiertes Vokabular verwenden und in der Regel die Suchbegriffe der Nutzer nicht damit übereinstimmen, entgeht den Nutzern auf die beschriebene

[48] Vgl. Lewandowski (2010), S. 87.

[49] Vgl. Kostädt (2008), S. 105.

[50] Vgl. Lossau (2004), S. 287.

[51] Vgl. Bertelmann; et al. (2007), S. 1302.

[52] Vgl. Bertelmann; et al. (2007), S. 1302.

[53] Vgl. Lewandowski (2006), S. 73-74.

[54] Vgl. Markey (2007).

Suchmethode wahrscheinlich ein Teil an relevanter Information, der durch die Volltextindexierung bei Internetsuchmaschinen, aber auch durch die größere Menge an „Datensätzen“, die durchsucht werden, vermutlich geringer sein dürfte.
So kann die eigentlich sinnvolle Verwendung von Schlagworten, durch die das Problem der unterschiedlichen Bezeichnungen für einen Sachverhalt verringert werden sollte, sogar entgegen ihrem Zweck zu weniger Suchergebnissen führen.

Allerdings können Informationssuchende auch genau das umgekehrte Problem bekommen. Wenn ihre Suchanfragen nämlich zu weit gefasst sind, erzielen sie zu viele Treffer, die sie dann nicht alle durchsehen und bewerten können.[55]
Bei den Internetsuchmaschinen ist jedoch hilfreich, dass zunächst mit einer großen Wahrscheinlichkeit Basisinformationen wie zum Beispiel Wikipedia-Artikel als Treffer angezeigt werden,[56] was den Suchenden dabei helfen kann, ihr Suchthema besser zu verstehen und dadurch einzugrenzen.
Trotz der Vorteile, die die Internetsuchmaschinen gegenüber den Bibliothekskatalogen haben, werden sie dennoch dafür kritisiert, dass sie die Online-Publikationen, die sie zugänglich machen, nicht gut genug erschließen, da keine bibliographischen Angaben und nicht genug Möglichkeiten zur Präzisierung von Suchanfragen vorhanden sind.[57]
Diese Tatsache zeigt, wie wichtig Bibliotheken in diesem Zusammenhang doch sein können, da sie zusätzliche Informationen zu den Volltexten zur Verfügung stellen, damit relevante Informationen auch gefunden werden, wenn ein Begriff, wie zum Beispiel das Erscheinungsjahr oder der Autor, nicht im Volltext genannt werden.

Wie bereits festgestellt, hat sich das Verhalten der Nutzer aufgrund der Suche mit Internetsuchmaschinen und dem allgemeinen Umgang mit dem Internet seit der Konzeption der Bibliothekskataloge verändert.
Ein Beispiel dafür, was sich geändert hat, ist, dass Nutzer in der Regel ihre Suche anfangs kaum einschränken, weil sie erwarten, dass die ersten Treffer in der

[55] Vgl. Larson (1991), S. 139.

[56] Vgl. Markey (2007).

[57] Vgl. Lauber-Reymann (2010), S. 153 .

Treffermenge die wichtigsten, d.h. für sie relevantesten, Informationen enthalten. Aus diesem Grund sehen sie sich häufig auch nur die ersten Treffer an, weshalb die Sortierung der Ergebnisse auch ein so wichtiges Kriterium für die Auswahl einer Suchoberfläche, die häufig genutzt werden soll, darstellt.[58] Dies zeigt auch, dass Informationssuchende erwarten, dass sie sofort das „Richtige" finden.[59]
Man will nicht erst überlegen müssen, welche Publikationsform die gewünschte Information haben soll, um dann nach der jeweils passenden Suchoberfläche suchen zu müssen, da dieser Vorgang viel Zeit benötigt und man die Suchanfrage für verschiedene Suchwerkzeuge jeweils anpassen und neu eingeben muss. [60]
Außerdem wird angenommen, dass die meisten Informationssuchenden die Unterschiede zwischen den verschiedenen Publikationsformen nicht kennen und daher dürfte ihnen auch nicht verständlich sein, warum sie in verschiedenen Systemen nach einer Information suchen sollten.[61]
Wie schon gesagt, schränken die Nutzer ihre Suchanfragen wenig ein, d.h. sie nutzen auch selten die Möglichkeiten ihre Suchanfrage zu präzisieren, wie sie zum Beispiel in herkömmlichen Bibliothekskatalogen angeboten werden.[62] Deshalb könnte man sich fragen, ob dann in den Suchoberflächen für die Nutzer wirklich so viele Suchfelder zur Verfügung gestellt werden sollten.
Indes gibt es auch Nutzer, die nicht wollen, dass sie nur noch einfache Suchmöglichkeiten angeboten bekommen, bei denen mit allen Begriffen aus einer Suchzeile immer alle Felder durchsucht werden, da sie dann keine präzisen Anfragen mehr formulieren können.[63]
Eine andere Entwicklung, die sich auf Seiten der Nutzer vollzogen hat, ist, dass sie nicht mehr nur nach Informationen suchen, sondern selbst Informationen erstellen und Informationen, die von anderen erstellt wurden, in Form von Kommentaren oder der Zuordnung von Schlagworten bewerten. Das bedeutet, der Nutzer hat eine neue Rolle bekommen und stellt deswegen auch andere Anforderungen an die

---

[58] Vgl. Lewandowski (2010), S. 91.

[59] Vgl. Soules (2010), S. 10-11.

[60] Vgl. Dempsey (2006).

[61] Vgl. Calhoun (2006), S. 35.

[62] Vgl. Soules (2010), S. 11.

[63] Vgl. Soules (2010), S. 11.

Informationssuche, wie zum Beispiel, dass er Kommentare zur Qualität des Gefundenen entdecken möchte.[64]

Man will also wissen, ob etwas hilfreich bzw. nützlich ist, bevor man eine Nutzung in Betracht zieht. Vergleichbar ist diese Situation damit, dass man, wenn man eine teure Anschaffung tätigen will, sich die Kommentare von anderen Käufern zu der Ware ansieht, bevor man sich dann unter anderem aufgrund der Kommentare selbst zum Kauf oder Nichtkauf entscheidet. Warum sollte man dieses Verhalten also nicht auch auf eine Informationssuche übertragen, da man in beiden Fällen doch von der Qualität der Ware bzw. Information abhängig ist, nachdem man sie gekauft bzw. genutzt hat.

Allerdings sind diese Informationen sehr subjektiv und stammen aus einem bestimmten Suchkontext, weshalb man sich gut überlegen muss, ob diese Bewertungen in einer Bibliothek, die in der Regel einen neutralen Zugang zu Informationen anbietet, verwendet werden sollen.

Des Weiteren wollen die Informationssuchenden überall und immer, wenn sie Informationen benötigen, danach suchen können und am liebsten auch direkt darauf zugreifen.[65]

Die Konsequenz aus dem genannten Verhalten und den Bedürfnissen der Nutzer, kann eigentlich nur sein, dass die Bibliotheken sich und ihre Suchoberflächen mehr daran orientieren müssen, um weiterhin wahrgenommen zu werden.

Angemerkt werden soll an dieser Stelle jedoch auch, dass eine Veränderung der Technologie, mit der die Informationen bereitgestellt werden, allein keine dauerhafte Steigerung der Nutzung hervorrufen wird.[66] Das bedeutet, man muss zum Beispiel auch Schulungen anbieten, um einen Zugang zu der Suchoberfläche zu schaffen.

Das Problem, das hier bei den herkömmlichen Suchoberflächen von Bibliothekskatalogen besteht, liegt vor allem darin, dass die Oberflächen nicht selbsterklärend genug sind, damit diejenigen, die nicht an einer Schulung teilgenommen haben, auch damit umgehen können.[67] Dieses Problem besteht,

---

[64] Vgl. Schmitt; Stehle (2010), S. 6.

[65] Vgl. Markey (2007).

[66] Vgl. Lewandowski (2010), S. 87.

[67] Vgl. Steiner (2007), S. 45.

obwohl die Kataloge von Anfang an einfach und ohne viel Übung nutzbar sein sollten.[68]

Demzufolge sollte noch mehr daran gearbeitet werden, die zukünftigen Suchoberflächen möglichst selbsterklärend zu gestalten, damit der Informationssuchende sich nicht auf die Beherrschung der richtigen Suchbefehle konzentrieren muss, sondern den Fokus auf den Inhalt seiner Suche legen kann.

2.3. Bibliothekssuchmaschinen

Eine Möglichkeit, die als neue Suchoberfläche besser die Nutzerbedürfnisse und das Nutzerverhalten beachten soll, ist die Bibliothekssuchmaschine. Sie kann als eine Art Weiterentwicklung der herkömmlichen Katalogoberflächen und Metasuchmaschinen angesehen werden.[69]

Die zugrundeliegende „Suchmaschinentechnologie […] kam in Deutschland erstmalig im Dreiländerkatalog des hbz zum Einsatz“.[70]

Eine Bibliothekssuchmaschine kann als ein Knotenpunkt betrachtet werden, an dem verschiedene und möglichst alle Informationen, die in einer Bibliothek verwaltet werden, gebündelt für den Nutzer zugänglich gemacht werden.

Die eigentlichen Informationen werden auch weiterhin in den dafür entwickelten Systemen verwaltet. Für die Verwendung in der Bibliothekssuchmaschine werden sie aus ihren Systemen exportiert und in den Index der Suchmaschine integriert.[71]

Das bedeutet, dass durch die Verwendung einer Bibliothekssuchmaschine eine Trennung von den Verwaltungssystemen und der Suchoberfläche für den Nutzer erzeugt wird.[72] Auf diese Weise werden also verschiedene Informationen, unter anderem auch aus Systemen, die nicht aus der Bibliothek stammen, aber trotzdem einen Mehrwert für die Nutzer bilden können, wie zum Beispiel Wikipedia-Artikel oder die Buchsuche von Google, unter einer Oberfläche vereinigt.[73]

---

[68] Vgl. Larson (1991), S. 134.

[69] Vgl. Keene (2011), S. 193.

[70] Wiesenmüller (2008), S. 28.

[71] Vgl. Lewandowski (2010), S. 89.

[72] Vgl. Coyle (2007b), S. 415 und vgl. Breeding (2010), S. 32.

[73] Vgl. Coyle (2007b), S. 416.

Für diese Vereinigung von verschiedenen Informationsquellen ist es nötig, dass die unterschiedlichen Quellen mit der Suchoberfläche kommunizieren bzw. Daten für die Suchoberfläche bereitstellen, was möglichst einheitlich geschehen sollte.[74] Dies geschieht in dem Beispiel, das in einem späteren Kapitel dieser Arbeit vorgestellt wird, über das Austauschformat XML.
Folglich wird mit einer Bibliothekssuchmaschine der Ansatz verfolgt, im ersten Schritt alle relevanten Informationen zu einer Suchanfrage aus der Gesamtmenge an vorhandenen Informationen zu filtern und dann im zweiten Schritt zu zeigen, von wo die gefundenen und ausgewählten Informationen bezogen werden können.[75]
Eigentlich wird jedoch gefordert, beide Schritte auf einmal zu realisieren, d.h. zu filtern und gleichzeitig zu zeigen, wo die Information verfügbar ist.[76]
Neben der bereits genannten Position zur Erklärung einer Bibliothekssuchmaschine gibt es ebenfalls den Ansatz, dass diese neue Oberfläche Funktionen des Web 2.0 enthalten soll. Dies bedeutet nicht nur, dass neue Mittel zur Kommunikation, wie Wikis oder Blogs, sondern auch die neuen Erwartungen von Nutzern in die Entwicklung einer Bibliothekssuchmaschine einbezogen werden müssen.[77]

Im Grunde erzeugt dies ebenso die Annahme, dass die zukünftigen Nutzer auch selbst an der Erstellung von bestimmten Funktionen bzw. Diensten der Suchmaschine beteiligt werden müssen.[78]
Eine weitere Folgerung ist, dass den Nutzern die Möglichkeit gegeben werden sollte, das fertige Produkt später noch mehr auf ihre persönlichen Bedürfnisse einstellen zu können.[79]
Da es jedoch andere von den Nutzern bereits akzeptierte und weit verbreitete Angebote für die Kommunikation gibt, sollten diese Komponenten nicht zu stark in die Suchmaschine integriert werden. Als wichtiger werden hingegen Schnittstellen

[74] Vgl. Dempsey (2006).
[75] Vgl. Lewandowski (2010), S. 88.
[76] Vgl. Dempsey (2006).
[77] Vgl. Schmitt; Stehle (2010), S. 6.
[78] Vgl. Schmitt; Stehle (2010), S. 8.
[79] Vgl. Schmitt; Stehle (2010), S. 10.

zu diesen Systemen, wie zum Beispiel zu Lernmanagementsystemen, betrachtet, um dort die bibliographischen Informationen aus der Bibliothek nutzen zu können.[80]

Grundsätzlich muss festgehalten werden, dass es nicht möglich ist eine eindeutige Definition zu finden, die alle Funktionalitäten beschreibt, die eine Bibliothekssuchmaschine ausmachen, da diese ständig um neue Angebote erweitert wird. Aus diesem Grund ist es auch nicht möglich ein Minimum an Funktionalitäten zu nennen, über das jede Bibliothekssuchmaschine verfügen muss.[81]
Prinzipiell sollte jede Bibliothek, die eine Bibliothekssuchmaschine einsetzen will, außerdem auch nur Funktionen anbieten, die auf ihre jeweilige Zielgruppe abgestimmt sind.
Daher können die folgenden Funktionalitäten nur als Ansammlung von Beispielen für Funktionen, die Bibliothekssuchmaschinen anbieten können, betrachtet werden.

Manchmal wird ein Versuch unternommen, die Funktionen in Kategorien einzuteilen, wobei die erste Kategorie als Funktionen, die zur Verbesserung der Suchmöglichkeiten beitragen, bezeichnet werden kann. Funktionen, durch die mehr Informationen in die Suchmaschine eingebracht werden, seien es mehr Informationsarten oder auch zusätzliche Informationen, können als zweite Kategorie benannt werden. Die dritte Art von Funktionen sind die, die es der Suchmaschine erlauben interaktiv zu sein, d.h. den Nutzern ermöglichen sich einzubringen und die Suchmaschine auf sich selbst anzupassen. Die letzte Kategorie sind Funktionen, die die Bündelung von Informationen und das Schaffen eines gemeinsamen Zugangs zu ihnen hervorrufen.[82]
Zu der ersten Kategorie, d.h. der Verbesserung der Suchmöglichkeiten, gehört eine der wichtigsten Eigenschaften, die eine Bibliothekssuchmaschine haben sollte, nämlich die Möglichkeit Suchergebnisse nach Relevanz zu sortieren. Einfach erklärt, wird dabei eine Häufigkeitsanalyse des Suchbegriffs bzw. der Suchbegriffe in den

[80] Vgl. Christensen (2009), S. 535-536.

[81] Vgl. Schmitt; Stehle (2010), S. 13.

[82] Vgl. Kneifel (2009), S. 27-28.

Metadaten und eine Analyse der Nähe der Suchbegriffe zueinander durchgeführt.[83] Allerdings können auch andere Faktoren für das Relevanzranking ausgewertet werden, wie zum Beispiel:

- die Anordnung der Suchbegriffe innerhalb des Treffers im Vergleich zu der Anordnung in der Suchanfrage
- die Ähnlichkeit der Schreibweise der Begriffe in Anfrage und Treffern, was zum Beispiel die Großschreibung betrifft
- die Aussagekraft des Feldes, in dem der Suchbegriff bzw. die Suchbegriffe gefunden wurden[84]
- die Verfügbarkeit der Information, d.h. ob das Medium zum Beispiel ausgeliehen ist
- die Aktualität der Information, insbesondere wenn die Bibliothekssuchmaschine von einem Publikum genutzt wird, dass auf aktuelle Informationen angewiesen ist[85]
- die Nähe der Information zum Nutzer in Relation dazu, ob der Nutzer sich in der Bibliothek, in einer Teilbibliothek oder zu Hause befindet
- die Publikationsform der Information, was dann wichtig ist, wenn in dem Fachgebiet, in dem die Bibliothekssuchmaschine eingesetzt wird, beispielsweise eher Aufsätze als Bücher verwendet werden[86]
- die Beliebtheit eines Mediums, was anhand von Ausleihzahlen und Exemplaranzahlen ermittelt werden könnte[87]

Zum Ranking muss jedoch angemerkt werden, dass die Analyse auf Relevanz, wenn nur die reinen Titelaufnahmen aus Bibliotheken verwendet werden, meist zu eher schlechten Ergebnissen führt, weil der Text, der ausgewertet werden kann, relativ kurz ist. Daher werden häufig Zusatzinformationen, wie Inhaltsverzeichnisse, Abstracts und wenn vorhanden auch der Volltext für das Ranking verwendet, d.h. es sollte so viel Inhalt wie möglich durch eine Bibliothekssuchmaschine verarbeitet

[83] Vgl. Kostädt (2008), S. 106.

[84] Vgl. Steiner (2007), S. 47.

[85] Vgl. Lewandowski (2010), S. 97.

[86] Vgl. Lewandowski (2010), S. 102.

[87] Vgl. Christensen (2009), S. 529.

werden.[88] Dabei ist zu beachten, dass die Treffer in diesen zusätzlichen Informationen anders gewertet werden müssen als die Treffer aus den Titelaufnahmen, da nicht alle Titelaufnahmen über Zusatzinformationen verfügen.[89]

Neben der Sortierung nach Relevanz, die wahrscheinlich das wichtigste Sortierkriterium sein wird, sollten aber auch andere Sortiermöglichkeiten, wie etwa nach Erscheinungsdatum oder Verfasser, vorhanden sein.[90]
Zur Verbesserung der Suchfunktionen trägt außerdem auch die Möglichkeit, Suchergebnisse nach bestimmten Kriterien zu gruppieren, erheblich bei, da die unterschiedlichen Gruppen eine einfache Reduzierung der Treffermenge ermöglichen können.[91] Hierbei werden bei bereits implementierten Bibliothekssuchmaschinen jedoch nicht, wie man vielleicht erwarten könnte, die thematischen Gruppierungen für eine Einschränkung der Treffermenge am häufigsten verwendet, sondern die Möglichkeit sich nur verfügbare Informationen anzeigen zu lassen.[92]
Durch diese verbesserten Navigationsmöglichkeiten, d.h. durch Auswahlmenüs und Cluster bzw. Facetten,[93] wird es dem Nutzer neben dem konkreten Suchen auch ermöglicht die Treffermenge zu erkunden.[94] Durch das Erkunden bietet sich die Möglichkeit Informationen zu finden, die der „Zielinformation“ thematisch ähnlich sind und andere Gemeinsamkeiten mit ihr haben, wie zum Beispiel die Publikationsform oder das Erscheinungsjahr.[95] Das heißt, dass man aufgrund der Gruppen vielleicht auch Aspekte an seinem Thema entdeckt, die eine Spezialisierung oder eine Erweiterung des Themas bzw. einen neuen Sucheinstieg ermöglichen. Diese Aspekte wären vielleicht ohne die Gruppen verborgen geblieben. Demzufolge kann die Gruppierung von Suchergebnissen einen großen Einfluss auf den Verlauf und das Ergebnis einer Suche nehmen.

---

[88] Vgl. Casey (2007), S. 20.

[89] Vgl. Lewandowski (2010), S. 101.

[90] Vgl. Casey (2007), S. 19.

[91] Vgl. Casey (2007), S. 19.

[92] Vgl. Christensen (2009), S. 532-533.

[93] Vgl. Soules (2010), S. 12.

[94] Vgl. Lewandowski (2010), S. 90.

[95] Vgl. Kostädt (2008), S. 105.

Zu der ersten Kategorie von Funktionen kann ebenfalls die Möglichkeit gezählt werden, Suchanfragen automatisch immer wieder durchführen zu lassen, sodass man neue Titel zu dem Thema der Suche als RSS-Feed abonnieren kann.[96]
Das Präsentieren von Vorschlägen zur Vervollständigung bzw. Ergänzung bei der Eingabe einer Suchanfrage, wozu auch die Korrektur von Rechtschreibfehlern gezählt werden kann, gehört ebenso zu den Funktionen, die eine Bibliothekssuchmaschine zur Unterstützung der Suche anbieten kann.[97]

Außer diesen Funktionen, die nicht unbedingt in den herkömmlichen Bibliothekskatalogen vorhanden sind, sollten einige der Funktionen der Kataloge beibehalten werden. Gleichwohl muss gesagt werden, dass viele Funktionen, die in diesem Kapitel aufgeführt werden, bereits in einigen Katalogen umgesetzt werden,[98] da diese auch immer mehr auf die Bedürfnisse der Nutzer ausgerichtet werden.

So soll beispielsweise auch in den Suchmaschinen die Gelegenheit gegeben werden, komplexere Suchanfragen zu stellen. Dies sollte über das Angebot zur Nutzung von Booleschen Operatoren, Phrasensuche und Trunkierung,[99] sowie über das Angebot einer komplexen Suchmaske ermöglicht werden. Allerdings sollte die komplexe Suchmaske einfach handhabbar sein und nur dann angeboten werden, wenn sie auch wirklich verwendet wird. Das heißt, man muss die Nutzungshäufigkeit beobachten und die Suchmaske gegebenenfalls verändern oder aus dem Angebot entfernen.[100]
Gleiches sollte auch für alle anderen Funktionen, die eine Suchmaschine anbieten kann, gelten.
Andere Funktionen, die aus den herkömmlichen Bibliothekskatalogen übernommen werden sollten, sind die Möglichkeit auf das Benutzerkonto mit der gleichen Oberfläche, mit der die Suche stattfindet, zugreifen zu können und die Anzeige von Informationen zur Verfügbarkeit und zu Vormerkungen.[101]

---

[96] Vgl. Kostädt (2008), S. 107-108.

[97] Vgl. Kneifel (2009), S. 52 und 54.

[98] Vgl. Steiner (2007), S. 83.

[99] Vgl. Schmitt; Stehle (2010), S. 72.

[100] Vgl. Casey (2007), S. 19-20.

[101] Vgl. Breeding (2007), S. 35.

Grundsätzlich sollten also alle Funktionen, die bisher hilfreich und deshalb sinnvoll waren, auch in den neuen Systemen beibehalten werden.
Außerdem wird empfohlen die vorhandenen Titelaufnahmen mit weiteren Informationen, d.h. etwa mit Inhaltsverzeichnissen, Rezensionen, Verlagsinformationen und Buchcovern, anzureichern.[102]
Neben dieser Art von zusätzlicher Information, sollten die Volltexte, soweit vorhanden, direkt aus der Suchmaschine heraus zugänglich sein.[103]
Aber zu den zusätzlichen Inhalten, gehören nicht nur Informationen, sondern auch Leistungen, zu denen beispielsweise Empfehlungen von ähnlichen Titeln und Neuerwerbungslisten gezählt werden können.[104]
Unter anderem werden zu diesen Leistungen auch Angebote gerechnet, die auf Basis des Themengebietes, das durchsucht wurde, erstellt werden. Zu ihnen gehören zum Beispiel Empfehlungen von thematisch passenden Fachdatenbanken, aktuelle Nachrichten für das Fachgebiet, Empfehlungen von inhaltlich relevanten Schulungsangeboten oder die Angabe von Informationen zu möglichen Ansprechpartnern in der Bibliothek.[105]
Wie zu Beginn des Kapitels bereits erwähnt, wird häufig auch die Beteiligung der Nutzer als Element der Bibliothekssuchmaschinen betrachtet. Um die Nutzer zu beteiligen, gibt es verschiedene Möglichkeiten, die einen Teil der dritten Kategorie von Funktionen darstellen.
Nutzer könnten in Bibliothekssuchmaschinen Bewertungen zu Medien verfassen,[106] Medien verschlagworten oder auch Empfehlungen zu bestimmten Themen schreiben,[107] obwohl letztere Funktion, wenn sie bereits in bestehenden Systemen eingesetzt wird, nur zögerlich von den Nutzern verwendet wird.[108]
Die Idee, Nutzer zu beteiligen und sie sich so selbst einen Mehrwert schaffen zu lassen, ist auch schon um 1990 angedacht worden. Damals sollten die Medien mit

---

[102] Vgl. Calhoun (2006), S. 19.

[103] Vgl. Calhoun (2006), S. 19.

[104] Vgl. Calhoun (2006), S. 19.

[105] Vgl. Blenkle; Ellis; Haake (2009), S. 621.

[106] Vgl. Lewandowski (2010), S. 89.

[107] Vgl. Kostädt (2008), S. 109.

[108] Vgl. Kostädt (2008.), S. 111.

Schlagworten, die auf die Nutzer angepasst waren und deshalb mehr aus dem natürlichem Sprachgebrauch stammten, zugänglich gemacht werden.[109] Wenn Nutzer nun Begriffe für einen Titel eingeben hatten, die inhaltlich passend waren, aber noch nicht verwendet wurden, sollten diese auch für den Titel verwendet werden.[110] Alle Schlagwörter, egal ob die Genannten, die in einem eigenen Thesaurus verwaltet werden sollten, oder die, mit denen Bibliothekare die Titel versehen hatten, sollten untereinander stark verknüpft werden,[111] damit automatisch verwandte oder ähnliche Begriffe und damit auch Datensätze vorgeschlagen werden konnten.[112]
Eine andere Möglichkeit eine Empfehlung für einen Titel in einer Bibliothekssuchmaschine zu erzeugen, kann auch aus dem System selbst kommen, indem die Metadaten von Titeln, auf ihre Ähnlichkeit bezogen, ausgewertet werden.[113]
Wenn jedoch der Nutzer beteiligt werden soll, könnte neben der direkten Beteiligung, d.h. der Nutzer verfasst selbst Zusatzinformationen, auch eine indirekte Beteiligung z.B. über die Ausleihzahlen von Medien erfolgen. Von dieser Methode wird aber abgeraten, da dies Probleme mit dem Datenschutz hervorrufen könnte.[114]

Abgesehen von den Funktionen, bei denen ein Nutzer anderen Nutzern behilflich sein kann, gehören auch die Angebote, bei denen die Suchmaschinen personalisiert werden können, zu den Funktionen der dritten Kategorie.[115] Denkbar wäre beispielsweise, dass ein Nutzer sich einen bestimmten Suchraum als Standardsuchraum speichern kann. Um dies zu ermöglichen, wäre es nötig eine Standardeinstellung für alle Nutzer zu treffen, die dann mit Funktionen erweitert bzw. verkleinert werden kann.

---

[109] Vgl. Bates (1986), S. 370.

[110] Vgl. Bates (1986), S. 369.

[111] Vgl. Bates (1986), S. 370.

[112] Vgl. Bates (1986), S. 368.

[113] Vgl. Steiner (2007), S. 49-50.

[114] Vgl. Steiner (2007), S. 49-50.

[115] Vgl. Christensen (2009), S. 529.

Das Schaffen eines besseren Zugangs zu den Informationen (vierte Kategorie) kann einerseits auf die Nutzer einer Bibliothek und andererseits auch auf andere Systeme bezogen werden.
Im ersten Fall gehört sicherlich eine ansprechende visuelle Aufbereitung der Inhalte zu den wichtigsten Möglichkeiten, durch die ein Nutzer einen besseren Zugang zu den Informationen[116] und so zu der Bibliothekssuchmaschine findet.
Ebenso soll der Nutzer sich Neuerscheinungen zum Beispiel als RSS-Feeds abonnieren können,[117] Titel als Literaturlisten exportieren[118] und als Lesezeichen speichern können, indem die Titelaufnahmen über einen sogenannten Permalink dauerhaft adressierbar gemacht werden.[119]
Um die Daten für andere Systeme zugänglich zu machen, ist es natürlich nötig, die Daten exportieren zu können oder über Schnittstellen zum Austausch mit den anderen Systemen zu verfügen.[120]
Außerdem wird im Zusammenhang mit der Interaktion mit anderen Systemen geraten, weiterhin mit Technologien zu arbeiten, die man bereits für die Verwaltung der Informationen einsetzt, sich aber auch grundsätzlich immer um bessere Systeme zu bemühen.[121] Ein Beispiel dafür ist in dieser Arbeit die EZB, da sie auch weiterhin als das Verwaltungssystem für elektronische Zeitschriften verwendet wird, anstatt dass ein neues System eingeführt wird.
Dieser Ansatz wird gleichwohl auch kritisiert, da man zum Beispiel, wenn man die Suchoberfläche auf ein bestehendes Bibliothekssystem aufsetze, von dem Code, der in einem proprietären Format geschrieben sei, abhängig sei und das ganze System deshalb nicht beliebig erweitert werden könne. Außerdem gebe es dann mindestens zwei Stellen, die gepflegt werden müssten.[122]
Dem kann wiederum entgegnet werden, dass an anderen Stellen, wie zum Beispiel bei Webseiten eine Trennung von Inhalt und Layout explizit gewünscht wird, da

---

[116] Vgl. Breeding (2010), S. 32 und Christensen (2009), S. 529.

[117] Vgl. Steiner (2007), S. 83.

[118] Vgl. Casey (2007), S. 21.

[119] Vgl. Steiner (2007), S. 49.

[120] Vgl. Christensen (2009), S. 529.

[121] Vgl. Calhoun (2006), S. 17 und 19.

[122] Vgl. Casey (2007), S. 22.

wenn eines von beiden geändert werden muss, das andere unverändert bleiben kann. Das kann in Bezug auf die Bibliothekssuchmaschinen auch bedeuten, dass man einfach die Suchoberfläche wechseln kann, ohne die Eingabeform der Daten ändern zu müssen.
Ein anderer Vorteil daran, dass die Daten aus verschiedenen Systemen stammen, kann auch darin liegen, dass sie unterschiedliche Strukturen haben, die jeweils auf ihre Eigenschaften abgestimmt sind. So können auch all diese Informationen unter einer Oberfläche zum Auffinden der Daten verwendet werden. In der Regel können nämlich die Spezifika der einzelnen Daten bei der Verwaltung in einem gemeinsamen System nicht genug berücksichtigt werden.
Jedoch könnte die unterschiedliche Struktur der Daten auch dazu führen, dass bestimmte Funktionen, wie das Gruppieren, nicht mehr so einfach zu implementieren sind.[123] Daraus lässt sich ableiten, dass zwar Unterschiede zwischen den Strukturen der Daten bestehen sollen, es aber eine Anzahl von Elementen geben muss, die in allen Strukturen enthalten sein sollten.
Problematisch an den verschiedenen Informationsarten könnte auch, je nach Konzeption der Suchmaschine, sein, dass es schwieriger wird, die gewünschte Information in einer großen Treffermenge zu finden. Beispielsweise könnte eine gesuchte Zeitschrift sich in der Trefferliste weit hinten befinden, weil zunächst alle Artikel aus der Zeitschrift angezeigt werden.[124]
Zusätzlich ist bedenkenswert, dass man bestimmte Quellen festlegen muss, aus denen die Informationen in der Bibliothekssuchmaschine stammen sollen. Vermutlich werden diese Quellen aber auch nicht alle in der Bibliothek vorhandenen Informationen darstellen und deshalb könnte es im Vergleich zu den verteilten Suchmöglichkeiten noch schwieriger werden, die Informationen, die nicht in der Suchmaschine sind, zu finden.[125]
Nachdem auf die möglichen Funktionen eingegangen und festgestellt wurde, dass die neuen Oberflächen mehr an die Bedürfnisse der Nutzer angepasst werden müssen, soll nun aufgeführt werden, welche Funktionen die Nutzer bevorzugen. Dafür

[123] Vgl. Dempsey (2006).

[124] Vgl. Keene (2011), S. 195.

[125] Vgl. Keene (2011), S. 195.

werden die Ergebnisse einer Untersuchung in der Stadtbücherei Frankfurt am Main 2008 mit etwa 1000 Teilnehmern genutzt.[126] In der Untersuchung wurde herausgefunden, dass die Nutzer sich wünschen, den Katalog an ihre Vorlieben anpassen zu können, Neuerwerbungslisten zu erhalten, Empfehlungen zu bestimmten Themen nutzen zu können, neue Medien zu einer Suchanfrage per RSS-Feed abonnieren zu können[127] und dass die Daten mit zusätzlichen Informationen angereichert werden sollen.[128]
Außerdem hätten die Nutzer gerne, dass Datenbanken in den Suchraum integriert werden,[129] dass sie ihre Suche mit Filtern oder einer komplexen Suchmaske einschränken können, dass die Ergebnisse nach Relevanz sortiert werden können[130] und dass es eine Rechtschreibfehlerkorrektur, sowie eine einfache Suchzeile geben soll.[131]
Grundsätzlich soll die Bibliothekssuchmaschine also möglichst viele Funktionen bieten und dabei möglichst einfach aussehen und zu bedienen sein.[132]

2.4. ALBERT

ALBERT ist ein Beispiel für eine Bibliothekssuchmaschine, die aus einer Kooperation des KOBV und der Bibliothek des Wissenschaftsparks Albert Einstein in Potsdam entstanden ist.[133] Die Abkürzung ALBERT steht dabei für „**A**ll **L**ibrary **B**ooks, journals and **E**lectronic **R**ecords **T**elegrafenberg“.
Die erste Version der Suchmaschine wurde im Oktober 2007 in Betrieb genommen[134] und Ende 2010 von einer zweiten Version abgelöst.[135] Sie enthält momentan circa 6 Millionen Datensätze.

---

[126] Vgl. Kneifel (2009), S. 83.
[127] Vgl. Kneifel (2009), S. 92.
[128] Vgl. Kneifel (2009), S. 95.
[129] Vgl. Kneifel (2009), S. 94.
[130] Vgl. Kneifel (2009), S. 96.
[131] Vgl. Kneifel (2009), S. 97.
[132] Vgl. Casey (2007), S. 23.
[133] Vgl. Bertelmann; et al. (2007), S. 1302.
[134] Vgl. Bertelmann; et al. (2007), S. 1306.
[135] Vgl. Höhnow (2010), S. 165.

Die Basis der Suchmaschine bilden Open-Source-Technologien, die aus dem Bereich der Java-Technologien kommen. Auch die verwendete Suchmaschinentechnologie ist eine Java-Version der Suchsoftware Lucene/SOLR,[136] die sich dazu eignet Volltextsuchen für verschiedenste Inhalte durchzuführen.[137]
ALBERT selbst soll ebenfalls von anderen Bibliotheken nachgenutzt werden können.[138]
Ziel bei der Konzeption der Bibliothekssuchmaschine war, fachlich relevante Informationen einfach zugänglich zu machen. Das heißt man wollte mit einer „benutzerfreundlichen Suchoberfläche“[139] eine Möglichkeit zur Verfügung stellen, mit der schnell und bequem gesucht werden kann.[140]
Außerdem sollte ein komfortabler Zugriff auf Volltexte direkt aus der Trefferanzeige heraus ermöglicht werden. Ebenso war es wichtig, verschiedene relevante Informationsquellen mit unterschiedlich strukturierten Inhalten unter einer Oberfläche anbieten zu können.[141] Ziel hierbei war aber nicht, alle Informationsquellen, insbesondere lizensierte Bestände, in die Suchmaschine zu integrieren, da ALBERT nicht zu einer Metasuchmaschine werden sollte.[142]
Zu den Informationsquellen, die ausgewertet werden, gehören beispielsweise das Bibliothekssystem, die Publikationsdatenbanken,[143] RSS-Feeds mit Zeitschrifteninhaltsverzeichnissen,[144] wissenschaftliche Primärdaten[145] und Datensätze aus den Nationallizenzen, die mit den Fächern, mit denen in der EZB die Zeitschriften kategorisiert werden, angereichert wurden.[146]

---

[136] Vgl. Bertelmann; et al. (2007), S. 1302-1303.

[137] Vgl. Kostädt (2008), S. 107.

[138] Vgl. Bertelmann; et al. (2007), S. 1306.

[139] Herm; Volz (2008), S. 4.

[140] Vgl. Herm; Volz (2008), S. 4.

[141] Vgl. Herm; Volz (2008), S. 4.

[142] Vgl. Herm; Volz (2008), S. 9.

[143] Vgl. Herm; Volz (2008), S. 5.

[144] Vgl. Bertelmann; et al. (2007), S. 1305.

[145] Vgl. Bertelmann; et al. (2007), S. 1306.

[146] Vgl. Höhnow (2010), S. 167.

Ferner sollten zum besseren Auffinden von Informationen auch zusätzliche Informationen, wie Abstracts, Verlagsangaben und die freizugänglichen Volltexte von Titeln, in der Suchmaschine indexiert werden und die Trefferlisten mit einer besseren Sortierung für den Nutzer attraktiver präsentiert werden.[147]
Insgesamt sollte also ein gemeinsamer Sucheinstieg zu den Informationen geboten werden, um den Nutzer dann auf die spezifisch benötigte Information weiterzuleiten.[148]
Suchanfragen können mit einer einfachen Suchzeile, in der die Suchbegriffe mit einer logischen „UND"-Verknüpfung verbunden werden, durchgeführt werden.[149] Es kann aber auch eine komplexere Suchmaske mit zwei Suchzeilen, die auf bestimmte Felder eingegrenzt werden können, genutzt werden.
Aufgrund der erwähnten Fachgebiete aus der EZB kann auch eine Personalisierung der Suchmaschine zum Beispiel in Form einer thematischen Einschränkung bzw. Erweiterung des Suchraums vorgenommen werden, [150] wobei natürlich nicht alle Einträge in ALBERT mit einem Themengebiet aus der EZB versehen sind, wie zum Beispiel die Medien aus dem Bibliothekssystem. Diese Medien bleiben von dieser Funktion ausgenommen und gehören zum Beginn einer Suche immer zum Suchraum.
Ebenfalls ist es nach der Anzeige der Trefferliste möglich, die Treffermenge mittels Gruppen, die sich nach einer Einschränkung dynamisch an die neue Treffermenge anpassen, zu verringern. So kann zum Beispiel zunächst zwischen den Publikationsformen ausgewählt werden, um dann, wenn man seine Treffermenge auf Bücher beschränkt hat, den Standort, d.h. die Teilbibliothek, als weiteres Auswahlkriterium zu verwenden.[151]
Eine andere Publikationsform, die auch Relevanz für diese Arbeit besitzt, sind die aktuellen Zeitschriftenaufsätze. Im Prinzip sind dies aktuelle Inhaltsverzeichnisse von Zeitschriften, die als RSS-Feeds abonniert werden, um sie in ALBERT zu

---

[147] Vgl. Herm; Volz (2008), S. 5.

[148] Vgl. Bertelmann; et al. (2007), S. 1303.

[149] Vgl. Bertelmann; et al. (2007), S. 1304.

[150] Vgl. Höhnow (2010), S. 167.

[151] Vgl. Bertelmann; et al. (2007), S. 1304-1305.

integrieren. Diese Feeds können schließlich auch von den Nutzern aus ALBERT abonniert werden. Durch diese Funktion werden die Zeitschriftenaufsätze also schneller nach ihrem Erscheinen für die Nutzer zugänglich gemacht, sodass man nicht mehr darauf warten muss, dass die Aufsätze in entsprechende Fachdatenbanken aufgenommen werden, damit man sie finden kann.[152]

Insgesamt werden nur Zeitschrifteninhaltsverzeichnisse abonniert, bei denen die Bibliothek auch einen Volltextzugang zu den Zeitschriften besitzt,[153] da der Volltext direkt von der Trefferanzeige verlinkt wird. Diese Daten werden ungefähr ein Jahr lang in ALBERT vorgehalten,[154] da davon ausgegangen wird, dass die Artikel nach diesem Zeitraum auch in den Fachdatenbanken verzeichnet sind.

Um die Informationen aus den unterschiedlichen Systemen für ALBERT nutzbar zu machen, werden sie aus den bestehenden Systemen exportiert und in den Suchindex der Suchmaschine integriert. Beispielsweise werden für den lokal vorhandenen Medienbestand der Bibliotheken die Daten aus dem Bibliothekssystem SISIS und einer Allegro-Datenbank exportiert und als XML-Dokumente im MAB-Format (im Maschinellen Austauschformat für Bibliotheken) für die Integration in den Index bereitgestellt. [155] Aber auch Daten aus anderen Systemen werden als XML-Dokumente für den Suchindex zur Verfügung gestellt, wie zum Beispiel die Metadaten aus den Nationallizenzen, die dafür von MAB zu MAB-XML umgewandelt wurden.[156]

Auch aus diesem Grund wurde für die Daten aus der EZB und den Zeitschriften-Feeds, wie es in einem späteren Kapitel dieser Arbeit noch beschrieben werden soll, XML als Format zur Integration in den Suchindex ausgewählt und die Dateien mit XPath (XML Path Language), XSLT (Extensible Stylesheet Language Transformation) und Apache Ant in die benötigte Struktur umgewandelt.

Diese Instrumente sollen im nun folgenden Teil der Arbeit beschrieben werden.

---

[152] Vgl. Bertelmann; et al. (2007), S. 1305.

[153] Vgl. Höhnow (2010), S. 168.

[154] Vgl. Bertelmann; et al. (2007), S. 1305.

[155] Vgl. Herm; Volz (2008), S. 14.

[156] Vgl. Höhnow (2010), S. 167.

3. Werkzeuge zur Gestaltung des Prozesses

Wenn XSLT beschrieben werden soll, muss zunächst XSL (Extensible Stylesheet Language) genannt werden. XSL ist eine Sprache, mit der man XML-Dokumente anders als in ihrer ursprünglichen Form darstellen kann und für die es seit Oktober 2001 eine Empfehlung vom W3C (World Wide Web Consortium) gibt.[157]
XSL hat zwei Bestandteile, nämlich XSLT und XSL FO (Extensible Stylesheet Language Formatting Objects). XSL FO dient zur Formatierung von XML-Dokumenten[158] und wird im Verlauf der Arbeit nicht weiter beachtet, da es nicht zur Erstellung der Prozesse verwendet wurde.
XSLT hingegen, das eine Sprache zum Umwandeln von XML-Dokumenten in das gleiche oder ein anderes Format ist,[159] soll in einem folgenden Kapitel näher beschrieben werden.
Zunächst soll jedoch eine Beschreibung von XPath, als ein Element, das in XSLT verwendet wird,[160] erfolgen.

3.1. XPath

XPath ist eine Sprache, mit der Inhalte in XML-Dokumenten angesprochen werden können und die dafür die Baumstruktur der XML-Dokumente nutzt.[161] Um die Inhalte anzusprechen, werden nämlich die Pfade entlang der Struktur des XML-Dokuments verwendet.[162]
XPath ist allerdings kein Teil des Vokabulars von XML. Es kann in der Funktion des Adressierens in Bezug auf die Struktur der Pfade mit den Pfaden zu Ordnern oder Dateien in Computern verglichen werden,[163] nur dass mit XPath verschiedene Knoten innerhalb eines XML-Dokuments adressiert werden.
Zu den verschiedenen Knotenarten eines XML-Dokuments gehören der Wurzelknoten, der virtuell als oberster Knoten über allen anderen steht, sowie

---

[157] Vgl. Vonhoegen (2009), S. 242.

[158] Vgl. Jackenkroll (2003), S. 27.

[159] Vgl. Jackenkroll (2003), S. 26.

[160] Vgl. Jackenkroll (2003), S. 30.

[161] Vgl. Jackenkroll (2003), S. 27.

[162] Vgl. Jackenkroll (2003), S. 28.

[163] Vgl. Kränzler (2002), S. 201.

Elementknoten, Attributknoten, Knoten mit Textinhalt, Namensraumknoten, Verarbeitungsanweisungsknoten und Kommentarknoten.[164]

Neben der Funktion des Adressierens, kann XPath auch durch eine Anzahl von Funktionen Operationen durchführen.[165]

In seiner zweiten Version ist XPath eine Abfragesprache.[166] Da diese Funktion jedoch für die vorliegende Arbeit nicht verwendet wird, wird nur auf die erste Version eingegangen.

Wie schon gesagt, wird für das Ansprechen von Inhalten ein Weg entlang der Struktur eines XML-Dokuments verwendet. Dieser Weg zu den Inhalten erfolgt über die sogenannten Achsen, die Verbindungen zwischen den Inhalten bzw. Knoten in einem Dokument herstellen. Die Achsen sind wichtig, da die Verarbeitung durch einen Prozessor immer bei einem Knoten startet und von dort zu dem Nächsten weitergeleitet wird.[167]

Das Benennen des nächsten Knotens und der Achse, die verwendet wird, um ihn zu erreichen, nennt sich Lokalisierungsschritt. Grundsätzlich sind die Lokalisierungsschritte wie folgt aufgebaut „Achsenname::Knoten“.

Insgesamt gibt es 13 Achsen um Knoten zu selektieren:

- self - der aktuelle Knoten soll angesprochen werden. Die Abkürzung in einem XPath-Ausdruck dafür ist ein „.“.
- child - ein Kindknoten, d.h. ein direkt untergeordneter Knoten soll angesprochen werden. In der abgekürzten Schreibweise wird die Bezeichnung der Achse komplett wegelassen und nur der Name des Kindknotens im Lokalisierungsschritt angegeben. Mit dieser Achse können keine Namensräume oder Attribute adressiert werden, weil diese Knotenarten keine Kindknoten sein können. Außerdem liegen diese Knotenarten auch auf eigenen Achsen.
- descendant - untergeordnete Knoten, auch Nachfahren genannt, sollen ausgewählt werden.

---

[164] Vgl. Montero Pineda (2004), S. 42.

[165] Vgl. Jackenkroll (2003), S. 30.

[166] Vgl. Kränzler (2002), S. 201.

[167] Vgl. Montero Pineda (2004), S. 42.

- descendant-or-self - untergeordnete Knoten und der aktuelle Knoten sollen angesprochen werden. Die Abkürzung für die Selektion dieser Knoten ist „.//*“.[168] Das Sternchen steht für einen beliebigen Knotennamen.
- parent - der Elternknoten, d.h. der direkt übergeordnete Knoten, soll verwendet werden. Abgekürzt wird diese Achse mit „../*“.
- ancestor - übergeordnete Knoten, auch Vorfahren genannt, sollen ausgewählt werden. Zu diesen Knoten gehört auch der Elternknoten.
- ancestor-or-self - übergeordnete Knoten und der aktuelle Knoten sollen adressiert werden.
- preceding - vorhergehende Knoten, die keine Vorfahren sind, da diese bereits über die ancestor-Achse angesprochen werden können, sollen ausgewählt werden.[169]
- preceding-sibling - vorhergehende Knoten, die keine Vorfahren sind und sich auf der gleichen Ebene wie der aktuelle Knoten befinden, sollen selektiert werden.
- following - nachfolgende Knoten, die keine Nachfahren sind, sollen angesprochen werden.
- following-sibling - nachfolgende Knoten, die keine Nachfahren sind und sich auf der gleichen Ebene wie der aktuelle Knoten befinden, sollen ausgewählt werden.[170]
- namespace - Namensräume des aktuellen Knotens sollen verwendet werden.
- attribute - Attribute des aktuellen Knotens sollen selektiert werden. Abgekürzt geschrieben werden mit „@*“ alle Attribute eines Knotens angesprochen.[171]

Mit diesen Achsen können die Wege zu den einzelnen Knoten des XML-Dokuments angegeben werden. Wenn man nämlich mehrere Lokalisierungsschritte, getrennt von einem „/“ hintereinander angibt, kann man einen Pfad entlang der Struktur des XML-Dokuments nachbilden,[172] der dann als Lokalisierungspfad bezeichnet wird.[173]

---

[168] Vgl. Montero Pineda (2004), S. 43.

[169] Vgl. Montero Pineda (2004), S. 44.

[170] Vgl. Kränzler (2002), S. 205.

[171] Vgl. Lenz (2006), S. 17.

[172] Vgl. Lenz (2006), S. 20.

[173] Vgl. Kränzler (2002), S. 204.

Beispielsweise bewirkt „child::element/attribute::id“ (abgekürzt „element/@id“), dass Attribute mit dem Namen „id“ von Kindelementen des aktuellen Knotens mit dem Namen „element“ ausgewählt werden.
Neben den Pfadangaben, die vom aktuellen Knoten ausgehen und als relativ bezeichnet werden, können auch sogenannte absolute Pfade, die immer beim Wurzelknoten beginnen und deshalb mit einem „/“ anfangen, verwendet werden. Bei ihnen wird also die gesamte Struktur des Dokuments bis zu den gewünschten Elementen angegeben.[174]
Bei den Pfadangaben gibt es nur eine Sache, die immer angegeben werden muss und das ist das Muster für die Elemente, die angesprochen werden sollen, da man in der verkürzten Schreibweise für die Kindachse beispielsweise die Angabe der Achse weglassen kann. Die Angabe der gewünschten Knoten ist der Teil eines Lokalisierungsschrittes, der hinter dem doppelten Doppelpunkt steht und wird auch Knotentest genannt.[175] Diese Angabe ist nötig, da mit ihr ein Muster definiert werden kann, mit dem alle Knoten, die auf der angegebenen Achse liegen, darauf getestet werden, ob sie dem Muster entsprechen und deshalb weiterverarbeitet werden sollen.

Außer der Angabe von Pfaden können in XPath-Ausdrücken auch Bedingungen für die Knoten angegeben werden, die man selektieren möchte. Diese Bedingungen werden als Prädikate bezeichnet und in eckigen Klammern hinter das Muster geschrieben, auf das sie sich beziehen. Prädikate können weitere Pfade sein und Boolesche Operatoren, Vergleichsoperatoren, Zeichenketten und Zahlen, sowie Funktionen, die später noch erläutert werden sollen, enthalten.[176]
Zum Beispiel würde der Ausdruck „element[@id]“ bedeuten, dass Kindelemente des aktuellen Knotens mit dem Namen „element“, die ein Attribut namens „id“ besitzen, verarbeitet werden sollen. Der Ausdruck „element[@id=‘eins‘][2]“ würde beispielsweise nach Kindelementen <element> suchen, die ein Attribut mit dem Namen „id“ besitzen, welches den Wert „eins“ hat. Von diesen Elementen würde dann das zweite Element, das die Bedingungen erfüllt, selektiert werden.

---

[174] Vgl. Montero Pineda (2004), S. 45-46.

[175] Vgl. Kränzler (2002), S. 206.

[176] Vgl. Montero Pineda (2004), S. 50.

Ein Beispiel für die Verwendung eines Booleschen Operators ist „element[@id='eins' or @id='zwei']". Durch diese Aussage würden alle Kindelemente <element> ausgewählt, die entweder ein Attribut „id" mit dem Wert „eins" oder dem Wert „zwei" haben.
Die Prädikate können in den Lokalisierungspfaden hinter jedem Knoten stehen, d.h. sie müssen sich nicht immer auf den letzten Schritt beziehen.
Bei der Selektion von Knoten mittels XPath ist wichtig zu wissen, dass die Ergebnismenge davon abhängig ist, in welchem Kontext der verarbeitende Prozessor sich gerade befindet. Zu einem Kontext gehören immer der Knoten, der gerade bearbeitet wird, die Position des Knotens in dem Kontext, in dem er aufgerufen wurde, sowie die Größe dieses Kontextes. Zusätzlich kann die Ergebnismenge auch von Variablen, einer Namensraumdeklaration und von Funktionen beeinflusst werden.[177]
Der aktuelle Knoten, d.h. der Knoten, der gerade vom Prozessor bearbeitet wird, ist dabei in der Regel der Startpunkt für den Pfad und die Verarbeitung des nächsten Knotens.[178]
Wie bereits erwähnt, können XPath-Ausdrücke auch aus Funktionen bestehen bzw. Funktionen in Lokalisierungspfaden enthalten. Die Funktionen können in vier Kategorien eingeteilt werden. Diese vier Kategorien sind: Funktionen für Knotenmengen, Funktionen für Zeichenketten, Boolesche Funktionen und numerische Funktionen.
Im Folgenden sollen nur die Funktionen, die auch für den Prozess benötigt wurden, der Thema eines späteren Kapitels ist, beschrieben werden.
Zu den Funktionen, die auf Knotenmengen angewendet werden können, gehören zum Beispiel die Funktion last( ), mit der die Größe des Kontextes, in dem der Prozessor gerade arbeitet, ausgegeben wird und die Funktion position( ), die die Position des aktuellen Knotens in diesem Kontext liefert. [179]
Eine hilfreiche Funktion, mit der mehrere Zeichenketten mit einem Trennzeichen zu einer Zeichenkette zusammengefügt werden können, ist concat( ), wobei die

[177] Vgl. Bach (2000), S. 67.

[178] Vgl. Bach (2000), S. 67.

[179] Vgl. Bach (2000), S. 102.

Zeichenketten folgendermaßen angegeben werden müssen concat(Zeichenkette1, Trennzeichen1, Zeichenkette2, Trennzeichen2, …).[180]
Mit contains(Zeichenkette1, Zeichenkette2) wird überprüft, ob die erste Zeichenkette die zweite Zeichenkette enthält. Wenn dies stimmt, ist diese Aussage auch das Ergebnis der Funktion.[181] Das bedeutet, contains( ) liefert als Ergebnis eine Aussage über den Wahrheitsgehalt seines Inhalts.
Um Werte einer Zeichenkette vor dem Auftreten einer anderen Zeichenkette ausgeben zu lassen, kann die Funktion substring-before(Zeichenkette1, Zeichenkette2) genutzt werden, wobei die erste Zeichenkette der Text ist, der auf die Zweite durchsucht wird. Wird die zweite Zeichenkette gefunden, wird der Inhalt aus Zeichenkette1 vor dem ersten Auftreten von Zeichenkette2 zum Ergebnis der Funktion.[182]
Auf die gleiche Weise funktioniert substring-after(Zeichenkette1, Zeichenkette2), nur dass der Rest von Zeichenkette1 nach dem ersten Auftreten von Zeichenkette2 in das Ergebnis der Funktion übernommen wird.[183]
Um genauer definieren zu können, welche Zeichen einer Zeichenkette als Ergebnis erhalten werden sollen, kann man die Funktion substring(Zeichenkette, Zahl1, Zahl2) benutzen. Dabei gibt der erste Bestandteil wieder an, welche Zeichenkette bearbeitet werden soll. Die erste Zahl gibt die Position des Zeichens an, ab dem die Zeichenkette ausgegeben werden soll und die zweite Zahl gibt an, wie viele Zeichen für das Ergebnis verwendet werden sollen.[184] Wird die zweite Zahl nicht angegeben, wird die gesamte Zeichenkette ab der Stelle, die in Zahl1 genannt wird, zum Ergebnis der Funktion.[185] Beispielsweise würde substring(‘Testzeichenkette‘, 4, 8) das Ergebnis „tzeichen“ hervorrufen.
Mit translate(Zeichenkette, Zeichen1, Zeichen2) werden alle Zeichen aus der Zeichenkette im ersten Bestandteil der Funktion, die in Zeichen1 stehen, mit den

[180] Vgl. Bach (2000), S. 104.
[181] Vgl. Bach (2000), S. 104-105.
[182] Vgl. Bach (2000), S. 105.
[183] Vgl. Bach (2000), S. 105.
[184] Vgl. Bach (2000), S. 105.
[185] Vgl. Lenz (2006), S. 112-113.

Zeichen aus Zeichen2 ersetzt.[186] Dabei entsprechen sich die Zeichen mit den gleichen Positionen in Zeichen1 und Zeichen2. Wenn nicht gleich viele Zeichen in Zeichen1 und Zeichen2 vorhanden sind, werden die überzähligen Zeichen aus Zeichen1 aus der zu bearbeitenden Zeichenkette komplett gelöscht. [187] Ein Beispiel dafür wäre translate('Testzeichenkette','ezc','a'), was als Ergebnis „Tastaihankatta" erzeugen würde.

Insgesamt können die Ausdrücke, die mit XPath formuliert werden können, also ziemlich komplex werden, wodurch die Wahrscheinlichkeit in so einem Ausdruck einen Fehler zu machen relativ hoch ist, sodass dann als Ergebnis keine passenden Knoten gefunden werden können.[188]

Die Ausdrücke können Pfade, Zeichenketten, Zahlen und/oder einen Funktionsaufruf enthalten. Aber auch eine Variable bzw. ein Parameter kann Inhalt eines XPath-Ausdrucks sein. Verknüpft werden können diese Inhalte mit Vergleichsoperatoren, Booleschen Operatoren und Klammern.[189]

Das Ergebnis eines XPath-Ausdrucks kann eine Knotenmenge, eine Aussage über die Richtigkeit des XPath-Ausdrucks, in Form von Booleschem „wahr" oder „falsch", eine Fließkommazahl oder eine Zeichenkette sein.[190]

Erweitert wird die Menge an möglichen Ergebnissen mit XSLT durch das sogenannte Ergebnisbaumfragment, das einen Teil von Werten, die direkt in das Ausgabedokument überführt werden sollen, gemischt mit XSLT-Funktionen enthält.[191] In XPath wird das Ergebnisbaumfragment dann meist über eine Variable oder einen Parameter weitergegeben und stellt eine Knotenmenge dar, die sich so verhält, als hätte sie ein eigenes Wurzelelement.[192]

Grundsätzlich findet man XPath-Ausdrücke in XSLT in den Attributen von Anweisungen.[193]

---

[186] Vgl. Bach (2000), S. 106.

[187] Vgl. Lenz (2006), S. 114.

[188] Vgl. Montero Pineda (2004), S. 41.

[189] Vgl. Bach (2000), S. 89 und Kränzler (2002), S. 208.

[190] Vgl. Bach (2000), S. 67.

[191] Vgl. Lenz (2006), S. 11.

[192] Vgl. Lenz (2006), S. 12.

[193] Vgl. Bach (2000), S. 66.

3.2. XSLT

3.2.1. Definition

XSLT ist in seiner ersten Version am 16. November 1999 als eine Empfehlung vom W3C veröffentlicht worden[194] und dient dazu XML-Dokumente in eine andere Struktur oder in ein anderes Format umzuwandeln. Neben XML- können auch HTML- (Hypertext Markup Language), XHTML- (Extensible Hypertext Markup Language) und Textausgaben erzeugt werden.[195]
Außerdem können mit XSLT auch Abfragen von Daten aus XML-Dokumenten gemacht werden, bei denen Informationen gefiltert bzw. umstrukturiert werden.[196]

XSLT gehört zu den XML-Vokabularen[197] und ebenfalls zu den XML-Standards.[198] Das heißt, die Stylesheets, in denen die Anweisungen formuliert werden, um die XML-Dokumente zu bearbeiten, sind selbst auch XML-Dokumente. [199]
Die Anweisungen, die ein XSLT-Stylesheet enthält, sind Regeln, die definieren, wie ein Dokument bearbeitet werden soll.[200] Stylesheets geben einem Prozessor also vor, was er mit dem Eingabedokument machen soll, überlassen ihm jedoch die Art, wie er die Regeln technisch umsetzt.[201]
Indem der Prozessor die Regeln aus dem Stylesheet auf ein Eingabedokument anwendet, wird ein Ausgabedokument hergestellt, das Inhalte aus dem Eingabedokument in der Form enthält, die in dem Stylesheet definiert wurde.[202]
Ziel bei der Erstellung eines Stylesheets ist es also immer, Regeln festzulegen, die es ermöglichen die Informationen aus dem Eingabedokument, die man verwenden

---

[194] Vgl. Kränzler (2002), S. 225.

[195] Vgl. Vonhoegen (2009), S. 241.

[196] Vgl. Li; Mani; Rundensteiner (2008), S. 524.

[197] Vgl. Kränzler (2002), S. 223.

[198] Vgl. Garcia-Sanchez; et al. (2008), S. 1021.

[199] Vgl. Jackenkroll (2003), S. 30.

[200] Vgl. Kränzler (2002), S. 223.

[201] Vgl. Vonhoegen (2009), S. 245.

[202] Vgl. Kränzler (2002), S. 226.

möchte, in ein Format und eine Struktur zu bringen, die schon vor der Erstellung des Stylesheets für die Ausgabe festgelegt werden.[203]
Wichtig zu wissen ist, dass nur wohlgeformte XML-Dateien mit XSLT bearbeitet werden können. Dies ist auch die einzige Bedingung, die an die Eingabedokumente gestellt wird. [204]
Außerdem kann nur mit den Auszeichnungselementen in XML-Dokumenten, nicht aber mit den DTDs (Document Type Definition) oder Schemata gearbeitet werden,[205] die zur Festlegung und Kontrolle der Einhaltung einer bestimmten Struktur dienen.

Bei der Umwandlung eines XML-Dokuments in eine andere Struktur oder ein anderes Format, wobei auch von Transformation gesprochen wird, können alle Informationen aus der Eingabedatei oder nur ein Teil von ihnen verwendet werden.[206] Das wiederum bedeutet, dass Informationen aus der Eingabedatei verloren gehen und/oder neue hinzugefügt werden, sodass aus einer transformierten Datei in der Regel nicht so einfach die ursprüngliche Datei wiederhergestellt werden kann.[207]

Neben einer Transformation, bei der eine neue Datei erzeugt wird, kann ein XSLT-Stylesheet auch mit der Anweisung

```
<?xml-stylesheet href="unterverzeichnis/stylesheet.xsl" type="text/xsl"?>
```

direkt in eine XML-Datei eingebunden werden. Hierdurch wird z.B. im Browser die Ansicht, die vom Stylesheet vorgeschrieben wird, bewirkt.[208]
Bedeutsam für das Funktionieren des Stylesheets ist es einerseits, dass das Wurzelelement des Stylesheets <xsl:stylesheet> oder <xsl:transform> ist. Das Element <xsl:transform> kann nur als Wurzelelement des Stylesheets verwendet werden, wenn das Stylesheet lediglich für Transformationen verwendet wird.[209]

---

[203] Vgl. Garcia-Sanchez; et al. (2008), S. 1022.

[204] Vgl. Vonhoegen (2009), S. 243.

[205] Vgl. Vonhoegen (2009), S. 243.

[206] Vgl. Bach (2000), S. 117.

[207] Vgl. Bach (2000), S. 118.

[208] Vgl. Vonhoegen (2009), S. 247.

[209] Vgl. Vonhoegen (2009), S. 248.

Andererseits muss immer der Namensraum für die XSLT-Anweisungen (xmlns:xsl="http://www.w3.org/1999/XSL/Transform") im Wurzelelement angegeben werden, damit die Anweisungen von den anderen Informationen im Stylesheet abgegrenzt werden können. Andernfalls würden die Anweisungen so, wie sie im Stylesheet stehen, als Elemente in die Ausgabedatei übernommen.[210]

3.2.2. Funktionen

Grundsätzlich benötigt man immer einen Prozessor, um ein Dokument mit den Anweisungen eines Stylesheets zu transformieren.[211]
Der Prozessor liest zu Beginn einer Transformation die Eingabedatei und das Stylesheet ein, d.h. er speichert sie zwischen. Dabei bildet er sozusagen die Baumstrukturen von beiden Dateien nach, um dann einen neuen Ergebnisbaum mittels der Ausgangsdaten und der Verarbeitungsanweisungen zu erzeugen und schließlich, nach dem Umsetzen aller Anweisungen, in dem gewünschten Format zu speichern.[212]
Mit den Stylesheets kann man die Eingabedateien umstrukturieren, Knoten umbenennen, sich nur bestimmte Knoten ausgeben lassen, zusätzliche Informationen, z.B. auch aus anderen XML-Dokumenten, hinzufügen, einfache Rechnungen durchführen lassen und Textinhalte verändern.[213]
Damit man aber mittels XPath auf die Knoten zugreifen kann, ist es wichtig die Namensräume der Eingabe- und Ausgabedateien im Stylesheet zu deklarieren, da sonst im Fall der Eingabedatei nicht auf die Elemente mit Namensraum und ihre Inhalte zugegriffen werden kann. Allerdings können die Präfixe der Namensräume, d.h. das Kürzel über das signalisiert wird, zu welchem Namensraum ein Element gehört, auch umbenannt werden.[214]
Um zu den konkreten Funktionen von XSLT-Stylesheets zurückzukommen, soll als Erstes festgestellt werden, dass eine Umwandlung immer mit einer eingebauten

---

[210] Vgl. Vonhoegen (2009), S. 245.

[211] Vgl. Jackenkroll (2003), S. 31.

[212] Vgl. Vonhoegen (2009), S. 253.

[213] Vgl. Cole; et al. (2001), S. 216.

[214] Vgl. Bach (2000), S. 136.

Funktion startet, die den Prozessor dazu bringt, ausgehend vom Wurzelknoten eine Verarbeitung für das Eingabedokument zu starten.[215] Hierbei sollte beachtet werden, dass alle Anweisungen, die dem Prozessor innerhalb des Stylesheets gegeben werden, so ausgeführt werden, dass immer die nächste definierte Anweisung bearbeitet wird. Das heißt zum Beispiel, dass zunächst ein Wurzelelement für das Ausgabedokument geöffnet wird, dann eine Anweisung für einen anderen Knoten aufgerufen und verarbeitet wird und wenn in der Verarbeitungsanweisung für diesen Knoten Anweisungen für weitere Knoten gegeben werden, diese wiederum als Nächstes bearbeitet werden. Erst ganz zum Schluss, wenn alle Verarbeitungen durchgeführt wurden, wird das Wurzelelement wieder geschlossen.[216]

Einer der bedeutendsten Bestandteile eines Stylesheets ist nach dem Wurzelelement <xsl:stylesheet> die Anweisung <xsl:template>. Wenn im Eingabedokument ein Element gefunden wird, das zu dem XPath-Ausdruck im match-Attribut von <xsl:template> passt, wird dieses Element mit den Anweisungen, die sich in <xsl:template> befinden, weiterverarbeitet.[217] Ein <xsl:template> besteht dabei aus zwei Bestandteilen, nämlich dem Pattern, d.h. der Angabe der zu suchenden Elemente, die in dem match-Attribut steht, und dem Template, das heißt den Verarbeitungsanweisungen im Element.[218]
Das Template selbst kann auch aus zwei verschiedenen Komponenten bestehen. Die eine Komponente sind die XSLT-Anweisungen. Die Dinge, die genauso, wie sie in dem Stylesheet angegeben sind, in die Ausgabedatei übernommen werden sollen, wie Textinhalte oder Elemente, werden als literale Ergebniselemente bezeichnet und sind die andere Komponente.[219]
Neben dem match-Attribut kann <xsl:template> auch über ein Attribut namens „mode“ verfügen, das es ermöglicht, für eine Elementmenge zwei unterschiedliche Verarbeitungen vorzunehmen. Sind zwei Verarbeitungen definiert, das mode-

[215] Vgl. Lenz (2006), S. 31.

[216] Vgl. Vonhoegen (2009), S. 254.

[217] Vgl. Montero Pineda (2004), S. 22.

[218] Vgl. Montero Pineda (2004), S. 23-24.

[219] Vgl. Vonhoegen (2009), S. 250-251.

Attribut aber nicht vorhanden, wird nur eine Verarbeitung durchgeführt, nämlich das spezialisiertere, das mit höherer Priorität versehene, [220] oder wenn beide Templates gleich spezialisiert sind, das letzte Template.
Weiterhin ist es möglich außer über das Muster, das mit dem Attribut „match“ angegeben wird, auch ein Template mit einem Namen über das Attribut „name“ aufzurufen. In diesem Fall wird das Template knotenunabhängig und wird anstatt in dem Kontext, der durch das match-Attribut erzeugt werden würde, im Kontext des Knotens ausgeführt, in dessen Template das benannte Template aufgerufen wurde.[221] Dieses Template benötigt also seinen Aufruf <xsl:call-template name=“ “/> im Stylesheet[222] und eignet sich besonders zur Weitergabe von Parametern und Variablen.[223]
Der Aufruf eines Templates ist ebenfalls sehr wichtig für die Funktionsweise eines Stylesheets, da so bestimmt wird, welche Verarbeitung als Nächstes stattfinden soll. So ein Aufruf ist ein Nachfahre eines xsl:template-Elements.
Ein knotenabhängiges Template, d.h. eins, das mit einem match-Attribut bestimmt, welche Knoten verarbeitet werden sollen, wird mit einem <xsl:apply-templates> aufgerufen. Dabei steht in der Regel in einem select-Attribut für welche Knotenmenge, ausgehend vom aktuellen Knoten, ein Template gesucht werden soll.[224] Hat <xsl:apply-templates> kein Attribut werden Templates für untergeordnete Elemente gesucht und wenn keine gefunden werden, die Textinhalte des aktuellen Knotens und seiner untergeordneten Element- und Attributknoten ausgegeben.[225]
Hierfür wird von sogenannten eingebauten Templates, die für alle Knoten in jedem Stylesheet immer definiert sind, Gebrauch gemacht, damit bei <xsl:apply-templates> immer eine Ausgabe erfolgt. So wird für alle Elemente inklusive des Wurzelknotens nach Templates für Kindelemente gesucht und wenn Templates mit modes im Stylesheet vorhanden sind, nach Kindelementen mit den modes. Wenn keine

[220] Vgl. Bach (2000), S. 127.
[221] Vgl. Vonhoegen (2009), S. 261.
[222] Vgl. Montero Pineda (2004), S. 26.
[223] Vgl. Vonhoegen (2009), S. 261.
[224] Vgl. Lenz (2006), S. 33.
[225] Vgl. Montero Pineda (2004), S. 22.

Templates für direkte Kindelemente gefunden werden, wird für Enkelelemente nach Templates gesucht und dieser Vorgang wird solange wiederholt, bis Templates gefunden oder keine Elemente mehr durchlaufen werden können.[226]
Für alle Attribut- und Textknoten wird in einem eingebauten Template festgelegt, dass ihr Inhalt ausgegeben werden soll. Bei Kommentaren und Verarbeitungsanweisungen besteht das eingebaute Template aus der Anweisung keine Ausgabe zu machen.[227]
Diese eingebauten Templates werden außer Kraft gesetzt, wenn es Templates für Knoten gibt, die über das <xsl:apply-templates> erreicht werden können.

Mit dem Element <xsl:output>, das direkt unter das Wurzelelement des Stylesheets geschrieben wird, kann man die Gestaltung der Ausgabedatei beeinflussen.

```
<xsl:output method="xml" encoding="UTF-8" indent="yes"/>
```

Dies wäre ein typisches Beispiel für ein xsl:output-Element, bei dem „method“ angibt, welche Art von Ausgabedatei erzeugt werden soll, „encoding“ die Zeichencodierung bestimmt und „indent="yes"“ festlegt, dass die Elemente im Ausgabedokument eingerückt werden sollen.[228] Zusätzlich können z.B. die DTD oder das Schema der Ausgabedatei angegeben werden.[229]
Um Werte wie Namen, Inhalte oder Attribute von Knoten, aus der Eingabedatei in die Ausgabedatei überführen zu können, wird in der Regel in einem Template <xsl:value-of> genutzt, bei dem im select-Attribut festgelegt wird, aus welchem Knoten des Eingabedokuments ein Wert verwendet werden soll. [230] Mit

```
<xsl:value-of select=“element“/>
```

wird beispielsweise der Inhalt des Kindelements <element> des aktuellen Knotens ausgegeben.
Mit XSLT können auch zwei Arten von Bedingungen realisiert werden, nämlich einfache Bedingungen und „Entweder-Oder-Bedingungen“.

---

[226] Vgl. Lenz (2006), S. 40-41.

[227] Vgl. Lenz (2006), S. 40-41.

[228] Vgl. Bach (2000), S. 141-142.

[229] Vgl. Montero Pineda (2004), S. 23.

[230] Vgl. Montero Pineda (2004), S. 26.

Die einfachen Bedingungen werden mit <xsl:if> umgesetzt. Immer, wenn die Bedingung, die in dem test-Attribut mit XPath angegeben wird, erfüllt werden kann, wird das durchgeführt, was in dem xsl:if-Element steht.[231]
Bei den „Entweder-Oder-Bedingungen“ werden Alternativen, die getestet werden sollen, definiert. Sie werden mit folgender Gruppierung realisiert:

```
<xsl:choose>
        <xsl:when test=“ “>  <!--Verarbeitungsanweisungen --> </xsl:when>
        <xsl:otherwise>      <!--Verarbeitungsanweisungen --> </xsl:otherwise>
</xsl:choose>
```

Dabei können beliebig viele when-Bedingungen verwendet werden, die genauso funktionieren, wie die xsl:if-Bedingungen. Es ist aber auf die Reihenfolge der Bedingungen zu achten, da immer die Verarbeitung durchgeführt wird, bei der als erstes die Bedingung erfüllt werden kann, egal ob das auch bei anderen Bedingungen so wäre.[232] Aus diesem Grund kann empfohlen werden, zunächst immer die spezifischsten bzw. bevorzugten Bedingungen zu definieren.
Das xsl:otherwise-Element kann verwendet werden, um eine alternative Verarbeitung anzugeben, die umgesetzt werden soll, wenn keine Bedingung erfüllt werden kann. Dieses Element ist jedoch optional.[233]
Eine weitere Funktion von XSLT, die für die vorliegende Arbeit von Bedeutung ist, ist die Funktion <xsl:sort>, mit der Knotenmengen sortiert werden können. Diese Funktion kann jedoch nur verwendet werden, wenn sie in einem <xsl:apply-templates> oder einem <xsl:for-each> steht, in dem die Knotenmenge selektiert wird, die sortiert werden soll. Ein Beispiel für <xsl:sort> ist:

```
<xsl:sort select=" " data-type="text" order="ascending"/>
```

Der Knoten anhand dessen eine Sortierung vorgenommen werden soll, steht in dem select-Attribut von <xsl:sort>. Mit dem Attribut „order“ kann man eine Sortierreihenfolge angeben, die in diesem Fall aufsteigend ist. Für eine absteigende Sortierung wäre der Wert „descending“ nötig. Das optionale Attribut „data-type“ gibt die Art der Information des Sortierkriteriums an und kann neben „text“ auch vom Typ Knotenname (qname) oder Zahl (number) sein.[234]

---

[231] Vgl. Montero Pineda (2004), S. 30-31.

[232] Vgl. Vonhoegen (2009), S. 271.

[233]Vgl. Vonhoegen (2009), S. 271.

[234] Vgl. Lenz (2006), S. 86-87.

Das bereits erwähnte Element <xsl:for-each> sorgt dafür, dass jeder Knoten aus einer Knotenmenge, die über ein select-Attribut angegeben wird, einzeln mit den Anweisungen innerhalb des Elements verarbeitet wird.[235] Das Element funktioniert also ähnlich wie die Kombination aus dem Aufruf eines Templates und dem Template selbst, wird aber innerhalb eines Templates als Verarbeitungsanweisung verwendet.

Die Funktion <xsl:param> dient dazu, häufig genutzte Ausdrücke nicht immer wieder eingeben zu müssen, v.a. wenn sie etwas länger sind,[236] oder dazu, einen Teil des Ausgabedokuments zwischenzuspeichern.[237] Ebenfalls kann durch die Verwendung eines Parameters der Kontext eines Elements beispielsweise in einem Template für einen anderen Knoten beibehalten werden, da sich sonst der Kontext an das neue Template anpassen würde.[238] Der Knoten, der mittels des Parameters weitergegeben wurde, gehört dann nämlich zum Kontext des aktuellen Knotens, d.h. dem Knoten, auf den das Template angewendet wird.[239]

Parameter können global oder lokal definiert werden. Bei einem globalen Parameter handelt es sich um einen Parameter, der direkt in der Ebene unterhalb des Wurzelelements <xsl:stylesheet> definiert wird, und dann in allen Templates verwendet werden kann.[240]

Ist ein Parameter lokal, wird er von einem Template zu einem anderen weitergegeben und muss innerhalb eines <xsl:apply-templates> oder <xsl:call-template> mit

<xsl:with-param name=“ “ select=“ “/>

definiert werden. Um den Parameter eindeutig identifizierbar zu machen, wird ein Name vergeben. Der Wert des Parameters kann entweder in das select-Attribut eingetragen werden, oder der Inhalt des Elements sein. Beide Optionen können jedoch nicht zusammen verwendet werden. [241]

---

[235] Vgl. Lenz (2006), S. 67.

[236] Vgl. Vonhoegen (2009), S. 280.

[237] Vgl. Vonhoegen (2009), S. 284.

[238] Vgl. Vonhoegen (2009), S. 281.

[239] Vgl. Vonhoegen (2009), S. 278.

[240] Vgl. Montero Pineda (2004), S. 39.

[241] Vgl. Vonhoegen (2009), S. 277.

Damit ein Parameter in einem Template verwendet werden kann, muss als erstes Kindelement des xsl:template-Elements die Definition des Parameters in Form von

<xsl:param name=“ “ select=“ “/>

erfolgen, wobei das select-Attribut optional ist[242] und ein Wert, der im Templateaufruf definiert und weitergegeben wurde, den an dieser Stelle angegeben Wert überschreiben würde.

Um den Parameter nun innerhalb einer Anweisung zu verwenden, wird ein $-Zeichen vor den Namen geschrieben.[243] So würde etwa mit

<xsl:value-of select=“$parameter“/>

der Inhalt des Parameters namens „parameter“ ausgegeben.

Eine andere Funktion, die für das Erzeugen von neuen XML-Dateien hilfreich ist, ist <xsl:copy-of select=“ “>. Bei dieser Funktion, wird der Knoten im select-Attribut mit seiner ganzen Struktur aus dem Eingabedokument, d.h. mit allen Namensräumen, Attributen und Kindelementen, in die Ausgabedatei kopiert.[244]

Mit <xsl:text> kann man Textknoten erzeugen. Diese Funktion ist besonders dann nützlich, wenn mehrere Leerzeichen oder Sonderzeichen wie das Et-Zeichen (&) als Teil eines codierten Sonderzeichens in ihrer angegebenen Form beibehalten werden sollen und nicht durch nur *ein* Leerzeichen oder ihre codierte Form ersetzt werden sollen. Zu diesem Zweck wird dem xsl:text-Element ein Attribut mit Wert „disable-output-escaping="yes"“ hinzugefügt.[245]

<xsl:text disable-output-escaping="yes" >test für leerzeichen</text>

Dieses Beispiel würde einen Textknoten mit dem Inhalt „test für leerzeichen“ erzeugen, anstelle von „test für leerzeichen“.

Um Werte aus einem anderen XML-Dokument als dem, das mit dem Prozessor gerade verarbeitet wird, zu nutzen, kann die Funktion document( ) verwendet werden. Innerhalb der Klammern muss dann der Pfad zu der entsprechenden Datei angegeben werden. Diese Funktion kann beispielsweise Inhalt eines match- bzw.

---

[242] Vgl. Vonhoegen (2009), S. 277.

[243] Vgl. Montero Pineda (2004), S. 39.

[244] Vgl. Montero Pineda (2004), S. 39.

[245] Vgl. Bach (2000), S. 153.

select-Attributs eines xsl:template-Elements bzw. xsl:apply-templates-Elements sein.[246] Ein Beispiel wäre:

```
<xsl:apply-templates select="document('../publishers.xml')/publisherslist">
```

Hierbei würde ein Template für das Wurzelelement <publisherslist> des Dokuments publishers.xml aufgerufen, das sich in dem übergeordneten Ordner des Ordners befindet, in dem das Stylesheet ist.

Eine weitere Funktion von XSLT ist <xsl:key>. Mit dieser Funktion können Referenzen aufgelöst bzw. Schlüssel definiert werden. Ein Schlüssel ist ein Merkmal eines Elements, das in dem Eingabedokument für den Inhalt eines anderen Knoten steht.[247] Verglichen werden kann dies mit dem System von Fußnoten, bei dem die hochgestellte Zahl auch für etwas anderes als den Zahlenwert steht.

Bei der Definition des Schlüssels, die direkt unter dem Wurzelelement des Stylesheets erfolgen muss, hat das xsl:key-Element drei Attribute. Das erste Attribut, „name“, enthält den Namen des Schlüssels. Das zweite Attribut, ein match-Attribut, beinhaltet die Knotenmenge, die die Referenz enthält. [248] Das heißt, um bei dem Beispiel mit der Fußnote zu bleiben, würde das match-Attribut den Fußnotentext zu einer hochgestellten Zahl enthalten.

Mit dem dritten Attribut namens „use“, wird das Merkmal beschrieben, an dem die Referenz identifiziert werden kann,[249] was in dem Beispiel die hochgestellte Zahl in der Fußnote wäre. Eine Schlüsseldefinition mit den Fußnoten könnte wie folgt aussehen:

```
<xsl:key name="fussnotenschluessel" match="fussnotentext" use="@zahl"/>
```

Dabei wäre <fussnotentext>, das Element, das den Text für die Referenz enthält und sein Attribut „zahl“ das Merkmal um die Referenz aufzulösen.

Wenn der Schlüssel in einem Template verwendet werden soll, könnte der Aufruf so aussehen:

```
<xsl:template match="satz">
    <p><xsl:value-of select="."/>
        (<xsl:value-of select="key(fussnotenschluessel, hochgestellte-zahl)"/>)
    </p>
</xsl:template>
```

[246] Vgl. Vonhoegen (2009), S. 293.

[247] Vgl. Bach (2000), S. 176.

[248] Vgl. Bach (2000), S. 178.

[249] Vgl. Bach (2000), S. 178.

Hierbei würde auf das Element <satz>, d.h. auf den Satz, auf den sich die Fußnote bezieht, das Template angewendet. Zunächst würde der Inhalt des Satzes in das Element <p> geschrieben. Dann würde eine Klammer geöffnet und danach würde für das Kindelement <hochgestellte-zahl> vom aktuellen Knoten <satz> mittels des Schlüssels nach einem fussnotentext-Element, das ein passendes Attribut „zahl“ besitzt, irgendwo im Eingabedokument gesucht. Schließlich würde die Klammer wieder geschlossen, nachdem der Inhalt des gefundenen fussnotentext-Elements ausgegeben worden wäre.
Das Attribut „zahl“ und das Element <hochgestellte-zahl> müssen über den gleichen Inhalt verfügen, damit eine Ausgabe erfolgen kann.
Nach dem Anwenden der key-Funktion und dem Auffinden von übereinstimmenden Inhalten, hätte man also einen Satz, bei dem die Quellenangabe früher als Fußnote angegeben war, zu einem Satz, bei dem sie in Klammern hinter dem jeweiligen Text steht, umgewandelt.

3.3. Apache Ant

Apache Ant ist ein Java basiertes Werkzeug, das entwickelt wurde, da die anderen sogenannten „Build-Werkzeuge“, die dazu dienen Anwendungen zu erstellen, Einschränkungen haben, die der Entwickler von Ant nicht akzeptieren wollte.[250] Es gehört zur Apache Software Foundation.[251]
Um die Funktionsweise von XSLT, die gerade beschrieben wurde, noch besser zu verstehen, ist es nötig zu wissen, dass man nicht allein mit einem Stylesheet eine Umwandlung durchführen kann, sondern einen Prozessor benötigt, der die Anweisungen aus dem Stylesheet umsetzt. Zusätzlich zu den Anweisungen, die man einem Prozessor über das Stylesheet gibt, können auch Anweisungen in der eigenen Sprache des Prozessors gegeben werden.[252]
Bei Ant werden diese Anweisungen in eine XML-Datei geschrieben.
Ant wird demnach durch die Anweisungen, die in sogenannten <target>s in dem XML-Dokument definiert werden, gesteuert. Außerdem verwendet es Java-Klassen.

---

[250] Vgl. Bailliez; et al. (2011), [http://ant.apache.org/manual/intro.html].

[251] Vgl. Apache Software Foundation (2011).

[252] Vgl. Cole; et al. (2001), S. 216.

Durch beides wird die Software plattformunabhängig.[253] Allerdings muss die Kommandozeile zum Starten und Durchführen der Anweisungen verwendet werden.[254]

Ant dient dazu Prozesse zum Herstellen bzw. Erzeugen von Anwendungen zu formulieren und wurde hauptsächlich für Java-Anwendungen konzipiert. Zu diesem Zweck enthält es auch Komponenten, um Java-Anwendungen zu testen und auszuführen. Allerdings können auch andere Prozesse mit Ant gestaltet werden, wie zum Beispiel mit C++ oder, wie für diese Arbeit benötigt, mit XSLT.[255]

Wichtig beim Arbeiten mit Ant ist, dass immer ein Verzeichnis benötigt wird, in dem Ant arbeiten kann und von dem aus auch andere Ordner angesteuert werden können, also eine Art Basisverzeichnis. In diesem Verzeichnis befinden sich zum Beispiel häufig auch die XML-Dokumente mit den Verarbeitungsanweisungen.

So ein XML-Dokument, das auch als „build-file“ bzw. im Verlauf der vorliegenden Arbeit als Build-Datei bezeichnet wird, enthält als Wurzelelement ein Projekt <project>, in dem mindestens ein <target> bzw. Ziel vorhanden ist, das wiederum Tasks, d.h. Aufgaben, beinhaltet. Die <target>s besitzen ein id-Attribut, über das sie eindeutig identifizierbar sind und beispielweise von einem anderen <target> aus aufgerufen werden können. Ein Task hingegen ist immer etwas, das ausgeführt werden kann.[256]

Beispielsweise können mit Ant Löschbefehle formuliert werden, mit denen eine oder mehrere Dateien oder ein Verzeichnis gelöscht werden kann.[257] Ein Beispiel, um eine Datei zu löschen, die im file-Attribut benannt wird, könnte so aussehen:

```
<delete file=“ “/>
```

Um Dateien automatisch aus dem Internet herunterzuladen, kann der Task

```
<get src=“ “ dest=“ “/>
```

verwendet werden. Hierbei ist der Wert des Attributes „src“ die Internetadresse, von der die Datei heruntergeladen werden soll und das obligatorische Attribut „dest“

---

[253] Vgl. Bailliez; et al. (2011), [http://ant.apache.org/manual/intro.html].

[254] Vgl. Apache Software Foundation (2011).

[255] Vgl. Apache Software Foundation (2011).

[256] Vgl. Bailliez; et al. (2011), [http://ant.apache.org/manual/using.html#buildfile].

[257] Vgl. Bailliez; et al. (2011), [http://ant.apache.org/manual/Tasks/delete.html].

enthält eine Datei oder ein Verzeichnis, in welche die heruntergeladene Datei gespeichert wird.[258]

Der Task <echo> ermöglicht es Texte bzw. Inhalte in eine Datei zu speichern. Es gibt zwei verschiedene Möglichkeiten, wie dies angegeben werden kann. Bei beiden Varianten bestimmt das file-Attribut von <echo> die Datei, die mit Inhalt gefüllt werden soll. Bei der ersten Variante steht der Inhalt, der in eine Datei geschrieben werden soll, in einem message-Attribut und bei der zweiten Variante als Inhalt im echo-Element. Um weitere Inhalte in die gleiche Datei einzufügen und die Datei dabei nicht zu überschreiben, kann das Attribut „append“ mit dem Wert „yes“ hinzugefügt werden.[259]

Damit mit Ant eine XSL-Transformation vorgenommen werden kann, muss der Task <xslt> verwendet werden.

<xslt in=“ “ out=“ “ style=” ” failOnError=“false“/>

In dem Schema wird im in-Attribut die Eingabedatei, im out-Attribut die Ausgabedatei und im style-Attribut das Stylesheet für die Transformation angegeben. Das Attribut „failOnError“, das nicht angegeben werden muss, bestimmt das Verhalten des Prozessors, wenn eine Eingabedatei nicht verarbeitet werden kann. Normalerweise wird in diesem Fall der Prozess abgebrochen. Mit dem Wert „false“ in dem Attribut kann eine nicht durchführbare Umwandlung jedoch übersprungen werden.[260]

Auch in Ant können Variablen genutzt werden, die hier jedoch als Properties bezeichnet werden. Sie bestehen aus einem Namen und einem Wert, der über die Angabe {$*Name*} verwertet werden kann.[261]

Mit dem Task <loadfile> kann man sogar den Inhalt einer ganzen Datei zu einer Property machen.

<loadfile property=“eigenschaft “ srcFile=“ “/>

Dieses Element würde zum Beispiel dazu führen, dass die Datei, die in „srcFile“ angegeben ist, zu der Property mit dem Namen „eigenschaft“ wird.

---

[258] Vgl. Bailliez; et al. (2011), [http://ant.apache.org/manual/Tasks/get.html].

[259] Vgl. Bailliez; et al. (2011), [http://ant.apache.org/manual/Tasks/echo.html].

[260] Vgl. Bailliez; et al. (2011), [http://ant.apache.org/manual/Tasks/style.html].

[261] Vgl. Bailliez; et al. (2011), [http://ant.apache.org/manual/using.html#buildfile].

Der Inhalt der Datei kann auch mit bestimmten Filtern, die an dieser Stelle nicht beschrieben werden sollen, da sie in dieser Arbeit keine Anwendung finden, auf Bestandteile eingeschränkt werden, denn sonst wird der ganze Inhalt der Datei in einer langen Zeichenkette zum Inhalt der Property.[262]

Mit dem Task <tstamp> kann das aktuelle Datum zu einer Variablen gemacht werden. Diese Funktion wird vor allem zum Generieren von Dateinamen, die das aktuelle Datum enthalten, verwendet.

```
<tstamp><format property="datum" pattern="dd-MM-yyyy"/></tstamp>
```

Diese Angabe würde bewirken, dass das aktuelle Datum z.B. als 25-11-2011 als Variable mit dem Namen „datum" erzeugt würde.[263]

Wie bereits gesagt, können in einem <target> auch andere <target>s aufgerufen werden. Dies wird durch den Task <foreach> zum Beispiel für mehrere Dateien, die alle hintereinander mit den gleichen Anweisungen verarbeitet werden sollen, ermöglicht. Dabei wird ein param-Attribut mit einem Namen für den Parameter, den die Dateien jeweils darstellen sollen, in das foreach-Element eingefügt und eine Beschreibung innerhalb des foreach-Elements für diese Dateien abgegeben. Außerdem muss natürlich in dem Attribut „target" der Name des <target>s, das auf die Dateien angewendet werden soll, genannt werden.

Beispielsweise könnte das folgendermaßen aussehen:

```
<foreach target="umwandlung-journaltoc" param="File">
        <path>
                <fileset dir="." casesensitive="no">
                        <include name="feeds-neu/*"/>
                        <exclude name="feeds-neu/*-ausgabe.xml"/>
                 </fileset>
        </path>
</foreach>
```

Mit Hilfe der Funktion <fileset> kann eine Menge von Dateien bestimmt werden, die aus einem Verzeichnis stammen oder alle zu einem Muster passen. Das dir-Attribut gibt dabei an, in welchem Verzeichnis nach den Dateien gesucht werden soll.[264] Im Fall des Beispiels wird dies aber erst in dem include- bzw. exclude-Element spezifiziert.

---

[262] Vgl. Bailliez; et al. (2011), [http://ant.apache.org/manual/Tasks/loadfile.html].

[263] Vgl. Bailliez; et al. (2011), [http://ant.apache.org/manual/Tasks/tstamp.html].

[264] Vgl. Bailliez; et al. (2011), [http://ant.apache.org/manual/Types/fileset].

In beiden Fällen wird mit einem Muster gearbeitet, wobei im name-Attribut des include-Elements angegeben wird, dass auf alle Dateien aus dem Ordner feeds-neu das <target> „umwandlung-journaltoc“ angewendet werden soll. Durch das exclude-Element hingegen wird diese Menge an Dateien dadurch eingeschränkt, dass alle Dateien aus dem Verzeichnis, die mit „-ausgabe.xml“ enden, nicht einbezogen werden sollen. Das Sternchen fungiert an dieser Stelle als Platzhalter für beliebig viele Zeichen.

Nach der vorausgegangenen Beschreibung der Instrumente und deren Funktionen, die für den Prozess zur Anpassung der Daten an das von ALBERT benötigte Format genutzt wurden, soll in den nachfolgenden Kapiteln der Prozess selbst beschrieben werden.

4. Prozess zur Anpassung von Daten für ALBERT

Der Prozess, mit dem lizensierte (gelbe) und freie (grüne) Zeitschriften aus der Elektronischen Zeitschriftenbibliothek, sowie die Zeitschrifteninhaltsverzeichnisse von Zeitschriften, die mit dem Dienst JournalTOCs abonniert werden, in ein geeignetes Format für die Bibliothekssuchmaschine ALBERT gebracht werden können, besteht eigentlich aus drei einzelnen Prozessen.

Diese Prozesse können zwar unabhängig voneinander durchgeführt werden, bauen allerdings aufeinander auf, da für die Anreicherung der aktuellen Zeitschriftenartikel mit Themengebieten aus der EZB beispielsweise die Ausgabedateien des ersten Prozesses benötigt werden, wie auch das Schema auf der folgenden Seite zeigt.

Dies bedeutet, dass die Prozesse beim ersten Mal auch am besten in der dargestellten Reihenfolge durchgeführt werden sollten. Wenn die ersten beiden Prozesse, insbesondere der zweite Prozess, ein Mal durchgeführt wurden und die Ausgabedateien nicht verschoben oder gelöscht werden, kann der letzte Prozess immer wieder durchgeführt werden, ohne dass man die anderen Prozesse auch ablaufen lassen muss.

Die Prozesse wurden entwickelt, um die Verwaltung von elektronischen Zeitschriften und Feeds mit neuen Zeitschriftenartikeln an möglichst wenigen Stellen zu zentralisieren und um die Daten für die beiden Informationsarten relativ einfach in die Suchmaschine ALBERT integrieren zu können.

Bei der Konzeption der Prozesse wurde in der Regel immer gleich vorgegangen. Zunächst wurden die Eingabedateien von den verschiedenen Systemen betrachtet und überlegt welche Informationen daraus für ALBERT genutzt werden sollen. Dann wurde darüber nachgedacht, in welche Struktur, d.h. in welche Elemente, diese Informationen für ALBERT gebracht werden müssen. Auf diese Weise wurde ein Musterbeispiel für einen Datensatz erzeugt. Dann wurde mit der Konzeption eines Stylesheets begonnen, das diese Umstrukturierung so weit wie möglich übernimmt. Da aber nicht alle Funktionen, wie zum Beispiel das Herunterladen der aktuellen Feed-Dateien im dritten Prozess, von den Stylesheets übernommen werden können, wurden entsprechende Befehle für die Software Ant gesucht (vgl. blau Gekennzeichnetes in „Abbildung 1“). Die Software sollte ursprünglich nur verwendet werden, da sie mehrere Dateien nacheinander mit einem oder mehreren Stylesheets bearbeiten kann.

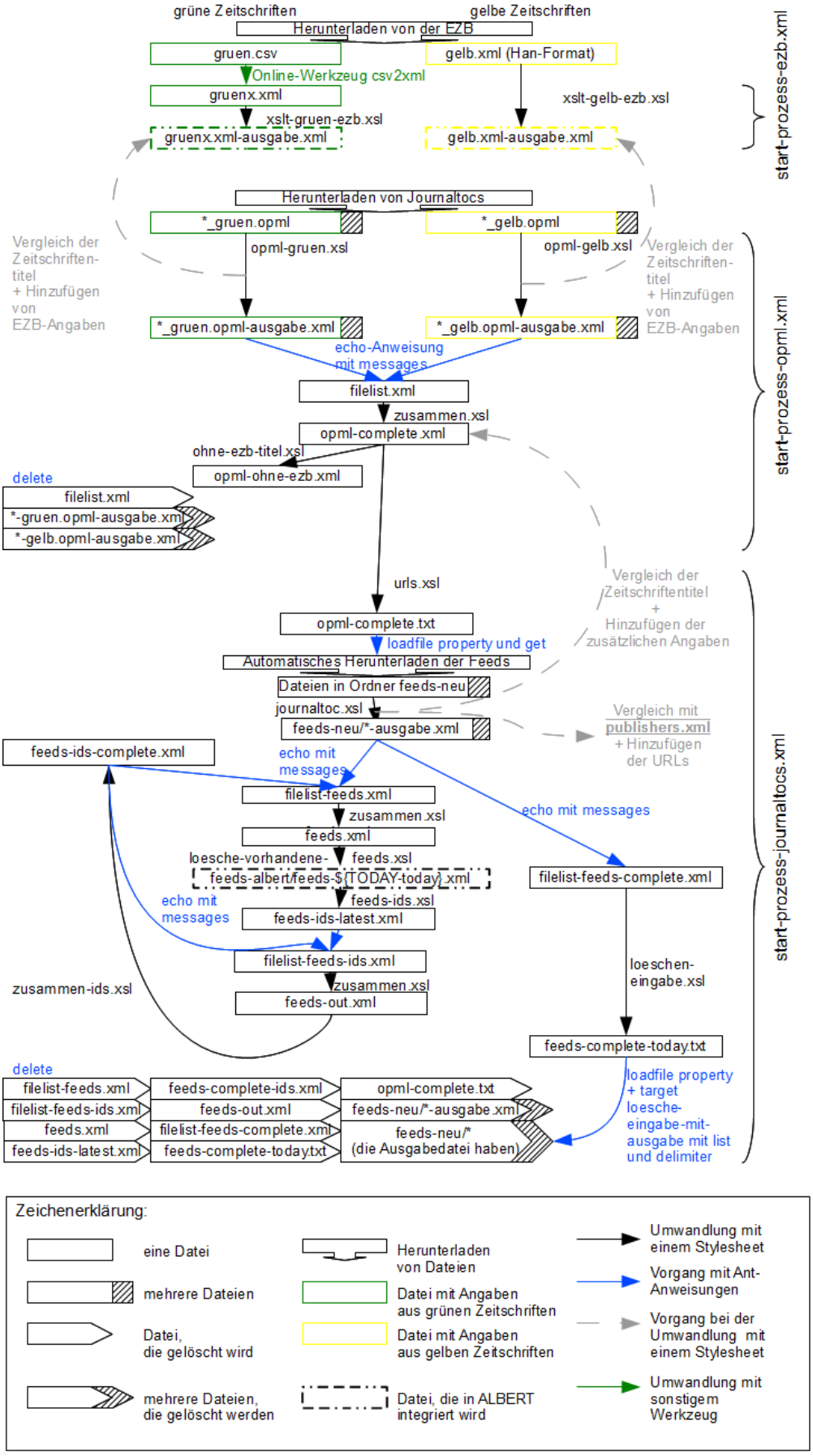

Abbildung 1: Schema des gesamten Prozesses

4.1. Zeitschriften aus der Elektronischen Zeitschriftenbibliothek

Die Elektronische Zeitschriftenbibliothek wurde als zentrale Stelle für das Verwalten von elektronischen Zeitschriften ausgewählt, weil sehr viele Zeitschriften enthalten sind und viele Bibliotheken die EZB verwenden. Dadurch wird der Prozess auch einfacher nachnutzbar. Außerdem werden durch die vielen Teilnehmer die allgemeinen Angaben zu den Zeitschriften immer relativ aktuell gehalten.
Um den Prozess für die Zeitschriften zu starten, müssen zunächst die Zeitschriften-bestände aus der EZB heruntergeladen werden. Dabei gibt es allerdings das Problem, dass die freien bzw. grünen Zeitschriften nur als CSV-Datei (Datei im Format Comma-Separated Values), die nicht mit XSLT verarbeitet werden kann, heruntergeladen werden können.
Da dieses Problem auch nicht mit Ant gelöst werden konnte, wird die Datei, die gruen.csv genannt wird, mit dem Werkzeug „CSV2XML Konverter“[265] in ein XML-Dokument namens gruenx.xml umgewandelt. Vorher müssen in der Datei allerdings erst alle Tabulatorzeichen, die auch als Tab-Stop bezeichnet werden und als Steuerzeichen als „\t“ in einem Texteditor angegeben werden können, durch |-Zeichen ersetzt werden, damit die einzelnen Informationen einer Zeitschrift zu eigenen Elementen im XML-Dokument umgesetzt werden können.
Bei der Umwandlung werden bei den Optionen des Werkzeugs das |-Zeichen als Trennzeichen und „row“ als Zeilenname eingetragen. Die anderen Einstellungen werden nicht verändert. Der Zeilenname dient dazu, die einzelnen Zeitschriften zu eigenen, voneinander unabhängigen Elementen umzuwandeln.
Wenn die Datei gruenx.xml erzeugt wurde, in den gleichen Ordner wie die Datei start-prozess-ezb.xml gespeichert wurde und die Datei für die gelben Zeitschriften aus der EZB im Han-Format als gelb.xml im gleichen Ordner gespeichert wurde, kann der Prozess mit der Eingabeaufforderung gestartet werden.
Aber vor dem Start des Prozesses müssen auch die Build-Dateien, der Stylesheetordner und eine binäre Version von Ant im Basisverzeichnis gespeichert und die Ordner feeds-neu und feeds-albert dort angelegt werden. Dann muss noch eine binäre Ant-contrib-Version heruntergeladen und entpackt so in das lib-Verzeichnis von Ant gespeichert werden, dass sich z.B. ant-contrib-0.6.jar direkt im

[265] http://www.oio.de/public/konverter/csv2xml.htm, [letzter Zugriff: 12.01.2012].

lib-Verzeichnis befindet. Ebenso müssen die Pfade zu dieser Datei in den Build-Dateien angepasst werden.

Zunächst begibt man sich in der Eingabeaufforderung in das dem Basisverzeichnis untergeordnete Ant-Verzeichnis „bin“. Dann gibt man einen Befehl an Ant, in dem die Build-Datei und das Verzeichnis, in dem die zu bearbeitenden Dateien liegen, genannt werden. Beides könnte etwa so aussehen:

```
cd Z:\lucene_kobv\daten\admintools\xslt\bachelor\ant\bin

ant -f ../../start-prozess-ezb.xml -
Ddirectory="Z:\lucene_kobv\daten\admintools\xslt\bachelor"
```

In der Build-Datei start-prozess-ezb.xml gibt es nur ein <target>, das dafür sorgt, dass die Transformationen von gruenx.xml und gelb.xml mit den jeweiligen Stylesheets (xslt-ezb-gruen.xsl und xslt-ezb-gelb.xml) zu gruenx.xml-ausgabe.xml und gelb.xml-ausgabe.xml nacheinander durchgeführt werden. Da in der Build-Datei die Namen explizit so angegeben sind, wie sie hier genannt werden, müssen die Dateien auch immer so heißen, damit sie von Ant bearbeitet werden können.

Der Ablauf der Arbeitsschritte soll auch noch einmal durch die folgende Abbildung verdeutlicht werden.

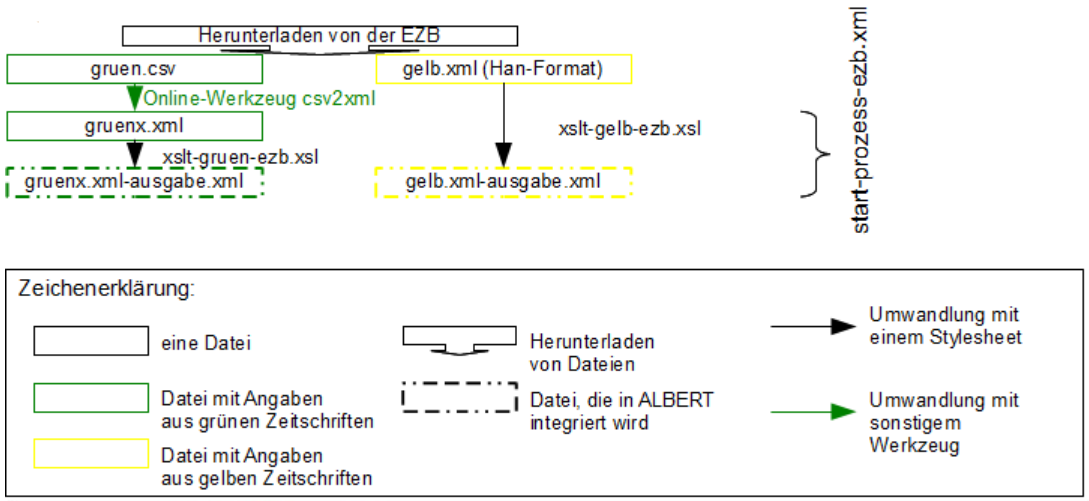

Abbildung 2: Schema des Prozesses für die EZB

Das Stylesheet für die grünen Zeitschriften wandelt die Angaben der Zeitschriften in das von ALBERT benötigte Format um, das eine Mischung aus MAB-XML und einem eigenen Format darstellt (Vgl. Anhang 1 und 2).

Bei dieser und allen anderen Transformationen der drei Prozesse werden hauptsächlich nur die Elementnamen verändert, indem ein Template für das jeweilige Element aus der Eingabedatei aufgerufen wird. In dem Template wird dann die

gewünschte Bezeichnung für die Ausgabe als Element um ein <xsl:value-of>, das den Inhalt des aktuellen Knoten selektiert, gruppiert. Ein Beispiel dafür ist:

```
<xsl:template match="Link_zur_Zeitschrift">
        <feld nr="94b" lb="">
            <xsl:value-of select="."/>
        </feld>
</xsl:template>
```

Besonders an der Umwandlung der grünen Zeitschriften aus der EZB ist jedoch, dass die Themengebiete, die im Eingabedokument mit Begriffen angegeben sind, mittels einer Schleife, die durch ein benanntes Template und einen Parameter erzeugt wird, in die Notation, die aus der Regensburger Verbundklassifikation stammt und bei den gelben Zeitschriften auch so angegeben wird, umgesetzt wird.
Dieser Vorgang wird mit einem Template für das Element <row>, das das übergeordnete Element für jede Zeitschrift ist, begonnen, indem mit dem Aufruf

```
<topics>
    <xsl:call-template name="Fach">
        <xsl:with-param name="topic" select="Fach"/>
    </xsl:call-template>
</topics>
```

ein Template namens „Fach“ aufgerufen und dabei der Inhalt des Kindelements <Fach> von <row>, d.h. die Themengebiete der jeweiligen Zeitschrift, als Parameter „topic“ übergeben wird.
Nun wird im Template „Fach“ mit einem <xsl:choose> und den dazugehörigen xsl:when- und xsl:otherwise-Elementen getestet, ob ein Semikolon im Parameter „topic“ enthalten ist. Dies weist nämlich darauf hin, dass die Zeitschrift mehreren Themengebieten zugeordnet ist. Gleichzeitig ist das der Grund dafür, dass die Schleife verwendet werden muss, weil die Themengebiete in der Ausgabe nämlich in eigene Elemente transferiert werden sollen.
Dann wird in einem weiteren xsl:choose-Komplex getestet, ob der Parameter „topic“ einen Bestandteil einer Fachbezeichnung enthält.

```
<xsl:when test="contains(substring-before($topic,';'), 'Anglistik')">
    <topic>H</topic>
        <xsl:call-template name="Fach">
            <xsl:with-param name="topic" select="substring-after($topic,';')"/>
        </xsl:call-template>
</xsl:when>
```

Diese Anweisung zeigt, dass die Zeichenkette “Anglistik“ vor dem Semikolon in dem Parameter gesucht wird. Wird die Zeichenkette gefunden, wird ein „H“ als der Inhalt des topic-Elements in die Ausgabedatei eingefügt.

Dann wird das gleiche Template noch einmal aufgerufen, wobei der Parameter „topic“ dieses Mal den Wert nach dem ersten Semikolon aus dem ehemaligen Parameterinhalt enthält. Dieser Wert ist das zweite Themengebiet aus dem Fach-Element des Eingabedokuments.
Nun wird diese Zeichenkette wieder darauf getestet, ob ein Semikolon vorhanden ist und bei einem positiven Ergebnis das Fachgebiet auf die gleiche Weise umgesetzt, wie gerade beschrieben. Dieser Prozess wird so lange wiederholt, bis kein Semikolon mehr vorhanden ist. Dann wird das letzte Themengebiet, das ja nicht mehr durch ein Semikolon von einem anderen getrennt werden muss, mit der gleichen Methode wie oben ersetzt, nur dass das Template danach nicht mehr aufgerufen wird. Gleiches gilt auch für die Fach-Elemente, in denen nur ein Themengebiet angegeben ist und die deshalb auch kein Semikolon enthalten.
Für den Fall, dass ein Themengebiet angegeben wurde, dass nicht zu den Alternativen gehört, die im Stylesheet aufgeführt werden, gibt es noch die Option in einem xsl:otherwise-Element, dass das Fachgebiet „nicht codiert“, d.h. in Form der Klassenbezeichnung, ausgegeben wird.
Auf die gleiche Weise funktioniert ein Template, bei dem E-ISSNs (Electronic International Standard Serial Numbers) bzw. Print-ISSNs, die in der Eingabedatei auch durch ein Semikolon voneinander abgegrenzt werden, getrennt und in einzelne Elemente verteilt werden. Auch ein Template für alle Textknoten, die Kindelemente von Titel- oder Verlag-Elementen sind, wird auf die gleiche Weise realisiert. Dabei wird im Template für die Textknoten ein Template namens „sonderzeichen4“ aufgerufen, um in der Ausgabe anstatt eines „&“ ein „&“ zu erzeugen.
Die Schleife ist an dieser Stelle nötig, weil die Textknoten ein „&“ auch mehrfach hintereinander enthalten könnten und ohne eine Schleife nur das erste „&“ ersetzt würde. Um das „&“ auch genauso in der Ausgabedatei zu erhalten, steht es in einem xsl:text-Element mit dem Attribut und Wert „disable-output-escaping="yes"“. Zusätzlich ist es nötig, damit der Dezimalcode nicht direkt in ein Et-Zeichen umgesetzt wird und die Ausgabedatei dadurch ungültige Zeichen enthält, dass der Code im Stylesheet in ein <![CDATA[…]> eingefügt wird. Dieses Element sorgt dafür, dass sein Inhalt von einem Prozessor genauso wie angegeben in die Ausgabedatei überführt wird, weil er vom Prozessor nicht interpretiert bzw. in gewisser Weise übergangen wird.

Bei den E-ISSNs wurden auch Angaben gefunden, die keine E-ISSNs, sondern nur einen Bindestrich enthalten, wodurch es nötig wurde, diese Angaben von der weiteren Auswertung mit <xsl:if test="E-ISSN/text() and not(E-ISSN/text()='-')"> auszuschließen. Dies bedeutet für den Prozessor so viel wie, mache das, was in der Bedingung steht, wenn ein E-ISSN-Kindelement mit einem Textknoten als Kindelement vorhanden ist und es kein E-ISSN-Kindelement mit einem Textknoten, der als einzigen Inhalt einen „-“ enthält, gibt. Dies funktioniert natürlich nur deshalb, weil es, wenn es eine E-ISSN gibt, kein weiteres E-ISSN-Kindelement mit einem „-“ als Inhalt geben kann, da alle E-ISSNs in einem Feld stehen.
Mit verschiedenen Bedingungen werden außerdem die Angaben zu den verfügbaren Jahrgängen der Zeitschriften je zu einer Bestandsangabe vereinigt, die aus einem from- und einem until-Element bestehen kann. Die Angaben werden, wenn möglich folgendem Schema angepasst: „Volume (Issue).Jahr“.
Dazu wird in einem Template für das erstes_Jahr-Element einer Zeitschrift der Eingabedatei für die Ausgabe des Startzeitpunktes des freizugänglichen Bestandes der Zeitschrift das moving_wall-Element auf der gleichen Ebene über den Aufruf

```
<xsl:when test="contains(../moving_wall,'&#43;')">
```

darauf getestet, ob es die Zeichenkette “+“ enthält. Wenn dieses Element die Zeichenkette enthält, ist klar, dass die Zeitschrift nur für einen bestimmten Zeitraum, etwa die letzten zwei Jahre, Monate oder Wochen für den Nutzer zugänglich ist. Dann wird über eine weitere „Entweder-Oder-Bedingung“ getestet, ob das moving_wall-Element ein „W“, „M“, oder „Y“ enthält, um festzustellen, welche Art von Zeitraum vorliegt (Woche, Monat oder Jahr). Auf der Basis des Testergebnisses wird eine entsprechende Ausgabe erzeugt, bei der zum Beispiel in „nur die letzten … Wochen“ die Zahlen, die hinter dem jeweiligen Buchstaben stehen, eingefügt werden. Für den Fall, dass die Zahl eine eins ist, wird über eine eigene Bedingung pro Art des Zeitraums die Ausgabe z.B. in „nur die letzte Woche“ angepasst.

Enthält <moving_wall> die Zeichenkette “+“ nicht, wird über verschiedene Bedingungen, wozu als erstes ein Test auf einen Textinhalt in <erstes_Jahr> gehört, die Ausgabe in dem beschriebenen Schema erzeugt, oder auf die vorhandenen Bestandteile angepasst. So ist z. B. auch die Angabe des Startzeitpunktes des zugänglichen Bestandes nach Schemata wie „Volume.Jahr“, „Issue.Jahr“ oder „Jahr“ möglich.

Hat weder das Element <erstes_Jahr>, noch das Element <moving_wall> einen Textinhalt wird die Ausgabe <from>gesamter Zeitraum</from> erzeugt, da davon ausgegangen wird, dass es ohne Startzeitpunkt auch keinen Abschlusszeitpunkt gibt und daher also alle Ausgaben im Erscheinungsverlauf der Zeitschrift zugänglich sind.
Etwa genauso wie bei der Erzeugung eines from-Elements wird auch für die Ausgabe des until-Elements, d.h. die Angabe der letzten Ausgabe, die zugänglich ist, vorgegangen, wobei das moving_wall-Element auf ein „-“ als Bestandteil des Textinhalts geprüft wird, welches so viel bedeutet, wie „nicht die letzten … Wochen/Monate/Jahre“. Ist keine solche Angabe vorhanden, werden die Bestandsangaben aus den entsprechenden Elementen in <until> übernommen.
Die vorrausgegangenen Trennungen von Themengebieten, E-ISSNs und die Ausgabe des Bestandszeitraums können auch an dem Beispiel im Anhang (Anhang 1 und 2) nachvollzogen werden.

Das Stylesheet für die Transformation der gelben, d.h. lizensierten, Zeitschriften gestaltet sich etwas umfangreicher, da mehr Umformungen vorgenommen werden müssen. Dies liegt daran, dass eine Zeitschrift über verschiedene Lizenzen zugänglich sein kann, die auch alle in der EZB angegeben werden. Ein Teil der Zeitschrift kann beispielsweise über eine Nationallizenz oder eine Konsortiallizenz zugänglich sein und der übrige Teil der Zeitschrift kann dann von der jeweiligen Bibliothek selbst lizensiert werden, um alle Ausgaben der Zeitschrift anbieten zu können. Dadurch erhält eine Zeitschrift allerdings verschiedene Bestandsangaben mit unter Umständen eigenen Angaben zu den Internetadressen, bei denen auf den Bestand zugegriffen werden kann.
Da sich der Nutzer in der Regel nicht erklären kann, warum mehrere Bestandsangaben vorhanden sind und ihn meist auch nur interessiert, zu welchen Jahrgängen er insgesamt Zugang hat, werden die unterschiedlichen Bestandsangaben mit dem Stylesheet xslt-ezb-gelb.xsl zu einer Angabe vereinigt, bei der das früheste Startdatum zum Anfang und die späteste Angabe im Abschlussdatumsfeld zum Ende des Zugangszeitraumes im Ausgabedokument wird.
Diese Vorgehensweise führt allerdings dazu, dass Bestandslücken ignoriert und deshalb nicht aufgeführt werden. Da aber Bestandslücken bei elektronischen Zeitschriften eher selten sind und bei der Konzeption des Prozesses keine

Zeitschriften mit Bestandslücken gefunden wurden, wird dieses Risiko hingenommen.
Sollte es eine Bestandsangabe geben, bei der das Abschlussdatum fehlt, wird angenommen, dass die Zeitschrift laufend bezogen wird und daher wird auch kein Abschlussdatum in die Ausgabedatei eingefügt.
Wenn es kein Startdatum gibt, wird davon ausgegangen, dass der gesamte Zeitraum der Zeitschrift lizensiert wird, was auch so, wie bei den grünen Zeitschriften, in der Ausgabe in das Element <from> geschrieben wird.
Um aufbauend auf diesen Voraussetzungen und der Forderung, für die Bestände nach dem gleichen Schema, wie bei den grünen Zeitschriften, eine Ausgabe für den gesamten Zugangszeitraum zu erzeugen, wurden sieben Templates geschrieben, die dies ermöglichen sollen.
Um den Prozess etwas verständlicher zu machen, soll er an nachstehendem Beispiel, das sich auch als vollständiger Datensatz im Anhang (Anhang 3 und 4) befindet, erklärt werden. Hat eine Zeitschrift beispielsweise folgende Bestandsinformationen, die hier verkürzt angegeben werden, ist nach längerem Betrachten ersichtlich, dass die Zeitschrift ab dem ersten Volume 1995 bis heute lizensiert wird.

```
<journal><!-- hier wurden Inhalte entnommen -->
    <licenseinfo>
        <first_volume>3</first_volume><first_date>1997</first_date>
        <anchor><![CDATA[Springer]]></anchor>
        <url>http%3A%2F%2Fwww.springerlink.com%2Flink.asp%3Fid%
        3D100529</url>
    </licenseinfo>
    <licenseinfo>
        <first_volume>1</first_volume><first_date>1995</first_date>
        <last_volume>8</last_volume><last_date>2002</last_date>
        <anchor><![CDATA[natli_springer]]></anchor>
        <url>http%3A%2F%2Fwww.springerlink.com%2Fopenurl.asp%3F
        genre%3Djournal%26issn%3D1610-2940</url>
    </licenseinfo>
</journal>
```

Damit dies auch so in die Ausgabedatei transportiert wird, wird ein Template für die Bestandsangaben einer Zeitschrift aufgerufen, die in einem journal-Element stehen. Mit einer xsl:for-each-Anweisung wird nun in diesem Template veranlasst, dass alle licenseinfo-Elemente, die die einzelnen Bestandsinformationen für eine Lizenz enthalten, nach dem Inhalt ihres first_date-Kindelements aufsteigend sortiert werden. In dem Beispiel würde die Reihenfolge der licenseinfo-Elemente umgekehrt, da in der zweiten Bestandsangabe das frühere Jahr steht.

Dann werden mittels einer xsl:if-Anweisung für das erste licenseinfo-Element, d.h. das mit der frühesten Jahresangabe in <first_date>, Templates für die Internetadresse und für die E-ISSNs, soweit sie vorhanden sind, aufgerufen. Für die licenseinfo-Elemente, die ein first_date-Kindelement haben, wird dann das Template namens „holding“ aufgerufen. Dies wäre im Beispiel das ehemalige zweite licenseinfo-Element, da es die früheste Jahreszahl, nämlich 1995, enthält.
Da es aber auch licenseinfo-Elemente gibt, die über kein first_date-Kindelement verfügen, weil die Jahresangabe fälschlicher Weise in ein first_volume-Element eingetragen wurde, oder weil der ganze Zeitraum lizensiert ist, wird mit einem <xsl:choose> getestet, ob das erste <licenseinfo> ein <first_volume> und ein anderes <licenseinfo> ein <first_date> enthält. Das heißt, es wird getestet, ob die Angabe des Jahres in das Element für das Volume eingetragen wurde und ob es eine andere Bestandsangabe gibt, bei der eine Jahresangabe vorhanden ist. In diesem Fall wird ein Template für alle licenseinfo-Elemente der Zeitschrift im mode „zwei“ aufgerufen. Hierbei werden die licenseinfo-Elemente wieder nach dem Inhalt ihres first_date-Kindelements sortiert.
Auf die gleiche Weise wird verfahren, wenn das licenseinfo-Element *kein* first_volume-Element enthält.
Davor gibt es aber auch einen Test, bei dem lediglich geprüft wird, ob das erste <licenseinfo> ein first_volume-Kindelement besitzt. Da aber vorher schon geprüft wurde, ob es ein anderes licenseinfo-Element gibt, bei dem ein <first_date> vorhanden ist, bedeutet ein positives Ergebnis bei diesem Test, dass die Jahresangabe in das Element für das Volume geschrieben wurde. In diesem Fall wird auch das Template namens „holding“ aufgerufen.
Im Template für die licenseinfo-Elemente im mode „zwei“ wird, weil ja das erste licenseinfo-Element nach der Sortierung kein first_date-Kindelement besaß, für das zweite <licenseinfo> das Template namens „holding“ aufgerufen.
Da es aber auch Zeitschriften gibt, die nur eine licenseinfo-Angabe und in dieser kein first_date- und kein first-volume-Element haben, wird in diesem Fall ein drittes Template für alle licenseinfo-Elemente im mode „drei“ aufgerufen. Auch an dieser Stelle werden die licenseinfo-Elemente wieder nach der gleichen Angabe sortiert.

In dem Template im mode „drei“, wird für das erste <licenseinfo>, wenn es keine first_date- oder first_volume-Elemente in irgendeinem licenseinfo-Element der

Zeitschrift gibt, das Template namens „holding“ aufgerufen. Eigentlich ist dieser erneute Test nicht nötig, da es in der ersten Bestandsangabe kein Jahr und kein Volume gibt und keine zweite Bestandsangabe vorhanden ist. Der Test dient demnach lediglich der Absicherung.
Dieser etwas längere Weg soll auch durch die folgende Grafik verdeutlicht werden, die als Entscheidungsbaum dargestellt wird, bei dem die erste Bedingung, die auf der jeweiligen Ebene überprüft wird, links steht. Die anderen Bedingungen, werden von links nach rechts weitergetestet, wenn die vorherigen Bedingungen auf der gleichen Ebene nicht erfüllt werden konnten.

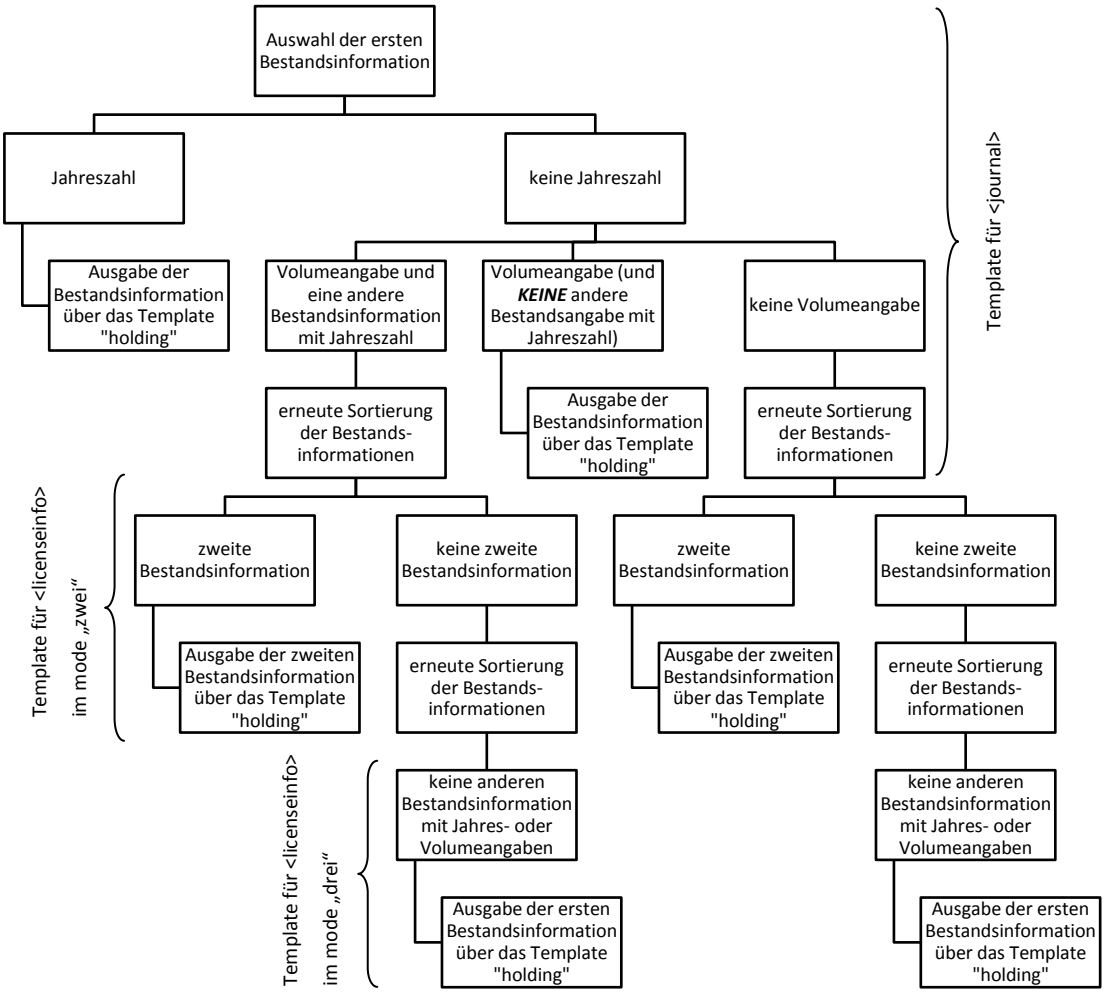

Abbildung 3: Auswahlprozess für den Startzeitpunkt des Zugangszeitraums für eine gelbe Zeitschrift

Das Template namens „holding“ führt nun endlich durch verschachtelte Bedingungen zu der Ausgabe des Beginns des Zugangszeitraums.
Zunächst wird getestet, ob ein first_date- oder ein first_volume-Kindelement vorhanden ist, was in dem Beispieldatensatz beides vorhanden wäre.

Ist keins der Elemente vorhanden, wird davon ausgegangen, dass der gesamte Zeitraum der Zeitschrift zugänglich ist. Die Ausgabe <from>gesamter Zeitraum</from> erfolgt durch die Verwendung von <xsl:otherwise>.
Danach wird für die Bestandsinformationen, bei denen die Jahresangabe im first_volume-Element steht, die Ausgabe nach dem oben genannten Schema erzeugt, nur dass die Volumeangaben als Jahresangaben und die Issueangaben, soweit sie vorhanden sind, an der Stelle von Volume ausgegeben werden. Durch weitere Bedingungen wird dann für die korrekten Angaben das jeweils passende Schema erzeugt. Im Beispiel würde als Ergebnis in der Ausgabedatei die Struktur <from>1.1995</from> erzeugt.
Dann wird in etwa der gleiche Prozess für die Ausgabe der Abschlussangabe des lizensierten Zeitraums über das Template „holding“ und zwei Templates für die licenseinfo-Element im mode „lastdate“ und „lastvolume“ durchgeführt, außer, dass die licenseinfo-Elemente natürlich in absteigender Reihenfolge anhand des Inhalts des last_date- bzw. last_volume-Elements sortiert werden. Außerdem wird, wenn es eine Angabe zum Startzeitpunkt des Zugangszeitraums und eine Bestandsangabe ohne Endzeitpunkt gibt, wie im Beispiel, keine Ausgabe eines Endzeitpunktes gemacht. Hier kann nämlich davon ausgegangen werden, dass es eine Lizenz gibt, die noch laufende Zeitschriftenbände enthält, weshalb auch keine Angabe eines Endzeitpunktes nötig ist.
Das bedeutet insgesamt gesehen, dass die Angaben zum Beginn und zum Ende der Lizenz getrennt voneinander betrachtet werden.
Für das Beispiel würde sich so aus den beiden sich überlappenden Zeiträumen über die Templates die folgende Bestandsangabe ergeben:

<holding type=“online“><from>1.1995</from></holding>

Das heißt also, dass alle Hefte ab dem ersten Volume 1995 elektronisch zugänglich sind.
Zusätzlich zu dieser komplexen Vorgehensweise um den Zugangszeitraum ausgeben zu lassen, wird analog zu den Mechanismen für die Ausgabe der Fachgebiete in codierter Form und für die Ausgabe der E-ISSNs aus dem Stylesheet für die grünen Zeitschriften vorgegangen, um bei den Angaben der Internetadressen die codierten Sonderzeichen durch ihre nicht codierte Form zu ersetzen. Bei dem Export aus der EZB werden nämlich beispielsweise Doppelpunkte zu „%3A“ und Gleichzeichen zu „%3D“. Die Zeichen werden in ihre nicht codierte Form gebracht, damit man die

Adressen direkt aus ALBERT aufrufen kann, da hierfür eine Angabe wie „http%3A%2F%2Fwww.springerlink.com%2Flink.asp%3Fid%3D100529“ nicht verwendbar ist. Damit die Schleife alle Zeichen hintereinander ersetzt, wird die Adresse als Parameter mit der Funktion contains( ) auf ein Prozentzeichen durchsucht und dann, je nachdem welche Zeichen dahinter stehen, das richtige Sonderzeichen eingefügt, wobei zuerst die Zeichenkette vor dem Prozentzeichen und dann das nicht codierte Sonderzeichen ausgegeben wird. Danach ruft sich das Template wieder selbst auf und durchsucht nun den Teil der Zeichenkette nach dem ersten codierten Sonderzeichen, der als Parameter weitergegeben wurde, auf ein Prozentzeichen. Dann wird der davor befindliche Parameterinhalt ausgegeben und wieder die codierte Form eines Sonderzeichens durch die nicht codierte ersetzt usw. bis keine Prozentzeichen mehr enthalten sind. Danach wird die übrige Zeichenkette hinter den anderen Bestandteilen der ursprünglichen Zeichenkette mit den ersetzten Sonderzeichen ausgegeben. Das Template für die Sonderzeichen beginnt wie folgt:

```
<xsl:template name=“sonderzeichen“>
<xsl:param name="string"/>
<xsl:param name="from"/>
<xsl:choose>
<xsl:when test="contains($string, $from)">
  <xsl:value-of select="substring-before($string, $from)"/>
  <xsl:choose>
    <xsl:when test="substring(substring-after($string, $from),1,2)='2B'">+
      <xsl:call-template name="sonderzeichen">
        <xsl:with-param name="string" select="substring(substring-after($string,
        $from),3)"/>
            <xsl:with-param name="from" select="$from"/>
      </xsl:call-template>
   </xsl:when>
```

Der Parameter „from“ ist das Prozentzeichen und der Parameter „string“ ist die Zeichenkette, die auf das Zeichen durchsucht wird. Im erneuten Aufruf des Templates aus dem Beispiel wird der Teil der Zeichenkette, die noch im Parameter „string“ ist, drei Zeichen nach dem Auftreten des Prozentzeichens zum Parameter „string“. Das Prozentzeichen wird hierbei mitgezählt.

Das Prozentzeichen wird als Muster zum Durchsuchen verwendet, weil es in allen codierten Sonderzeichen vorkommt und es, wenn man jedes Sonderzeichen als Muster einzeln verwenden würde, nicht so einfach möglich wäre eine Ausgabe zu erzeugen, in der alle Sonderzeichen hintereinander ersetzt worden sind.

Nachdem beide Eingabedateien umgewandelt wurden, können die Ausgabedateien gruenx.xml-ausgabe.xml und gelb.xml-ausgabe.xml in ALBERT integriert werden.

4.2. Aktuelle Zeitschriftenartikel von dem Dienst JournalTOCs

Der Dienst JournalTOCs wurde für das Abonnieren von neuen Zeitschriftenartikeln ausgewählt, weil er kostenlos genutzt werden kann und die Zeitschriftenartikel in einem einheitlichen Format vorliegen. Sonst werden nämlich die Feeds häufig direkt beim jeweiligen Verlag abonniert, wodurch man sehr unterschiedliche Formate verarbeiten müsste, wenn man sie einheitlich unter einer Oberfläche anbieten wollte. Hierdurch würde sich auch der Verwaltungsaufwand vergrößern. Dafür hat der Dienst jedoch den Nachteil, dass die Angaben nicht so aktuell wie bei den Verlagen sind.

Damit man JournalTOCs für das Abonnieren von Zeitschrifteninhaltsverzeichnissen verwenden kann, muss man sich zunächst anmelden und kann dann eine Sammlung von Zeitschriften anlegen, für die die Inhaltsverzeichnisse abonniert werden.

4.2.1. Vergleich der Zeitschriftentitel aus den beiden Systemen

Die gerade erwähnte Sammlung von Zeitschriften kann auch exportiert und als OPML-Datei (Datei im Format Outline Processor Markup Language) lokal gespeichert werden. Diese Datei oder besser gesagt diese Dateien, da die Sammlung an Zeitschriften auf die grünen und die gelben Zeitschriften aufgeteilt werden muss, werden dazu verwendet, die Angaben zu den Zeitschriftenartikeln von JournalTOCs mit zusätzlichen Informationen, wie den Themengebieten oder den E-ISSNs der Zeitschriften, die in der EZB vorhanden sind, anzureichern. Der Prozess ist allerdings nur ein Zwischenschritt, da an dieser Stelle lediglich die Zeitschriftentitel mit den zusätzlichen Angaben versehen werden. Dafür hat dieses Vorgehen den Vorteil, dass der Prozess nur dann wiederholt werden muss, wenn neue Zeitschriften von JournalTOCs abonniert werden sollen.

Im Prinzip wird für die grünen und gelben Zeitschriften das Gleiche gemacht. Sie werden nur getrennt, weil die Zeitschriften in der EZB auch getrennt vorliegen und weil der Prozess bei einer Trennung schneller abläuft, da nicht jeder Zeitschriftentitel aus JournalTOCs mit jedem Titel aus der EZB verglichen werden muss.

Wie in „Abbildung 4“ zu erkennen ist, wird am Schluss des Prozesses eine Datei erzeugt, die alle Zeitschriften mit ihren zusätzlichen Informationen enthält. Diese Datei wird opml-complete.xml genannt.

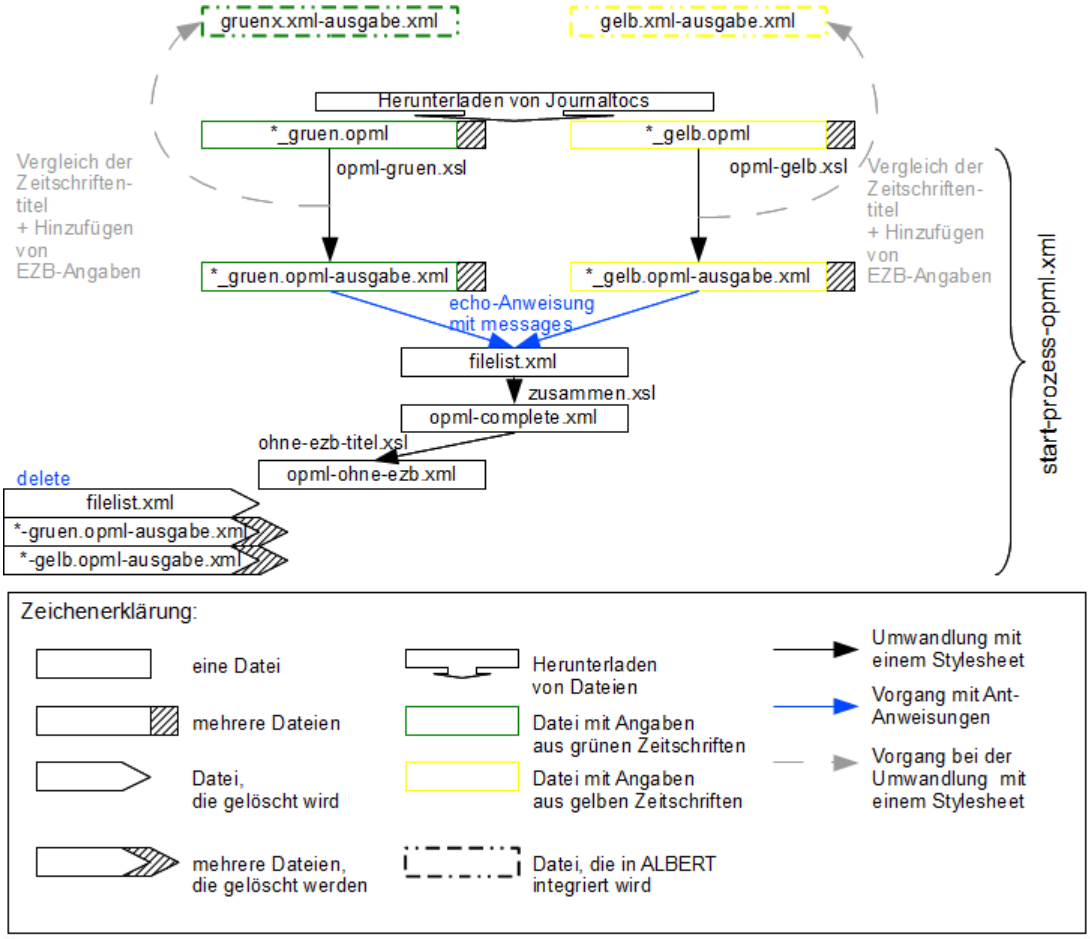

Abbildung 4: Schema des Prozesses für die OPML-Dateien aus JournalTOCs

Nachdem man also die OPML-Dateien heruntergeladen hat und der vorherige Prozess schon durchgeführt wurde, kann dieser Prozess mit der Build-Datei start-prozess-opml.xml mit Ant begonnen werden.

Der vorherige Prozess muss ausgeführt worden sein, damit die Ergebnisdateien für den Vergleich der Zeitschriftentitel vorhanden sind. Auf die Ergebnisdateien des vorherigen Prozesses wird zugegriffen, weil beide die Daten im gleichen Format anbieten und das Format deshalb einfach für die Ausgabe dieses Prozesses übernommen werden kann.

Da sowohl die Eingabedateien als auch die zum Vergleich verwendeten Dateien das gleiche Format haben, sind die Stylesheets opml-gruen.xsl und opml-gelb.xsl bis auf eine document( )-Anweisung, in der auf die jeweilige Ergebnisdatei des EZB-Prozesses zugegriffen wird, identisch.

Der Vergleich der Zeitschriftentitel mittels der Stylesheets beginnt damit, dass für die Elemente, die in den OPML-Dateien die Zeitschriftentitel enthalten, ein Template aufgerufen wird, in dem die Titel auf eine Klammer getestet werden. Wenn eine Klammer enthalten ist, wird ein Template für die Zeitschriftenelemente aus dem

jeweiligen EZB-Ausgabedokument (gruenx.xml-ausgabe.xml oder gelb.xml-ausgabe.xml) aufgerufen und der Zeitschriftentitel aus der Eingabedatei als Parameter „opmltitel“ weitergegeben. Ebenso wird eine Variante des Titels, der aus einer Kombination der Zeichenketten vor und nach der Klammer hergestellt wird, ohne Doppelpunkte, Kommata, Bindestriche und Schrägstriche als Parameter „opmltitel-sonderzeichen“ weitergegeben.
Für die Kombination ohne die Zeichen wird folgender Ausdruck verwendet:

```
translate(concat(substring-before(@title,'('),substring-after(@title,')')),':,-/','')
```

Hierbei wird kein Trennzeichen für die Funktion concat( ) angegeben, d.h. die Zeichenkette vor der öffnenden Klammer und die Zeichenkette hinter der schließenden Klammer werden direkt aneinandergefügt. Die „Sonderzeichen“ werden durch das translate( ), bei dem der Bestandteil leer ist, der die Zeichen enthält, die als Ersatz eingefügt werden sollen, aus der Kombination des Titels entfernt.
Als letztes beinhaltet dann für die Titel, die keine Klammer enthalten, der Parameter „opmltitel-sonderzeichen“ den Zeitschriftentitel aus der Eingabedatei ohne die oben genannten Zeichen. Der Titel, so wie er in der Eingabedatei angegeben wird, wird als Parameter „opmltitel“ an das nächste Template, das auch für die Titel aus der jeweiligen EZB-Ausgabedatei aufgerufen wird, weitergegeben.
Diese unterschiedlichen Verarbeitungen werden mit einem <xsl:choose> umgesetzt.

Nachdem für alle Titel aus der OPML-Datei für alle Zeitschriften aus der EZB-Datei z.B. mit

```
<xsl:apply-templates select="document('../gelb.xml-ausgabe.xml')/records//record">
```

ein Template aufgerufen wurde, werden in diesem aufgerufenen Template auch für die EZB-Zeitschriften die Titel nach den gleichen Zeichen durchsucht und diese für bestimmte Parameter entfernt.
Der Parameter „hantitel“ enthält den Titel so, wie er in der EZB-Datei angegeben wird und der Parameter „hantitel-sonderzeichen“ enthält den Titel ohne Doppelpunkte, Kommata, Bindestriche und Schrägstriche. Dafür wird zum Beispiel die Funktion translate(feld[@nr='20'],':-,/','') angewendet.
Außerdem wird jeweils die ganze Zeitschrift, d.h. das übergeordnete Zeitschriftenelement, zum Parameter „journal“.

Für den Fall, dass der EZB-Titel eine Klammer enthält, werden weitere Parameter gebildet. Der erste Parameter ist „vorklammer“, der den EZB-Titel vor der Klammer ohne die Zeichen, die auch in „hantitel-sonderzeichen“ fehlen, enthält. Der Parameter „vorklammer-leerzeichen“ hat den gleichen Inhalt, wie der Parameter „vorklammer“, außer dass das Leerzeichen vor der öffnenden Klammer ebenfalls fehlt.
Ein weiterer Parameter ist „nachtitel“. Er beinhaltet den Rest des EZB-Titels nach der schließenden Klammer.
Der Inhalt der Klammer kann unter Umständen auch ein älterer Titel der Zeitschrift sein und noch in JournalTOCs verwendet werden, weshalb dieser Titel zum Inhalt des Parameters „inklammer“ wird.
Unter Weitergabe all dieser Parameter und der Parameter, die im Zusammenhang mit der OPML-Datei erzeugt wurden, wird für den Titel aus der OPML-Datei mit Hilfe des Parameters „opmltitel“ ein neues Template im mode „klammer“ aufgerufen.
In diesem Template für das Attribut „title“, das sich im Parameter „opmltitel“ befindet, wird dann mit verschiedenen Bedingungen geprüft, ob sich ein EZB-Titel und der JournalTOCs-Titel entsprechen bzw. enthalten.
Nach Erfüllen einer Bedingung wird immer getestet, ob der aktuelle Knoten auch der Parameter „opmltitel“ ist, weil sonst eine große Menge an eventuell passenden Treffern ausgegeben würde.
Getestet wird beim Vergleich der Titel, ob:

- der EZB-Titel nach der Klammer eine weitere Klammer enthält und der Titel vor der ersten Klammer ohne „Sonderzeichen“ dem JournalTOCs-Titel ohne „Sonderzeichen“ entspricht.

  ```
  contains($nachklammer,'(') and $opmltitel-sonderzeichen=$vorklammer
  ```

- der EZB-Titel vor dem Leerzeichen vor der Klammer nicht leer ist und dieser Titel und der JournalTOCs-Titel ohne „Sonderzeichen“ den gleichen Inhalt haben. Ersteres ist nötig, weil sonst auch wieder viele vermeintliche Übereinstimmungen gefunden würden. Dies betrifft Titel aus der EZB, die mehr als eine Klammer enthalten, da dann der Parameter mit dem Titel vor der zweiten Klammer leer wäre und ein leerer EZB-Titel in jedem JournalTOCs-Titel enthalten wäre. Zusätzlich wird auch getestet, ob eine zweite Klammer im EZB-Titel ist.

  ```
  not($vorklammer-leerzeichen='') and contains($nachklammer,'(') and $opmltitel-sonderzeichen=$vorklammer-leerzeichen
  ```

Beispiel: *EZB-Titel:* Journal of Earth System Science (1978-) (Formerly: Proceedings of the Indian Academy of Sciences : Earth and Planetary Sciences)
JournalTOCs-Titel: Journal of Earth System Science

- der JournalTOCs-Titel ohne „Sonderzeichen“ den Inhalt des EZB-Titels vor und nach der Klammer enthält und ob die Verbindung der beiden Teile des EZB-Titels den JournalTOCs-Titel ohne „Sonderzeichen“ enthält.

  contains($opmltitel-sonderzeichen,$vorklammer) and contains($opmltitel-sonderzeichen,$nachklammer) and contains(concat($vorklammer,$nachklammer),$opmltitel-sonderzeichen)

- der EZB-Titel vor dem Leerzeichen vor der Klammer nicht leer ist und dieser Titel, sowie der Bestandteil des EZB-Titels nach der Klammer im JournalTOCs-Titel ohne „Sonderzeichen“ enthalten sind. An dieser Stelle wird ebenfalls getestet, ob der JournalTOCs-Titel ohne „Sonderzeichen“ in der Verkettung der beiden Bestandteile des EZB-Titels gefunden werden kann.

  not($vorklammer-leerzeichen='') and contains($opmltitel-sonderzeichen,$vorklammer-leerzeichen) and contains($opmltitel-sonderzeichen,$nachklammer) and contains(concat($vorklammer-leerzeichen,$nachklammer),$opmltitel-sonderzeichen)

- der EZB-Titel ohne „Sonderzeichen“ dem JournalTOCs-Titel ohne „Sonderzeichen“ entspricht.

  $opmltitel-sonderzeichen=$hantitel-sonderzeichen

  Beispiel: *EZB-Titel:* Rem : Revista Escola de Minas
  JournalTOCs-Titel: Rem: Revista Escola de Minas

- der ursprüngliche EZB-Titel und der ursprüngliche JournalTOCs-Titel gleich sind.

  $opmltitel=$hantitel

  Beispiel: *EZB-Titel* und *JournalTOCs-Titel:* Antarctic Science

- der Inhalt der Klammer im EZB-Titel dem JournalTOCs-Titel ohne „Sonderzeichen“ entspricht.

  $inklammer=$opmltitel-sonderzeichen

  Beispiel: *EZB-Titel:* Marine Geophysical Research (formerly: Marine Geophysical Researches)
  JournalTOCs-Titel: Marine Geophysical Researches

Wird eine dieser Bedingungen erfüllt und stimmt der aktuelle Knoten mit dem Parameter „opmltitel“ überein, wird ein Template namens „zwei“ aufgerufen und

dabei der Parameter „journal“, d.h. das übergeordnete Zeitschriftenelement aus der EZB, mit seinem Inhalt weitergegeben.
Mit den Anweisungen in dem Template „zwei“ werden dann die entsprechenden Angaben und der Titel aus der EZB ausgeben. Der EZB-Titel wird ausgegeben, damit er in der Ergebnisdatei mit dem JournalTOCs-Titel verglichen werden kann.

An dieser Stelle muss klar gemacht werden, dass nicht für alle Zeitschriften, die mit JournalTOCs abonniert werden, der passende Titel in der EZB gefunden werden kann, weil die Titel manchmal sehr unterschiedlich geschrieben werden. Bei den 286 Zeitschriften, die bei der Konzeption des Prozesses verwendet wurden, konnten zum Beispiel bei insgesamt 44 Zeitschriften, von denen 24 Zeitschriften grün und 20 gelb sind, keine übereinstimmenden Titel gefunden werden. Insgesamt wurden bei der Entwicklung des Prozesses 130 grüne und 156 gelbe Zeitschriften ausgewählt, die auch in der EZB enthalten sind.
Außer den Angaben die aus der EZB übernommen werden, wird auch eine Internetadresse aus der Zeichenkette „http://www.journaltocs.ac.uk/api/journals/“, der ersten E-ISSN und danach der Zeichenkette „?output=articles“ erzeugt, die für das automatische Herunterladen der Feeds wichtig ist, da dort die aktuellsten Artikel der Zeitschrift, zu der die ISSN gehört, abgerufen werden können.
Allerdings haben nicht alle Zeitschriften in der EZB eine E-ISSN und für die Zeitschriften ohne E-ISSN wird dann auch keine Adresse erzeugt, d.h. es werden später auch keine Artikel heruntergeladen. Aus diesem Grund müssten die E-ISSNs für diese Zeitschriften in die jeweilige EZB-Ausgangsdatei oder die E-ISSNs und Adressen in die Ergebnisdatei des Prozesses opml-complete.xml eingefügt werden.

Im Prozess wird dann mit echo-Anweisungen in der Build-Datei eine Datei mit Pfaden zu den Ausgabedateien der Transformationen erzeugt, die filelist.xml heißt.

Mit dem Stylesheet zusammen.xsl wird danach filelist.xml wiederum umgewandelt. Dabei wird auf die Pfade zu den Dateien eine xsl:for-each-Anweisung angewendet, bei der die Pfade zu Parametern werden. Diese Parameter werden dann als Inhalt einer document( )-Funktion, die sich in einem select-Attribut eines xsl:copy-of-Elements befindet, dazu verwendet, den gesamten Inhalt der jeweiligen Datei in die Ausgabe zu übernehmen.

Zum Überprüfen, aus welcher Datei die Informationen in opml-complete.xml, das die Ergebnisdatei von dieser Transformation ist, ursprünglich stammen, wird in einem ausgangsdatei-Element jeweils der Pfad zu der Datei angeben, die kopiert wurde. Damit die Ergebnisdatei wohlgeformt ist, werden die Kopien der Ausgabedateien in einem rowdata-Wurzelelement erzeugt.
Nach der Erstellung von opml-comlete.xml wird eine Datei namens opml-ohne-ezb.xml mit dem Stylesheet ohne-ezb-titel.xsl erzeugt, in der nur die Titel aus den OPML-Dateien aufgeführt werden, für die kein oder mehrere passende Titel in der EZB gefunden wurden, oder die über keine E-ISSN in der EZB verfügen. Hinter dem jeweiligen Titel steht die Datei, aus der der Titel ursprünglich stammt.
Diese Funktion wird dadurch ermöglicht, dass für alle vor dem Titel befindlichen ausgangsdatei-Elemente in opml-complete.xml ein Template über die preceding-Achse aufgerufen wird und dann nur für das letzte von diesen eine Ausgabe erzeugt wird. Dies ist nötig, weil die beiden Elemente unabhängig voneinander sind, da die ausgangsdatei-Elemente vor den ehemaligen Wurzelelementen der kopierten Dateien stehen und sich daher mit ihnen auf der gleichen Ebene befinden, sodass keine Verbindung zwischen den Zeitschriftentiteln und der Angabe der Ursprungsdatei besteht.
Bei den Zeitschriften, bei denen mehrere Titel aus der EZB gefunden wurden, werden auch die zusätzlichen Angaben aus der EZB in opml-ohne-ezb.xml übernommen, um eine Entscheidung, welche Zeitschrift gewünscht war, zu erleichtern. Da sich diese Angaben in opml-complete.xml in einem Element auf der gleichen Ebene wie der Zeitschriftentitel befinden, wird über die Position des Titels, die zu einem Parameter wird, das Element mit den Angaben zur Zeitschrift mit der gleichen Position selektiert und dann kopiert. Auf die gleiche Weise wird bei den Zeitschriften ohne E-ISSN vorgegangen.
Schließlich werden mit dem Ant-Task <delete> die Dateien, die nicht mehr gebraucht werden, gelöscht. Dies sind die Ausgabedateien, die zu einer zusammengefügt wurden, sowie die Datei filelist.xml.
Dieser Prozess ist so konzipiert, dass beliebig viele Eingabedateien vorhanden sein können, um die Verwaltung der Feeds zu erleichtern, indem beispielsweise alle Zeitschriften eines Fachgebiets je in einer OPML-Datei gespeichert werden können. Die Zeitschriften müssen nur, damit sie mit den korrekten Angaben aus der EZB verbunden werden können, in grüne und gelbe Zeitschriften getrennt in OPML-

Dateien verwaltetet werden. Damit der Prozess auf sie angewendet werden kann, muss der Dateiname zusätzlich auf „_gruen.opml“ bzw. „_gelb.opml“ enden. In der Abbildung des Prozesses (Abbildung 4) und in der Build-Datei für Ant wird dies über ein Sternchen vor den genannten Endungen dargestellt bzw. realisiert.
Insgesamt können in diesem Prozess im Vergleich zu den anderen beiden Prozessen die meisten Probleme auftreten, die auch nicht bisher nicht gelöst werden konnten. Dies liegt an dem Vergleich der Zeitschriftentitel aus zwei verschiedenen Systemen. Wenn nämlich eine Zeitschrift in den Systemen unterschiedliche Titel hat, kann keine Verknüpfung der Angaben stattfinden. Beispielsweise kann schon die unterschiedliche Groß- bzw. Kleinschreibung eines Wortes aus dem Titel dazu führen, dass keine Verknüpfung der Titel stattfindet. Ebenso bereiten dem Finden von übereinstimmenden Titeln aus den Systemen auch Umlaute oder Sonderzeichen, wie das Et-Zeichen, Probleme. Zusätzlich können keine übereinstimmenden Titel gefunden werden, wenn die Titel unterschiedlich zusammengesetzt sind.
Manchmal gibt es jedoch auch genau das umgekehrte Problem, nämlich dass zwei oder mehr passende Zeitschriftentitel gefunden werden, weil die Titel gleich sind.

Wegen dieser Probleme muss also opml-ohne-ezb.xml auf fehlende oder überflüssige Verknüpfungen durchsucht werden und die Titel, sowie die E-ISSNs in opml-complete.xml eingefügt werden, bevor der nächste Prozess beginnen kann.
Am einfachsten kann dieses Ergänzen für die fehlenden Titel darüber erfolgen, dass in die Eingabedateien, d.h. die OPML-Dateien ein weiteres Attribut namens „title-ezb“ in die Zeitschriftenangabe eingefügt wird. In dem Attribut muss der ganze Titel mit allen Klammern und Sonderzeichen aus der jeweiligen EZB-Ausgabedatei (gelb.xml-ausgabe.xml und gruenx.xml-ausgabe.xml) stehen. Beispielsweise könnte das dann so aussehen:

> <outline type="rss" version="RSS1" text="ATMOSPHERE-OCEAN" title="ATMOSPHERE-OCEAN" *title-ezb="Atmosphere-Ocean (formerly: Atmosphere)"* … />

Der ursprüngliche Titel darf nicht ersetzt werden, weil er für einen späteren Prozess verwendet wird.
Damit nun mit dem ergänzten Titel die Angaben aus der EZB auch verknüpft werden, muss der Prozess noch einmal durchgeführt werden. Der hinzugefügte Titel wird im Prozess nun zum Parameter „opml-ezb“ und wird nach dem Durchlaufen der unterschiedlichen Templates mit dem ursprünglichen Titel aus der EZB verglichen,

weshalb es auch nötig ist, den Titel mit allen Sonderzeichen und Klammern in die OPML-Datei einzufügen.
Die fehlenden E-ISSNs können in die EZB-Ausgabedateien oder in die Ergebnisdatei opml-complete.xml eingefügt werden. Im letzten Fall müsste auch die Adresse des Feeds angegeben werden.
Allerdings muss dieser Prozess, wie bereits gesagt, nach dem ersten Durchlauf und dem zweiten Durchlauf, bei dem die fehlenden Angaben ergänzt wurden, nur noch dann durchgeführt werden, wenn beispielsweise neue Zeitschriften abonniert werden sollen.
Wenn man Abonnements oder einen zweiten gefundenen Titel entfernen will, ist es wahrscheinlich einfacher, die Zeitschrift bzw. die Daten des Titels in opml-complete.xml zu löschen, anstatt den ganzen Prozess noch einmal durchzuführen.

4.2.2. Anreicherung und Umwandlung der neusten Zeitschriftenartikel

Der folgende Prozess dient der Erzeugung einer Datei, die die neusten Artikel von Zeitschriften, die bei JournalTOCs abonniert werden und noch nicht in ALBERT integriert wurden, enthält.
Die Basis für diesen Prozess bildet die Ergebnisdatei opml-complete.xml aus dem vorherigen Prozess, d.h. es muss keine Datei mehr heruntergeladen werden, sondern nur noch der Prozess mit Ant über die Build-Datei start-prozess-journaltocs.xml angestoßen werden.
Mit dem Stylesheet urls.xsl werden die Internetadressen der Zeitschriftenartikel bei JournalTOCs aus opml-complete.xml extrahiert und in der Textdatei opml-complete.txt als eine Zeichenkette gespeichert. Hinter jeder der Adressen, außer bei der letzten Adresse, befindet sich ein #-Zeichen.
Nun wird diese Textdatei mittels des Ant-Befehls <loadfile> zu einer Variablen mit dem Namen „url“ gemacht. Das bedeutet, dass der ganze Inhalt der Datei als lange Zeichenkette durch „url“ repräsentiert wird.
Um die Feeds einzeln herunterladen zu lassen, wird ein <target> mit <foreach> aufgerufen:

```
<foreach target="hole-feeds" list="${url}" delimiter="#" param="einzelne-url"/>
```

Mit den Attributen „list“, „delimiter“ und „param“ kann die lange Zeichenkette aus dem Parameter „url“ in die einzelnen Internetadressen aufgeteilt werden. Das Attribut list beinhaltet dabei die Quelle für den neuen Parameter „einzelne-url“, für

den das <target> „hole-feeds“ aufgerufen wird. In „delimiter“ wird das Zeichen angegeben, mit dem die Bestandteile der Zeichenkette getrennt werden. In diesem Fall wurde für „delimiter“ das #-Zeichen gewählt, weil es in den Internetadressen nicht vorkommt und sich daher gut als Trennzeichen eignet.
Nun wird in dem <target> „hole-feeds“ für den jeweiligen Parameter „einzelne-url“ mittels der Ant-Funktion <get> die Datei von der Adresse, die sich in dem Parameter befindet, in das Verzeichnis feeds-neu heruntergeladen.
Das Herunterladen der Dateien ist nötig, weil Ant immer nur mit lokalen Dateien Transformationen durchführen kann, da es von seinem Basisverzeichnis aus arbeitet und die Internetadressen auch in diesem Basisverzeichnis suchen würde, wenn man sie als Ziel der Transformation angeben würde.
Nachdem alle Dateien heruntergeladen wurden, werden diese Dateien mit einem eigenen <target> mit dem Stylesheet journaltoc.xsl umgewandelt.
Da der Prozess in der Konzeptionsphase auch mehrfach auf die heruntergeladenen Dateien angewendet wurde, war es nötig mit einem exclude-Element das Muster „feeds-neu/*“, was so viel bedeutet wie, dass alle Dateien aus dem Ordner feeds-neu verwendet werden sollen, um die Ausgabedateien, die mit „-ausgabe.xml“ enden, zu reduzieren. Dies bedeutet, dass nur noch die heruntergeladenen Dateien, die über keine Dateiendung verfügen, mit dem Stylesheet bearbeitet werden sollen.

Bei der Transformation ist zu beachten, dass es Feed-Dateien bei JournalTOCs gibt, die nicht wohlgeformt sind, da sie ungültige Zeichen enthalten. Da nicht wohlgeformte Dateien nicht transformiert werden können, würde jede dieser Dateien zu einem Abbruch des ganzen Prozesses mit Ant führen. Aus diesem Grund wurde in der xslt-Anweisung über „failOnError="false"“ festgelegt, dass der Prozess weiter ausgeführt werden soll, wenn eine Datei nicht umgewandelt werden kann und so die Fehler quasi übersprungen werden sollen.
Das Stylesheet journaltoc.xsl wird einerseits dafür eingesetzt, die Feeds in die von ALBERT benötigte Struktur zu bringen und andererseits die Zeitschriften in den Feeds mit den Angaben aus der EZB anzureichern, die sich in der Ausgabedatei opml-complete.xml des vorherigen Prozesses befinden. Ebenso werden die Verlagsangaben mit Daten angereichert, die in der Datei publishers.xml vorhanden sind. Da diese Datei nicht im Prozess erstellt wird, ist sie in dem Schema zum Prozess (Abbildung 5) auch anders gekennzeichnet, als die anderen Dateien.

Da all diese Funktionen mit einem Stylesheet umgesetzt werden, ist dieses Stylesheet auch das umfangreichste der drei Prozesse.
Die Vergleiche mit den Zeitschriftentiteln bzw. Verlagsnamen werden im Prinzip genauso durchgeführt, wie der Vergleich der Zeitschriftentitel im vorhergehenden Prozess. Das heißt, zunächst wird ein Parameter mit dem Inhalt des Titels/ Namens aus der Feed-Datei in einem Template für die Zeitschriftenangabe definiert. Dann wird dieser Parameter an ein Template für das entsprechende Element in der Datei, die zum Vergleich genutzt wird und in der document( )-Funktion steht (opml-complete.xml für die Zeitschriftentitel und publishers.xml für die Verlagsangaben), übergeben. In diesem Template werden dann die ganze Zeitschrift/ der ganze Verlag und ihr/ sein jeweiliger Name auch zu Parametern, die bei einem erneuten Templateaufruf für den Parameter aus der Feed-Datei, d.h. den Titel der Zeitschrift oder den Verlagsnamen, weitergegeben werden.
Das dritte Template wird dann dazu genutzt, die beiden Titel- bzw. Namensparameter zu vergleichen und danach zu prüfen, ob der Parameter aus der Feed-Datei der aktuelle Knoten ist. Wird bei beiden Tests die Bedingung erfüllt, werden die Angaben aus der Vergleichsdatei mit Hilfe der Parameter in die Ausgabedatei integriert.
Außer den beiden Vergleichen enthält das Stylesheet auch zwei Schleifen, wie sie in den anderen Prozessen z.B. für die Entfernung von Sonderzeichen und zur Aufteilung von E-ISSN-Angaben aus einem in mehrere Elemente genutzt wurden. In diesem Stylesheet werden damit die „
“-Angaben, d.h. die Zeilenumbrüche, in bestimmten Textknoten entfernt.
Außerdem werden alle Autoren aus einem dc:creator-Element in je ein eigenes dc:creator-Element pro Autor gebracht.
Diese zweite Schleife, d.h. das sich selbst aufrufende Template, funktioniert nur über die Verwendung von Verlagsnamen, da die Verlage unterschiedliche Muster verwenden, um die Autorennamen voneinander zu trennen. Weil es sehr viele Verlage gibt, wurde die Trennung der Autoren nur für eine Auswahl von Verlagen exemplarisch umgesetzt. Die Verlage, für die es umgesetzt wurde, gehören größtenteils zu den Verlagen mit den meisten Zeitschriften, die bei JournalTOCs abonniert werden können. Für alle anderen Verlage werden alle Autorennamen auch weiterhin in einem Element ausgegeben werden, d.h. die Angabe aus der Feed-Datei übernommen.

Genauso abhängig von den Verlagsnamen ist die Anpassung der Datumsangaben an das Format yyyy-MM-dd, d.h. zum Beispiel 2011-11-25. Allerdings gibt es hier auch Muster, die verlagsunabhängig sind, d.h. bei mehreren Verlagen verwendet werden. Die Anpassung an die Struktur des Datums wird über mehrere Bedingungen, d.h. <xsl:choose>, und substring( )-Funktionen realisiert.

Beispielsweise gibt es Datumsangaben, die in der Feed-Datei so aussehen:

```
<dc:date>2011-10-17T21:00:39Z</dc:date>
```

Diese Angabe wird dadurch angepasst, dass

```
<xsl:when test="substring(.,5,1)='-' and substring(.,8,1)='-'">
    <xsl:value-of select="substring(.,1,10)"/>
</xsl:when>
```

ausgeführt wird. Das bedeutet, wenn das fünfte und das achte Zeichen der Datumsangabe Bindestriche sind, sollen die ersten zehn Zeichen der Zeichenkette, also im Falle des Beispiels „2011-10-17“, in die Ausgabe übernommen werden. Für die anderen Datumsangaben muss etwas komplexer vorgegangen werden, um sie in das gewünschte Format zu bringen.

Nach der Transformation aller wohlgeformten Zeitschriftenfeeds, wird die Datei filelist-feeds.xml mit der Ant-Anweisung <echo> erzeugt. Diese Datei enthält die Pfade zu den Ausgabedateien, d.h. den an ALBERT angepassten Feed-Dateien, und den Pfad zur Datei feeds-ids-complete.xml. Die Datei feeds-ids-complete.xml beinhaltet die Identifier aller Artikel, die sich bereits in ALBERT befinden. Sie wird später in dem Prozess noch um die neusten Identifier erweitert werden.

Wenn der Prozess zum ersten Mal durchgeführt wird, muss die Datei auch vorhanden sein und kann zum Beispiel so aussehen:

```
<?xml version="1.0" encoding="UTF-8"?>
<rowdata>
    <identifier>
        <test/>
    </identifier>
</rowdata>
```

Die Ausgabedateien und die Datei feeds-ids-complete.xml werden, genauso wie die Ausgabedateien aus dem OPML-Prozess über eine Transformation der Datei filelist-feeds.xml mit den Pfaden durch das Stylesheet zusammen.xsl zu einer Datei zusammengefügt, die dann feeds.xml heißt.

Dann wird in der Build-Datei für Ant eine Variable mit dem aktuellen Datum über

```
<tstamp><format property="TODAY-today" pattern="yyyy-MM-dd"/></tstamp>
```

erzeugt.

Diese Variable wird zur Benennung der Datei, die nur die aktuellsten Zeitschriftenartikel enthält, die noch nicht in ALBERT integriert wurden, verwendet. Die Benennung erfolgt nach dem Muster „feeds-${TODAY-today}.xml“, d.h. zum Beispiel feeds-2011-11-25.xml.
Diese Datei wird über die Umwandlung der Datei feeds.xml, die eine Kombination der in ALBERT bereits vorhandenen Identifier und der umgewandelten Zeitschriftenartikel darstellt, mit dem Stylesheet loesche-vorhandene-feeds.xsl erzeugt. Sie wird über die xslt-Anweisung in der Build-Datei, mit der sie erzeugt wird, im Ordner feeds-albert gespeichert.
Ebenfalls wird in dieser Anweisung die Datumsvariable aus Ant auch an das Stylesheet weitergegeben und als Datumsangabe im Element <dc:date> verwendet, wenn ein transformierter Artikel keine eigene Datumsangabe besitzt.
Im Stylesheet loesche-vorhandene-feeds.xsl wird eine key-Funktion benutzt, um die Artikel zu identifizieren, die bereits in ALBERT sind. Die anderen, noch nicht vorhandenen Artikel werden schließlich mit allen Kindelementen mit <xsl:copy-of> aus der Eingabedatei feeds.xml in die Ausgabedatei kopiert.
Da die key-Funktion nur innerhalb eines Dokuments, d.h. nicht zusammen mit der Funktion document( ) verwendet werden kann, mussten die umgewandelten Feed-Dateien und die Datei mit den vorhandenen Identifiern zu einer Datei vereinigt werden.
Bei der Definition des Schlüssels werden die Elemente, die ein Kindelement mit dem Identifier als Kindelement enthalten, aus der Datei mit den bereits in ALBERT vorhandenen Artikeln (feeds-ids-complete.xml) verwendet. Das dem Schlüsselelement untergeordnete Kindelement, das den Identifier als Inhalt hat, dient als Erkennungsmerkmal.
Beim Aufruf des Schlüssels werden die Elemente mit den Zeitschriftenartikelangaben, die gerade erst heruntergeladen wurden, darauf geprüft, ob ihr dc:identifier-Element dem Inhalt eines Schlüssels entspricht. Ist dies nicht der Fall, wird das Element mit den Artikelangaben kopiert und wenn kein dc:date-Element vorhanden ist, eins mit dem aktuellen Datum, das ja in der Build-Datei von Ant als Parameter definiert wurde, mit folgenden Anweisungen eingefügt.

```
<xsl:template match=”oai_dc:dc”>
<xsl:if test="not(key('vorhanden',dc:identifier))">
        <xsl:copy>
                <xsl:apply-templates select="child::*"/>
                <xsl:if test="not(dc:date)">
                        <dc:date><xsl:value-of select="$datum"/></dc:date>
                </xsl:if>
        </xsl:copy>
</xsl:if>
</xsl:template>
<xsl:template match="child::*"><xsl:copy-of select="."/></xsl:template>
```

Da nun die Datei feeds-ids-complete.xml immer alle Identifier enthalten soll, die in ALBERT bereits integriert wurden, müssen auch die Identifier der Zeitschriftenartikel, die gerade in die Datei mit dem aktuellen Datum im Namen im Ordner feeds-albert gespeichert wurden, in feeds-ids-complete.xml eingefügt werden.

Dazu wird die soeben erzeugte Ausgabedatei mit den neusten Artikeln mit einem Stylesheet namens feeds-ids.xsl in eine Datei, die nur noch die Identifier der Artikel enthält und feeds-ids-latest.xml heißt, umgewandelt.

Dann wird mit der echo-Anweisung von Ant wieder eine Datei mit den Pfaden zu den Dateien feeds-ids-latest.xml und feeds-ids-complete.xml erzeugt. Mit dieser Datei namens filelist-feeds-ids.xml werden dann mit dem gleichen Stylesheet wie immer, wenn Dateien zusammengefügt werden, nämlich zusammen.xsl, die beiden Dateien zu feeds-out.xml vereinigt.

Die Datei feeds-out.xml wird schließlich, weil sie sonst nicht die gleiche Struktur hätte, wie das Ausgangs-feeds-ids-complete.xml mit dem Stylesheet zusammen-ids.xsl bearbeitet.

Dieses Stylesheet macht nichts anderes, als alle identifier-Elemente, in die die test-Elemente mit den Adressen der Artikel eingebettet sind, zu den direkten Kindelementen des Wurzelelements zu machen. Dies geschieht, indem weitere Verschachtelungen durchlaufen werden und dann nur die ihnen untergeordneten identifier-Elemente mit test-Kindelementen in die Ausgabedatei übernommen werden. Diese Ausgabedatei wird auch feeds-ids-complete.xml genannt und ersetzt dadurch die ursprüngliche, gleichnamige Datei, sodass nun wieder alle Identifier von in ALBERT bereits enthaltenen Zeitschriftenartikeln in der Datei vorhanden sind.

Der nun folgende Rest des Prozesses dient nur noch dazu, nicht mehr benötigte Dateien zu löschen.

Zu diesen Dateien gehören auch die Feed-Dateien, für die eine Ausgabe erzeugt werden konnte. Um auch nur genau diese und nicht die nicht wohlgeformten und deshalb nicht transformierten Dateien zu löschen, wird mit echo-Anweisungen von Ant eine Datei mit Pfaden zu den Ausgabedateien der Feeds erzeugt, die filelist-feeds-complete.xml genannt wird.
Diese Datei wird mit dem Stylesheet loeschen-eingabe.xsl so bearbeitet, dass nur noch die Namensbestandteile der Ausgabedateien vor der Endung „-ausgabe.xml“, getrennt durch #-Zeichen, in eine Textdatei namens feeds-complete-today.txt gespeichert werden. Die erwähnten Namensbestandteile sind die Namen der Feed-Dateien, für die Ausgabedateien erzeugt wurden, also genau die Dateien, die gelöscht werden sollen.
Dann wird so vorgegangen wie bei dem automatischen Herunterladen der Feed-Dateien. Die Textdatei wird nämlich nun wieder zu einer Variablen mit Hilfe der loadfile-Anweisung von Ant.
Diese Variable namens „dateien“, wird mit einer foreach-Anweisung in die Namensbestandteile, für die im gleichen Schritt ein <target> namens „loesche-eingabe-mit-ausgabe“ aufgerufen wird, zerteilt. Zum Teilen der Namensbestandteile wird wieder das #-Zeichen genutzt. Damit für die Namensbestandteile das <target> „loesche-eingabe-mit-ausgabe“ ausgeführt werden kann, werden sie zu einem Parameter, der „einzelne-datei“ heißt.
Im <target> „loeschen-eingabe-mit-ausgabe“ wird dann einfach nur noch der Pfad zu den Dateien und dahinter der Parameter in einem delete-Task angegeben.

```
<delete file="feeds-neu/${einzelne-datei}"/>
```

Um nun alle Ausgabedateien der Feeds zu löschen, die nicht mehr benötigt werden, weil die neusten Artikel in einer eigenen Datei für ALBERT vorhanden sind, wird auch ein delete-Task verwendet. In diesem delete-Task befindet sich ein Muster für die Ausgabedateien:

```
<delete>
        <path>
                <fileset dir="feeds-neu" casesensitive="no">
                        <include name="*-ausgabe.xml"/>
                </fileset>
        </path>
</delete>
```

Hierbei wird im fileset-Element bereits angegeben, dass die Dateien aus dem Verzeichnis feeds-neu gelöscht werden sollen. In seinem Kindelement <include>

befindet sich das Muster, mit dem die Ausgabedateien, die alle auf „-ausgabe.xml“ enden, gelöscht werden.

Die übrigen Dateien, die für die Erstellung der gewünschten Ergebnisdateien als Zwischenschritte nötig waren, wie etwa die Dateien, die nur Pfadangaben enthalten, werden mit einem einfachen delete-Task, bei dem der Name der Datei im Attribut „file“ steht, zum Schluss des Prozesses gelöscht.

Um das Verständnis dieses längeren Prozesses mit zahlreichen Dateien noch etwas zu verbessern, kann die folgende Grafik, in der der Prozess schematisch dargestellt wird, betrachtet werden.

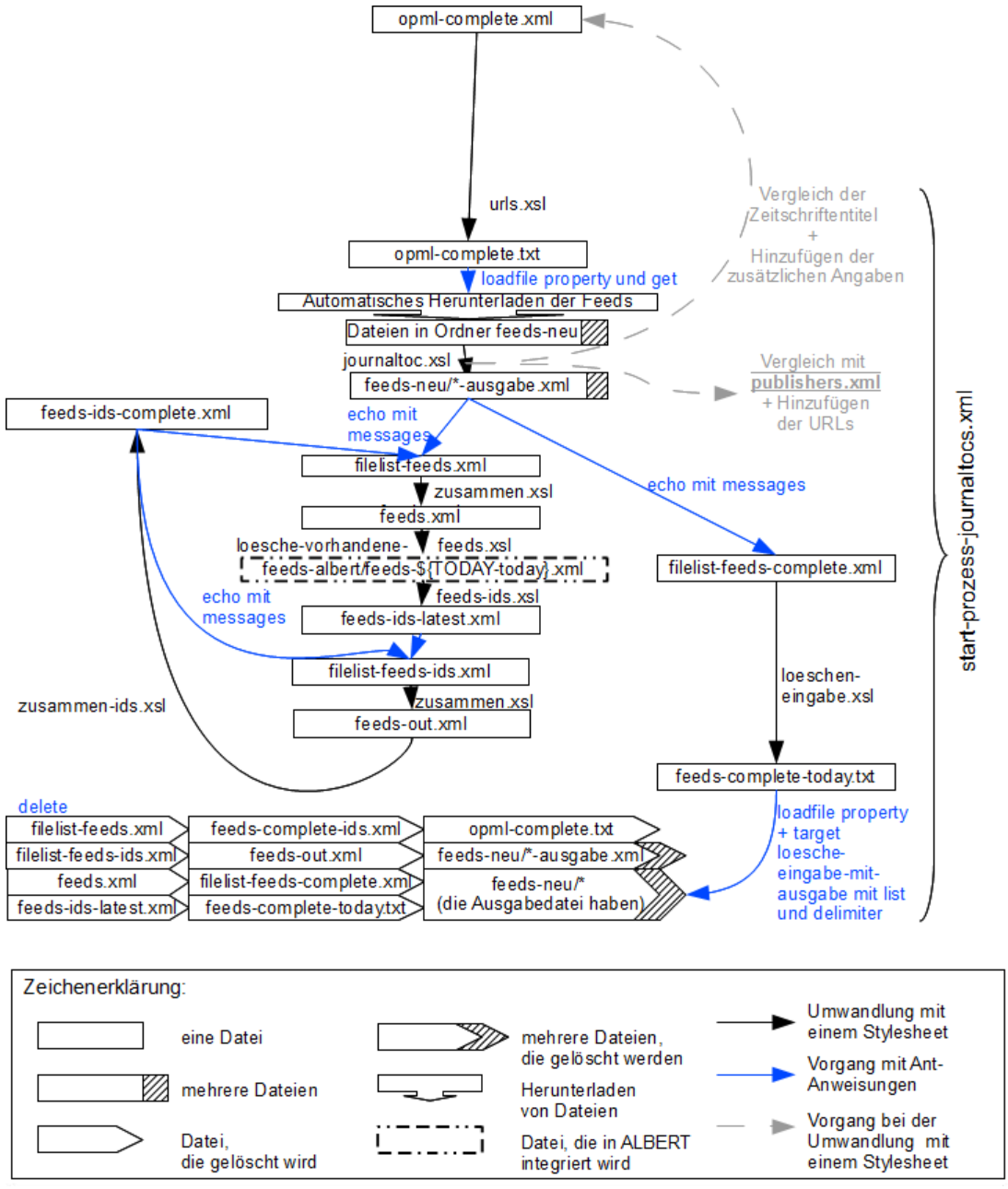

Abbildung 5: Schema des Prozesses für das automatische Herunterladen neuer Artikel aus JournalTOCs

Außerdem gibt es im Anhang einen Datensatz, wie er vor der Umwandlung in das Schema von ALBERT aussieht (Anhang 5) und wie er danach aussieht (Anhang 6), damit man sich das Ergebnis des Prozesses besser vorstellen kann.
Durch das Betrachten der beiden Versionen wird vor allem klar, dass viele Informationen aus dem ursprünglichen Datensatz nicht für ALBERT verwendet werden und dass neue, vorher nicht vorhandene Informationen eingefügt wurden.

Nach dem Durchführen des Prozesses befinden sich aber nicht alle Feeds, die heruntergeladen wurden auch in der Ausgabedatei, weil nur wohlgeformte Dateien transformiert werden können. Da sich jedoch nicht alle heruntergeladenen Zeitschrifteninhaltsverzeichnisse in wohlgeformten Dateien befinden, d.h. die Artikel aus diesen Zeitschriften noch nicht in ALBERT integriert wurden, sollten die fehlerhaften Zeichen aus den Dateien entfernt werden, um die Artikel aus diesen Dateien in ALBERT integrieren zu können.
Wenn man nach dem Entfernen der fehlerhaften Zeichen jedoch den ganzen, oben beschriebenen Prozess ablaufen lassen würde, um die nun wohlgeformten Dateien auch umzuwandeln, würden erstens alle Feed-Dateien erneut heruntergeladen und somit die ausgebesserten Dateien überschrieben und zweitens die Ergebnisdatei mit den neusten Zeitschriftenartikeln auch überschrieben, sodass diese Datei leer wäre.

Damit dies nicht passiert, gibt es eine weitere Build-Datei für Ant, die genau den gleichen Prozess hervorruft wie start-prozess-journaltocs.xml, außer dass die Ausgabedatei mit den neuen Artikeln beispielsweise feeds-2011-11-25-fehlerhafte.xml heißt, wobei auch hier immer das aktuelle Datum in den Dateinamen eingefügt wird.
Außerdem beginnt der Prozess nicht mit dem Herunterladen der Feed-Dateien, sondern mit ihrer Transformation in das Format von ALBERT.
Diese Build-Datei trägt den Namen start-prozess-journaltocs-fehlerhafte.xml.

5. Fazit

Obwohl die beschriebenen Prozesse noch ein paar Schwächen aufweisen, wie zum Beispiel, dass es nicht möglich ist, alle Zeitschriftenartikel automatisch mit den richtigen Angaben für die Zeitschriften zu versehen, können sie dennoch hilfreiche Instrumente für die schnelle Integration von Daten in die Suchmaschine ALBERT darstellen. Insbesondere weil die erstellten XSLT-Stylesheets und Anweisungsdateien für Ant relativ einfach veränderbar sind, sollte eine Nachnutzung bzw. Anpassung an die eigenen Bedürfnisse gut durchführbar sein.
Außerdem wurde versucht den Verwaltungsaufwand so gering bzw. anwender-freundlich wie möglich zu halten, indem beispielsweise bei der Verknüpfung der Zeitschriftentitel im zweiten Prozess eine Datei erzeugt wird, die die Titel ohne oder mit mehr als einer Verknüpfung, Titel ohne E-ISSN, für die deswegen nicht automatisch neue Artikel heruntergeladen werden können, und die Angabe der Herkunftsdatei des jeweiligen Titels enthält.
Zusätzlich gibt es in den Fällen, in denen aus Gründen der Übersichtlichkeit zunächst nicht alle möglichen Varianten umgesetzt wurden, immer Mechanismen, durch die die Fälle, die nicht beachtet werden konnten, dennoch für ALBERT nutzbar gemacht werden. Beispielsweise ist, wie bereits gesagt, im dritten Prozess nicht für alle Verlage eine Verteilung der Autoren in eigene Elemente umgesetzt worden. Wenn bei den Autoren keine Trennung durchgeführt wird, werden die Autoren so, wie sie in den ursprünglichen Dateien angegeben werden, auch in die Ausgabe übernommen, sodass kein Datenverlust stattfindet.
An dieser Stelle kann es jedoch zu Problemen kommen, wenn ein Verlag die Struktur bzw. das Muster nach dem die Autoren angegeben werden, verändert. In so einem Fall kann es nämlich, dadurch, dass es eine Verarbeitungsanweisung für den Verlag gibt, dazu kommen, dass die Autoren falsch getrennt oder gar nicht mehr in die Ausgabedatei übernommen werden. Allerdings wird dann meist ein leeres Autorenelement erzeugt.
Grundsätzlich sollten die fehlenden Varianten jedoch zukünftig noch bearbeitet werden und bei der Nutzung der Prozesse die bereits umgesetzten Varianten, zumindest was die Autorenangabe betrifft, auf Veränderungen beobachtet und dann ggf. abgeändert werden.

Gleichfalls sollte eine Funktion geschaffen werden, bei der das Starten der Prozesse nicht einzeln über die Eingabeaufforderung, sondern über eine gemeinsame Oberfläche durchgeführt und unter dieser auch auf die Dateien zugegriffen werden kann. Dies konnte allerdings nicht mit den derzeitigen Kenntnissen der Verfasserin durchgeführt werden.
Insgesamt gesehen können also mit den Prozessen nicht nur Daten aus einem Format in ein anderes Format umgewandelt werden, sondern die Daten auch mit zusätzlichen Informationen angereichert werden, die dem Suchenden bei seiner Recherche helfen können. Diese Hilfe besteht beispielsweise darin, dass die zusätzlichen Informationen, wie die Themengebiete aus der EZB, vor allem bei der Einschränkung von Suchergebnissen eingesetzt werden können, indem sie z.B. als Gruppierungsmöglichkeit verwendet werden.
Generell sollen die Prozesse also einerseits bei der Verwaltung der Informationen, d.h. der elektronischen Zeitschriften und der neusten Zeitschriftenartikel, hilfreich sein. Die Unterstützung soll darin liegen, dass die Informationen weiterhin in geeigneten Systemen gepflegt werden können und die Informationen relativ einfach mit den Prozessen in das Format für ALBERT gebracht werden können, um dort durchsucht werden zu können.
Andererseits sollen die Ergebnisse der Prozesse dem Suchenden nutzen, indem sie neue Möglichkeiten bieten, die beim Finden der gewünschten Informationen behilflich sein können.
Das bedeutet, dass die Prozesse die Trennung von Suchoberfläche und Verwaltungssystem fördern, die durch den Einsatz von Bibliothekssuchmaschinen hervorgerufen wird. [266]
Der Katalog behält demnach eine seiner beiden Kernfunktionen, nämlich das Verwalten von Informationen und gibt die andere, d.h. die Möglichkeit nach Informationen suchen zu können,[267] bzw. Informationen zugänglich zu machen, an die neuen Suchoberflächen ab. Hierdurch wird es leichter, die Suchoberfläche und das Verwaltungssystem immer besser an die an sie gestellten Anforderungen anzupassen. Zum Beispiel kann eine der beiden Komponenten verändert werden,

[266] Vgl. Coyle (2007b), S. 415 und Breeding (2010), S. 32.

[267] Vgl. Calhoun (2006), S. 31.

während die andere unverändert bleibt. Zusätzlich muss bei der Konzeption von neuen Funktionen in einer Komponente auch weniger Rücksicht auf die Anforderungen der anderen Komponente genommen werden.
Letztlich unterstützen die beschriebenen Prozesse also dadurch, dass Informationen, die aus verschiedenen Systemen kommen, verknüpft werden und unter einer Oberfläche zugänglich bzw. auffindbar gemacht werden können, den Auftrag, den ein Bibliothekskatalog hat, nämlich ein Instrument zu sein, das zum Nachweisen des in einer Bibliothek vorhandenen Wissens dienen soll.[268] Durch die Informationen, die mit den Prozessen gewonnen werden können, werden nicht nur vorhandene Informationen aus der Bibliothek in einer Oberfläche vereinigt, sondern auch Informationen, die nur implizit vorhanden waren, wie die ISSNs und Themengebiete von Zeitschriften aus der EZB, explizit bei den zugehörigen Datensätzen, d.h. im Beispiel bei den Artikeln der Zeitschriften, angegeben.
Außerdem ist im Zusammenhang mit dem Thema dieser Arbeit wichtig festzustellen, dass, obwohl es in der Regel die Nutzer einer Bibliothek nicht interessiert, welche Handlungen im Hintergrund einer Suchoberfläche durchgeführt werden und momentan viel über die Suchoberflächen an sich diskutiert wird, ein Katalog mehr als eine bloße Ansicht für die Nutzer ist.[269]
Natürlich ist die Ansicht für den Nutzer aber auch ein Instrument, das die Nutzung einer Bibliothek entscheidend beeinflusst, da sie einen Zugang zu den Angeboten der Bibliothek darstellt und die Nutzer nur diese Ansicht des Katalogs kennen. Außerdem wissen sie in der Regel nicht, welche Bedeutung der Katalog für die interne Bibliotheksarbeit hat und mit dem Einsatz einer Suchmaschine auch nicht verliert.
Da also die Suchoberfläche des Katalogs für die Nutzer ein zentrales Instrument ist, mit dem sie in einer Bibliothek konfrontiert werden und das auch zu der Entscheidung über die weitere Nutzung der Bibliothek beiträgt, versucht man die Suchoberflächen schließlich auch immer mehr an die Bedürfnisse und das Verhalten der Zielgruppen anzupassen.

---

[268] Vgl. Calhoun (2006), S. 7.

[269] Vgl. Casey (2007), S. 15.

Literaturverzeichnis

Apache Software Foundation (2011)

– Apache Software Foundation: Apache Ant. - http://ant.apache.org/. – zuletzt aktualisiert am: 11.05.2011, zuletzt geprüft am: 12.01.2012.

Bach (2000)

– Bach, Mike: XSL und XPath - verständlich und praxisnah : Transformation und Ausgabe von XML-Dokumenten mit XSL. München : Addison-Wesley, 2000. - (net.com).

Bailliez; et al. (2011)

– Bailliez, Stephane ; et al.: Apache Ant 1.8.2 Manual. Version 1.8.2, 2011. - http://ant.apache.org/manual/, zuletzt inklusive aller verwendeten Unterseiten geprüft am: 12.01.2012.

Bates (1986)

– Bates, Marcia J.: Subject access in online catalogs: A design model. In: Journal of the American Society for Information Science 37 (1986), Nr. 6, S. 357–376. - http://dx.doi.org/10.1002/(SICI)1097-4571(198611)37:6<357::AID-ASI1>3.0.CO;2-H, zuletzt geprüft am: 12.01.2012.

Bertelmann; et al. (2007)

– Bertelmann, Roland ; et. al.: Bibliothekssuchmaschine statt Bibliothekskatalog. In: Bibliotheksdienst 41 (2007), Nr. 12, S. 1302–1306. - http://www.zlb.de/aktivitaeten/bd_neu/heftinhalte2007/Erschliessung011207.pdf, zuletzt geprüft am: 12.01.2012.

Blenkle; Ellis; Haake (2009)

– Blenkle, Martin ; Ellis, Rachel ; Haake, Elmar: E-LIB Bremen – Automatische Empfehlungsdienste für Fachdatenbanken im Bibliothekskatalog / Metadatenpools als Wissensbasis für bestandsunabhängige Services. In: Bibliotheksdienst 43 (2009), Nr. 6, S. 618–625. - http://www.zlb.de/aktivitaeten/bd_neu/heftinhalte2009/Erschliessung010609BD.pdf, zuletzt geprüft am: 12.01.2012.

Breeding (2007)

– Breeding, Marshall: The Birth of a New Generation of Library Interfaces. In: Computers in Libraries 27 (2007), Nr. 9, S. 34–37.

Breeding (2009)

– Breeding, Marshall: Open Discovery Interfaces. In: American Libraries Magazine (2009), Nr. 6/7, S. 40. - http://americanlibrariesmagazine.org/columns/open-discovery-interfaces, zuletzt geprüft am: 12.01.2012.

Breeding (2010)

– Breeding, Marshall: The State of the Art in Library Discovery 2010. In: Computers in Libraries 30 (2010), Nr. 1/2, S. 31–35. - http://www.librarytechnology.org/ltg-displaytext.pl?RC=14574, zuletzt geprüft am: 12.01.2012.

Calhoun (2006)

– Calhoun, Karen: The Changing Nature of the Catalog and its Integration with Other Discovery Tools : Final Report. prepared for the Library of Congress. - 2006. - http://www.loc.gov/catdir/calhoun-report-final.pdf, zuletzt geprüft am: 12.01.2012.

Casey (2007)

– Casey, Michael: Looking toward catalog 2.0. In: Courtney, Nancy (Hrsg.): Library 2.0 and beyond : Innovative technologies and tomorrow's user. 1. Aufl. Westport, Conn : Libraries Unlimited, 2007, S. 15–24.

Christensen (2009)

– Christensen, Anne: Partizipative Entwicklung von Diensten in der Bibliothek 2.0 : Methoden und Ergebnisse aus Katalog-2.0-Projekten. In: Bibliotheksdienst 43 (2009), Nr. 5, S. 527–537. - http://www.zlb.de/aktivitaeten/bd_neu/heftinhalte2009/Erschliessung010509BD.pdf, zuletzt geprüft am: 12.01.2012.

Cole; et al. (2001)

– Cole, Timothy W. ; et al.: Using XML and XSLT to process and render online journals. In: Library Hi Tech 19 (2001), Nr. 3, S. 210–222. - http://dx.doi.org/10.1108/07378830110405067, zuletzt geprüft am: 12.01.2012.

Coyle (2007a)

– Coyle, Karen: The Library Catalog in a 2.0 World. In: The Journal of Academic Librarianship 33 (2007), Nr. 2, S. 289–291. - http://dx.doi.org/10.1016/j.acalib.2007.02.003, zuletzt geprüft am: 12.01.2012.

Coyle (2007b)

– Coyle, Karen: The Library Catalog: Some Possible Futures. In: The Journal of Academic Librarianship 33 (2007), Nr. 3, S. 414–416. - http://dx.doi.org/10.1016/j.acalib.2007.03.001, zuletzt geprüft am: 12.01.2012.

Dempsey (2006)

– Dempsey, Lorcan: The Library Catalogue in the New Discovery Environment: Some Thoughts. In: Ariadne (2006), Nr. 48. - http://www.ariadne.ac.uk/issue48/dempsey/.– zuletzt aktualisiert am: 14.08.2006, zuletzt geprüft am: 12.01.2012.

Garcia-Sanchez; et al. (2008)

– Garcia-Sanchez, P. ; et al.: Evolving XSLT Stylesheets for Document Transformation. In: Rudolph, Günter (Hrsg.): Parallel problem solving from nature - PPSN X : 10th international conference, Dortmund, Germany, September 13 - 17, 2008 ; proceedings. Berlin : Springer, 2008 (Lecture Notes in Computer Science, 5199), S. 1021–1030.

Herm; Volz (2008)

– Herm, Karin ; Volz, Sibylle: OPAC und mehr – Suchraumerweiterung für Bibliotheken : Die neue Library Search Engine der KOBV-Zentrale. 97. Deutscher Bibliothekartag. - 06.06.2008. - http://www.opus-bayern.de/bib-info/volltexte//2008/578/, zuletzt geprüft am: 12.01.2012.

Höhnow (2010)

– Höhnow, Tobias: Suchmaschine, ERM & Co.: Ressourcenmanagement im Backend des Bibliothekars. In: Mittermaier, Bernhard (Hrsg.): eLibrary - den Wandel gestalten : 5. Konferenz der Zentralbibliothek, Forschungszentrum Jülich, 8. - 10. November 2010 ; Proceedingsband. Jülich : Forschungszentrum Zentralbibliothek, 2010 (Reihe Bibliothek, 20), S. 163–176. - http://hdl.handle.net/2128/4296, zuletzt geprüft am: 12.01.2012.

Jackenkroll (2003)

– Jackenkroll, Melanie: Cross Media Publishing mittels XML : Die Enzyklopädie als Beispiel. - (Kölner Arbeitspapiere zur Bibliotheks- und Informationswissenschaft 35). - http://opus.bibl.fh-koeln.de/volltexte/2003/78/, zuletzt geprüft am: 12.01.2012.
  Zugl.: Köln, Fachhochschule, Diplomarbeit, 2003 u.d.T.: Nutzen von XML für die Herstellung verschiedener medialer Varianten von Informationsmitteln.

Keene (2011)

– Keene, Chris: Discovery services: next generation of searching scholarly information. In: Serials: The Journal for the Serials Community 24 (2011), Nr. 2, S. 193–196. - http://dx.doi.org/10.1629/24193, zuletzt geprüft am: 12.01.2012.

Keith (2004)

– Keith, Corey: Using XSLT to manipulate MARC metadata. In: Library Hi Tech 22 (2004), Nr. 2, S. 122–130. - http://dx.doi.org/10.1108/07378830410524549, zuletzt geprüft am: 12.01.2012.

Kneifel (2009)

– Kneifel, Fabienne: Mit Web 2.0 zum Online-Katalog der nächsten Generation : Innovationspreis 2009. Wiesbaden : Dinges & Frick, 2009. - (B.I.T. online : Innovativ 23). - http://www.b-i-t-online.de/daten/BIT_Innovativ_23_Kneifel.pdf, zuletzt geprüft am: 12.01.2012.
  Zugl.: Berlin, Humboldt-Univ., Masterarbeit, 2008 u.d.T.: Welche Funktionen und Inhalte sollte ein Bibliothekskatalog im Zeitalter des Web 2.0 bieten?

Kostädt (2008)

– Kostädt, Peter: Innovative Recherchemöglichkeiten in Katalogen und Bibliotheksportalen. In: Hutzler, Evelinde; Geißelmann, Friedrich (Hrsg.): Bibliotheken gestalten Zukunft : Kooperative Wege zur Digitalen Bibliothek; Dr. Friedrich Geißelmann zum 65. Geburtstag. Göttingen : Universitätsverl., 2008, S. 101–113.

Kränzler (2002)

– Kränzler, Christine: XML, XSL : Für Buch und Web. München : Markt-und-Technik-Verl., 2002. - (Digital studio one).

Larson (1991)

– Larson, Ray R.: Classification Clustering, Probabilistic Information Retrieval, and the Online Catalog. In: The library quarterly 61 (1991), Nr. 2, S. 133–173.

Lauber-Reymann (2010)

– Lauber-Reymann, Margrit: Informationsressourcen : Ein Handbuch für Bibliothekare und Informationsspezialisten. Berlin : de Gruyter, 2010. - (Bibliothekspraxis 42).

Lenz (2006)

– Lenz, Evan: XSLT 1.0 kurz gut. 1. Aufl. Beijing [u.a] : O'Reilly, 2006.

Lewandowski (2006)

– Lewandowski, Dirk: Suchmaschinen als Konkurrenten der Bibliothekskataloge : Wie Bibliotheken ihre Angebote durch Suchmaschinentechnologie attraktiver und durch Öffnung für die allgemeinen Suchmaschinen populärer machen können. In: Zeitschrift für Bibliothekswesen und Bibliographie 53 (2006), Nr. 2, S. 71–78.

Lewandowski (2010)

– Lewandowski, Dirk: Der OPAC als Suchmaschine. In: Bergmann, Julia; Danowski, Patrick (Hrsg.): Handbuch Bibliothek 2.0. Online-Ausg. Berlin : de Gruyter Saur, 2010 (Bibliothekspraxis), S. 87–108. - http://hdl.handle.net/10760/16087, zuletzt geprüft am: 12.01.2012.

Li; Mani; Rundensteiner (2008)

– Li, Ming ; Mani, Murali ; Rundensteiner, Elke A.: Constraint-Aware XSLT Evaluation. In: Li, Qing; et al. (Hrsg.): Conceptual modeling - ER 2008: 27th International Conference on Conceptual Modeling, Barcelona, Spain, October 20-24, 2008 ; proceedings. Berlin [u.a.] : Springer, 2008 (Lecture Notes in Computer Science, 5231), S. 524–525.

Lossau (2004)

– Lossau, Norbert: Suchmaschinentechnologie und Digitale Bibliotheken – Bibliotheken müssen das wissenschaftliche Internet erschließen. In: Zeitschrift für Bibliothekswesen und Bibliographie 51 (2004), Nr. 5-6, S. 284–294.

Markey (2007)

– Markey, Karen: The Online Library Catalog : Paradise Lost and Paradise Regained? In: D-lib magazine 13 (2007), Nr. 1/2. - http://www.dlib.org/dlib/january07/markey/01markey.html, zuletzt geprüft am: 12.01.2012.

Montero Pineda (2004)

– Montero Pineda, Manuel; Krüger, Manfred (Mitarb.): XSL-FO in der Praxis : XML-Verarbeitung für PDF und Druck. 1. Aufl. Heidelberg : dpunkt-Verl., 2004. - (XML.Bibliothek).

Potter (1989)

– Potter, William Gray: Expanding the Online Catalog. In: Information Technology and Libraries 8 (1989), Nr. 2, S. 99–104.

Schmitt; Stehle (2010)

– Schmitt, Jörg ; Stehle, Marcel: Der OPAC aus dem Baukasten : Realisierung eines Katalog 2.0 unter Einbeziehung der Community. überarbeitete Version 1.0. - 31.12.2010. - http://opus.haw-hamburg.de/volltexte/2011/1143/, zuletzt geprüft am: 12.01.2012.
Hamburg, Hochschule für Angewandte Wissenschaften, Bachelorarbeit, 2010.

Soules (2010)

– Soules, Aline: New e-sources, new models: reinventing library approaches to providing access. In: Library Hi Tech News 27 (2010), Nr. 2, S. 10–14. - http://dx.doi.org/10.1108/07419051011050420, zuletzt geprüft am: 12.01.2012.

Steiner (2007)

– Steiner, Esther Susanne: OPAC 2.0 : Mit Web 2.0-Technologie zum Bibliothekskatalog der Zukunft? - http://nbn-resolving.de/urn:nbn:de:bsz:900-opus-6245, zuletzt geprüft am: 12.01.2012.
Stuttgart, Hochschule der Medien, Bachelorarbeit, 2007.

Tennant (2003)

– Tennant, Roy: Library Catalogs: The Wrong Solution. In: Library journal (2003), Nr. 3. - http://www.libraryjournal.com/article/CA273959.html, zuletzt geprüft am: 12.01.2012.

Tröger (1999)

– Tröger, Beate: Bücher, Bytes und Bibliotheken: Integrierte Information im Internet : 4. InetBib-Tagung in Oldenburg. In: Bibliotheksdienst 33 (1999), Nr. 4, S. 648–656. - http://bibliotheksdienst.zlb.de/1999/1999_04_Informationsvermittlung01.pdf, zuletzt geprüft am: 12.01.2012.

Vonhoegen (2009)

– Vonhoegen, Helmut: Einstieg in XML : [Grundlagen Praxis Referenz ; für Entwickler und XML-Einsteiger ; Formatierung Transformation Schnittstellen XML Schema DTD XSLT 1.02.0 XPath 1.02.0 DOM SAX SOAP Open XML]. 5. Aufl. Bonn : Galileo Press, 2009. - (Galileo computing).

Wiesenmüller (2008)

– Wiesenmüller, Heidrun: Neue Trends und alte Desiderate: Der OPAC der Zukunft (Teil 2). In: VDB-Mitteilungen (2008), Nr. 1, S. 27–29. - http://www.vdb-online.org/publikationen/vdb-mitteilungen/vdb-mitteilungen-2008-1.pdf#page=27, zuletzt geprüft am: 12.01.2012.

Yee; Beaubien (2004)

– Yee, Raymond ; Beaubien, Rick: A preliminary crosswalk from METS to IMS content packaging. In: Library Hi Tech 22 (2004), Nr. 1, S. 69–81. - http://dx.doi.org/10.1108/07378830410524512, zuletzt geprüft am: 12.01.2012.

Anhang

1. Grüne Zeitschrift von der Elektronischen Zeitschriftenbibliothek

```
<row>
        <EZB-Id>46473</EZB-Id>
        <Titel>Bioinformatics (1996 - älter als 12 Monate)</Titel>
        <Ampelfarbe>1</Ampelfarbe>
        <Verlag>Oxford University Press ; HighWire Press</Verlag>
        <Fach>Medizin;Informatik;Biologie</Fach>
        <E-ISSN>1367-4811;1460-2059</E-ISSN>
        <P-ISSN>1367-4803</P-ISSN>
        <ZDB-Nummer>1468345-3</ZDB-Nummer>
        <FrontdoorURL>http://ezb.uni-
        regensburg.de/?1468345&amp;amp;bibid=GFZPO</FrontdoorURL>
        <Typ>Volltext, Online und Druckausgabe</Typ>
        <Preistyp>kostenlos</Preistyp>
        <Zugangsbedingung>frei zug&amp;amp;auml;nglich</Zugangsbedingung>
        <Link_zur_Zeitschrift>http://bioinformatics.oxfordjournals.org/</Link_zur_Zeitschr
        ift>
        <anchor/>
        <erstes_Jahr>1996</erstes_Jahr>
        <erstes_volume>12</erstes_volume>
        <erstes_issue/>
        <letztes_Jahr/>
        <letztes_volume/>
        <letztes_issue/>
        <moving_wall>-12M</moving_wall>
        <verfuegbar/>
 </row>
```

2. Umgewandelte grüne Zeitschrift im Format von ALBERT

```
<record>
        <feld nr="00" lb="IdNr">46473</feld>
        <feld nr="11h" lb="frei">zs;ezb</feld>
        <feld nr="20" lb="Titel">Bioinformatics (1996 - älter als 12 Monate)</feld>
        <feld nr="94b" lb="">http://bioinformatics.oxfordjournals.org/</feld>
        <eissns>
                <eissn>1367-4811</eissn>
                <eissn>1460-2059</eissn>
        </eissns>
        <holdings>
                <from>12.1996</from>
                <until>nicht die letzten 12 Monate</until>
        </holdings>
        <publisher>Oxford University Press ; HighWire Press</publisher>
        <topics>
                <topic>WW-YZ</topic>
                <topic>SQ-SU</topic>
                <topic>W</topic>
        </topics>
</record>
```

3. Gelbe Zeitschrift von der Elektronischen Zeitschriftenbibliothek

```
<journal>
        <ezbid>1</ezbid>
        <title><![CDATA[Journal of Molecular Modeling]]></title>
        <publisher><![CDATA[Springer]]></publisher>
        <zdbid>1284729-x</zdbid>
        <eissn>0948-5023</eissn>
        <subject>V</subject>
        <licenseinfo>
                <first_volume>3</first_volume>
                <first_date>1997</first_date>
                <anchor><![CDATA[Springer]]></anchor>
                <url>http%3A%2F%2Fwww.springerlink.com%2Flink.asp%3Fid%3D1005
                29</url>
        </licenseinfo>
        <licenseinfo>
                <first_volume>1</first_volume>
                <first_date>1995</first_date>
                <last_volume>8</last_volume>
                <last_date>2002</last_date>
                <anchor><![CDATA[natli_springer]]></anchor>
                <url>http%3A%2F%2Fwww.springerlink.com%2Fopenurl.asp%3Fgenre%3
                Djournal%26issn%3D1610-2940</url>
        </licenseinfo>
</journal>
```

4. Umgewandelte gelbe Zeitschrift im Format von ALBERT

```
<record>
        <feld nr="00" lb="IdNr">1</feld>
        <feld nr="11h" lb="frei">zs;ezb</feld>
        <feld nr="20" lb="Titel">Journal of Molecular Modeling</feld>
        <feld nr="94b" lb="">http://www.springerlink.com/link.asp?id=100529</feld>
        <eissns>
                <eissn>0948-5023</eissn>
        </eissns>
        <holdings>
                <holding type="online">
                        <from>1.1995</from>
                </holding>
        </holdings>
        <publisher>Springer</publisher>
        <topics>
                <topic>V</topic>
        </topics>
</record>
```

5. Artikel von JournalTOCs

<item
rdf:about="http://www.sciencedirect.com/science?_ob=GatewayURL&_origin=IRSSCONTENT&_method=citationSearch&_piikey=S1040618211005842&_version=1&md5=7dd9c1743c27972422ab3e8c5146a71e">
<title>Landscape Archaeology at the LAC2010 conference</title>
<link>
http://www.sciencedirect.com/science?_ob=GatewayURL&_origin=IRSSCONTENT&_method=citationSearch&_piikey=S1040618211005842&_version=1&md5=7dd9c1743c27972422ab3e8c5146a71e
</link>
<description>
Publication year: 2011
 Source: Quaternary International, Available online 15 October 2011
 Sjoerd J. Kluiving, Frank Lehmkuhl, Brigitta Schütt
</description>
<dc:identifier>
http://www.sciencedirect.com/science?_ob=GatewayURL&_origin=IRSSCONTENT&_method=citationSearch&_piikey=S1040618211005842&_version=1&md5=7dd9c1743c27972422ab3e8c5146a71e
</dc:identifier>
<dc:creator>
Sjoerd J. Kluiving, Frank Lehmkuhl Brigitta Schütt
</dc:creator>
<dc:date>2011-10-17T21:00:39Z</dc:date>
<dc:source>Quaternary International, Vol. , No. (2011) pp. - </dc:source>
<dc:publisher>Elsevier</dc:publisher>
<prism:PublicationName>Quaternary International</prism:PublicationName>
<prism:publicationDate>2011-10-17T21:00:39Z</prism:publicationDate>
<content:encoded>
<![CDATA[<a href="http://www.sciencedirect.com/science?_ob=GatewayURL&_origin=IRSSCONTENT&_method=citationSearch&_piikey=S1040618211005842&_version=1&md5=7dd9c1743c27972422ab3e8c5146a71e">Landscape Archaeology at the LAC2010 conference</A>
Sjoerd J. Kluiving, Frank Lehmkuhl Brigitta Schütt
Quaternary International, Vol. , No. (2011) pp. -
Publication year: 2011
Source: Quaternary International, Available online 15 October 2011
Sjoerd J. Kluiving, Frank Lehmkuhl, Brigitta Schütt]]>
</content:encoded>
</item>

6. Umgewandelter Artikel von JournalTOCs im Format von ALBERT


```
<oai_dc:dc>
	<dc:identifier xmlns="http://purl.org/rss/1.0/">
		http://www.sciencedirect.com/science?_ob=GatewayURL&amp;_origin=IR
		SSCONTENT&amp;_method=citationSearch&amp;_piikey=S10406182110
		05842&amp;_version=1&amp;md5=7dd9c1743c27972422ab3e8c5146a71e
	</dc:identifier>
	<dc:date>2011-10-17</dc:date>
	<dc:title>Landscape Archaeology at the LAC2010 conference</dc:title>
	<dc:creator>Sjoerd J.&amp;#160;Kluiving</dc:creator>
	<dc:creator>Frank&amp;#160;Lehmkuhl</dc:creator>
	<dc:creator>Brigitta&amp;#160;Sch&amp;#252;tt</dc:creator>
	<dc:description>Publication year: 2011. Source: Quaternary International, Available
	online 15 October 2011. Sjoerd J. Kluiving, Frank Lehmkuhl,
	Brigitta Schütt</dc:description>
	<dc:source>Quaternary International, Vol. , No.  (2011) pp.  - </dc:source>
	<wae xmlns:wae="http://localhost/XML/LSE">
		<wae:topic>TE-TZ</wae:topic>
		<wae:topic>R</wae:topic>
		<wae:eissn>1040-6182</wae:eissn>
		<wae:eissn>1873-4553</wae:eissn>
		<wae:ezbID>6576</wae:ezbID>
		<wae:publisher>Elsevier</wae:publisher>
		<wae:publisherURL>http://www.sciencedirect.com</wae:publisherURL>
	</wae>
	<dc:relation>
		http://www.sciencedirect.com/science?_ob=GatewayURL&amp;_origin=IR
		SSCONTENT&amp;_method=citationSearch&amp;_piikey=S10406182110
		05842&amp;_version=1&amp;md5=7dd9c1743c27972422ab3e8c5146a71e
	</dc:relation>
</oai_dc:dc>
```

Eidesstattliche Erklärung

Ich versichere hiermit an Eides statt, dass ich die vorliegende Bachelorarbeit mit dem Titel „XSL Transformationen als Vorbereitung für die Bibliothekssuchmaschine ALBERT“ selbstständig verfasst und hierzu keine anderen als die angegebenen Hilfsmittel verwendet habe. Alle Stellen der Arbeit, die wörtlich oder sinngemäß aus fremden Quellen entnommen wurden, sind als solche kenntlich gemacht.
Die Arbeit wurde bisher in gleicher oder ähnlicher Form in keinem anderen Studiengang als Prüfungsleistung vorgelegt oder an anderer Stelle veröffentlicht. Ich bin mir bewusst, dass eine falsche Erklärung rechtliche Folgen haben wird.

_____

(Ort, Datum, Unterschrift)