Eberhard Karls Universität Tübingen

Mathematisch-Naturwissenschaftliche Fakultät

Wilhelm-Schickard-Institut für Informatik

Masterarbeit Informatik

Konzeption und Implementation des Aufbaus eines Suchmaschinenindexes für Solr im bibliothekarischen Kontext.

Oliver Obenland

27. Februar 2017

Erstgutachter

Prof. Dr. Thomas Walter

Direktor des ZDV

Universität Tübingen

Zweitgutachter

Prof. Dr. Wolfgang Küchlin

Wilhelm-Schickard-Institut für Informatik

Universität Tübingen

Zusammenfassung

Diese Arbeit befasst sich mit der Struktur und den Betrieb von VuFind, das eine bibliographische Suchmaschine darstellt. VuFind wird als Open-Source-Projekt entwickelt und mittlerweile von mehreren hundert Bibliotheken weltweit genutzt. Außerdem wird SolrMarc, ein Teilprojekt von VuFind, analysiert und überarbeitet, wodurch die Effizienz dieses Programmes erhöht wird.

Selbständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Masterarbeit selbständig und nur mit den angegebenen Hilfsmitteln angefertigt habe und dass alle Stellen, die dem Wortlaut oder dem Sinne nach anderen Werken entnommen sind, durch Angaben von Quellen als Entlehnung kenntlich gemacht worden sind. Diese Masterarbeit wurde in gleicher oder ähnlicher Form in keinem anderen Studiengang als Prüfungsleistung vorgelegt.

Oliver Obenland

Kirchentellinsfurt, 26.02.2017

Inhaltsverzeichnis

1 Einführung

Zwischen 1950 und 2014 stieg die Anzahl von Buchveröffentlichung von etwa 14.000 jährlich auf 87.000 pro Jahr[ddB]. Dies ist darauf zurückzuführen, dass wir als Gesellschaft einen technischen Wandel vollzogen haben. Während früher Bücher zur Vervielfältigung handschriftlich abgeschrieben werden mussten, führte der Buchdruck zu einer technischen und gesellschaftlichen Revolution, wodurch sich die Verfügbarkeit von Büchern nicht mehr auf akademische Kreise beschränkte, und die weite Verbreitung des persönlichen Computers zu einer schnelleren Publikation von Büchern führt.

Dieser Wandel vollzog sich auch im bibliothekarischen Umfeld. Da die Anzahl von Publikationen die Pflegbarkeit hinsichtlich Zeit- und Platzbedarf von Kartenkatalogen übersteigt übernimmt der Computer heutzutage diese Aufgabe. So wurden Datenbanken genutzt um den Inhalt der Zettelkataloge zu digitalisieren. Diese sind platzsparender und einfacher nach verschiedenen Kriterien zu durchsuchen.

Aber auch die Softwaresystem des Computers werden stetig überarbeitet und müssen den Ansprüchen des Zeitgeistes genügen. Die einzelnen Datenbanken wurden zu integrierten Bibliothekssystemen erweitert, die nicht nur zur Verwaltung der bibliographischen Informationen genutzt werden können. Diese Systeme umfassten fast alle Aufgaben des bibliothekarischen Alltags. So können unter anderem bibliographische Daten gepflegt und ausgetauscht, aber auch Aus- und Fernleihservices angeboten werden. Auch das Durchsuchen des Bestandes über ein Webinterface ist Teil dieses Systems. Somit stellten integrierte Bibliothekssysteme eine monolithische Komplettlösung dar.

Inzwischen werden spezialisierte Applikationen verwendet, die einen Teilbereich des integrierten Bibliothekssystems abbilden und mit bestehenden Lösungen verknüpft werden können. Die so gewonnene Freiheit führt zu größerem Wettbewerb und einer breiteren Varianz an Produkten. Gleichzeitig entstanden Projekte durch die Open-Source-Bewegung, die kostenlos genutzt und angepasst werden können. Diese Projekte werden meist von einer Bibliothek direkt betrieben und durch die Mithilfe von Mitarbeitern anderer Bibliotheken unterstützt. Die so entehrenden Communities bündeln dabei das bibliothekarische Fachwissen mit der technischen Kompetenz.

Traditionelle integrierte Bibliothekssysteme sind meist kommerzielle Produkte. Der Erwerb eines solchen Systems bedeutet nicht nur ein hohen finanziellen Aufwand, sondern auch eine langjährige Bindung an das System und den Support des Herstellers. Bei Open-Source-Lösungen hingegen entstehen Kosten nur durch eigene Entwicklungsarbeiten.

In dieser Arbeit wird anhand des Open-Source-Projekts VuFind die Struktur einer bibliographischen Suchmaschine aufgezeigt. Außerdem wird der Betrieb und die Einflüsse von VuFind auf die Arbeitsprozesse einer Bibliothek näher beleuchtet. Dabei wird ein Teilprojekt von VuFind genau analysiert und überarbeitet, wodurch die Anforderungen an den Betrieb reduziert werden.

Abgrenzung

Diese Arbeit bezieht sich vorwiegend auf Version 2.5.4 von VuFind, wobei ein Rückblick auf die ersten Versionen und ein Ausblick auf Version 3.0.0 von VuFind gegeben wird. Die jeweiligen Versionen von VuFind können sich stark unterscheiden und werden oft mit unterschiedlichen Versionen von Drittanbieter-Software ausgeliefert.

Neuerungen, die während dieser Arbeit entstanden oder veröffentlicht wurden sind, finden teilweise in dieser Arbeit Erwähnung, sind jedoch nicht Teil dieser Arbeit.

2 VuFind

VuFind ist ein Projekt, das im Jahr 2007 von der Universitätsbibliothek Villanova, USA, unter dem Namen ViewFind begonnen wurde. Noch im selben Jahr wurde es auf SourceForge.net der Öffentlichkeit unter dem Namen VuFind als Open-Source-Projekt zugänglich gemacht.

VuFind ist eine Web-Oberfläche zum Durchsuchen von großen Datenbeständen, die auf bibliographische Daten spezialisiert ist. Durch die Nutzung von Apache Solr verbindet es die bibliographische Datenbank, die seit den 1980er-Jahren die Zettelkataloge ersetzt hat, mit moderner modularer Web-Technologie in einer Web-Applikation, wobei bestehende Kataloge in das System integriert werden können, um ein reibungsloses Zusammenarbeiten mit etablierten Lösungen zu gewährleisten.

Eine VuFind-Installation besteht aus verschiedenen Softwarekomponenten, die wiederum aus verschiedenen Open-Source-Programmen bestehen. Die zentrale Komponente bildet VuFind selbst, das von einem PHP-Webserver, wie Apache, ausgeführt wird. Dieses bekommt die bibliographischen Informationen von Solr, das in einen Java-Servlet-Container-Webserver, wie Jetty, läuft. Außerdem legt VuFind Daten in einer Datenbank ab, die von einem Datenbanksystem, wie MySQL, verwaltet wird. SolrMarc ist ein Java-Programm zum importieren und indizieren von Marc-Daten. Dieses Programm wird mit VuFind ausgeliefert. Ein Browser wird benötigt um die Informationen abzurufen.

2.1 Funktionalitäten von VuFind

Von Haus aus werden viele Funktionalitäten mit ausgeliefert, die große Bereiche der Suche von bibliographischen Daten abdecken, beziehungsweise den Umgang mit VuFind erleichtern. Diese werden von der Community um VuFind mit der Zeit stetig ausgebaut und erweitert.

Installation

Als ersten Schritt gilt es, VuFind so zu konfigurieren, dass es in Betrieb genommen werden kann. Dabei hilf ein Installationsskript, das mit der Online-Dokumentation komplementiert wurde. Dabei wird abgefragt, über welche Domain die VuFind-Instanz erreicht werden kann, in welchem Verzeichnis Anpassungen von Konfigurationsdateien und der Cache gespeichert werden und ob ein neues Modul angelegt werden soll.

Bis zur Version 3.0.0 war es mit dem Installationsskript nicht möglich, VuFind so zu konfigurieren, dass es direkt über die Domain, ohne weitere Angabe eines Pfades, erreichbar ist.

Jedoch konnte man dieses Verhalten nachträglich anpassen. Dazu hat man einen Namen für das Unterverzeichnis angegeben, der sonst nicht in dem Projekt vorkommt, und hat nach der Installation alle Vorkommen dieses Namens händisch gelöscht.

Suche

Die Suche von Monographien, Aufsätzen, Zeitschriften oder Ähnlichem ist die Hauptaufgabe von VuFind. Diese Komponente ist somit der wohl umfangreichste und komplexeste Teil. Sie ist durch viele Konfigurationsdateien stark an die individuellen Bedürfnisse anpassbar. Dabei wird eine einfache Suche, eine erweiterte Suche und eine Autorensuche angeboten. Es können Suchergebnisse durch Facettierung (siehe Kapitel 2.1) eingeschränkt werden, um das gesuchte Buch oder den gesuchten Artikel schneller zu finden.

Je nach Suche werden dabei unterschiedliche Felder des Solr-Indexes durchsucht. Diese sind durch die Art der Suche vorgegeben oder können in der Suchanfrage angegeben werden. Die Ergebnisse einer Suche werden dann nach unterschiedlichen Kriterien angeordnet. Zum Beispiel kann nach Erscheinungsdatum oder alphabetisch sortiert werden, aber auch eine Anordnung nach Relevanz ist möglich. Die Relevanz ist jedoch ein sehr unscharfer Begriff und wird in VuFind mit einem Scoring dargestellt. Das Scoring wird aus der Anzahl der Treffer innerhalb eines Feldes des Dokuments und je einem zu dem Feld gehörenden Gewichtungsfaktor berechnet. Diese Gewichte können in einer Konfigurationsdatei (search.yaml) angepasst werden. Jedoch führt diese Art der Relevanz-Suche zu unvorhersehbaren Anordnungen. Daher müssen die Gewichtungen mit viel Fingerspitzengefühl, Wissen über den Datenbestand und Ausprobieren angepasst werden.

Eine weitere Konfigurierung betrifft die Unschärfe einer Suche. Damit ist gemeint, wie viele Terme einer Suchanfrage in einem Treffer gefunden werden müssen. Dies ist eine Abwägung zwischen einerseits zu vielen und andererseits zu wenigen Resultaten zu einer Suche.

Einfache Suche

Auf jeder Seite, die VuFind standardmäßig ausliefert, gibt es einen Suchschlitz, wie in Abbildung 2.1 zu sehen. Dieses Eingabefeld kann genutzt werden, um mit einzelnen Suchtermen, Phrasen, aber auch syntaktisch komplexeren Ausdrücken eine Suche zu beginnen.

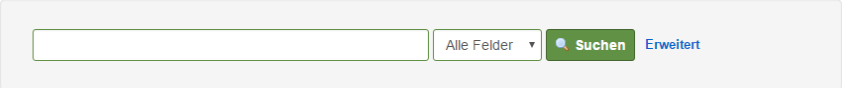


Abbildung 2.1: Suchschlitz
Screenshot: http://vufind.org/demo/Search/Advanced, 24.06.16

Es werden einfache logische Operatoren unterstützt, welche die einfache Suche zur

einer Expertensuche erweitern.

So kann ein `AND` zwischen zwei Suchterme gestellt werden, um nur Resultate zu bekommen, in denen beide Terme vorkommen. Zum Beispiel wird man bei der Suche nach `Java` wohl Ergebnisse für die Insel, aber auch für die Programmiersprache bekommen. Durch die Ergänzung zu `Java AND Programmiersprache` wird man nur Werke über die Programmiersprache Java angezeigt bekommen. Zu bedenken ist, dass nicht jedes Werk, das sich mit der Programmiersprache Java auseinandersetzt, auch mit dem Schlagwort `Programmiersprache` versehen ist.

Durch ein `NOT` oder ein Minuszeichen vor einem Term werden Resultate unterdrückt, die diesen Term enthalten. Somit könnte man auch nach `Java -Insel` oder `Java NOT Insel` suchen um ähnliche Resultate zu bekommen. Jedoch wird dabei unter anderem das Standardwerk `Java ist auch nur eine Insel` von `Rheinwerk Computing` unterdrückt.

Auch ein `OR` kann man angeben, das mit Klammerung sehr mächtig ist. Die beiden obigen Suchanfragen können so kombiniert werden:
`(Java AND Programmiersprache) OR (Java AND -Insel)`

Als letztes können noch einzelne Felder des Solr-Indexes direkt abgefragt werden. Durch `titel:Java` oder `author:Einstein` kann man die Suche verfeinern. Allerdings ist ein Spezialwissen über die Struktur des Indexes vonnöten, das jedoch nicht jeder Benutzer hat.

Erweiterte Suche

Um die Nutzung von logischen Operatoren und die Nennung von zu durchsuchenden Feldern zu vereinfachen, wurde die erweiterte Suche eingeführt. Sie besteht aus zwei Bereichen. Einmal, wie in Abbildung 2.2 zu sehen, ein dynamisch erweiterbares Formular, mit dem die Suchterme, die zu durchsuchenden Felder und deren Beziehung beschrieben werden können.

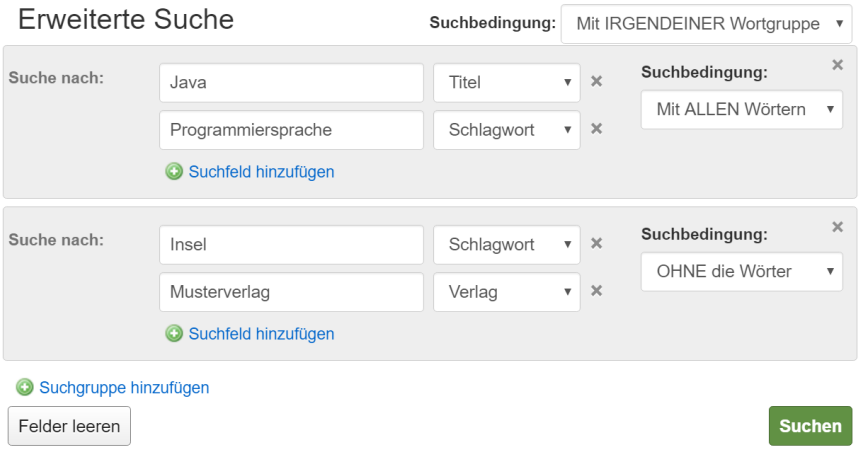


Abbildung 2.2: Erweiterte Suche - Terme.
Screenshot: http://vufind.org/demo/Search/Advanced, 24.06.16

Sowie eine Auswahl an Facetten, die zur groben Eingrenzung genutzt werden können. Dabei ist es nicht notwendig, alle Facetten zu nutzen. In Abbildung 2.3 sieht man zum Beispiel, dass keine Signatur ausgewählt wurde.

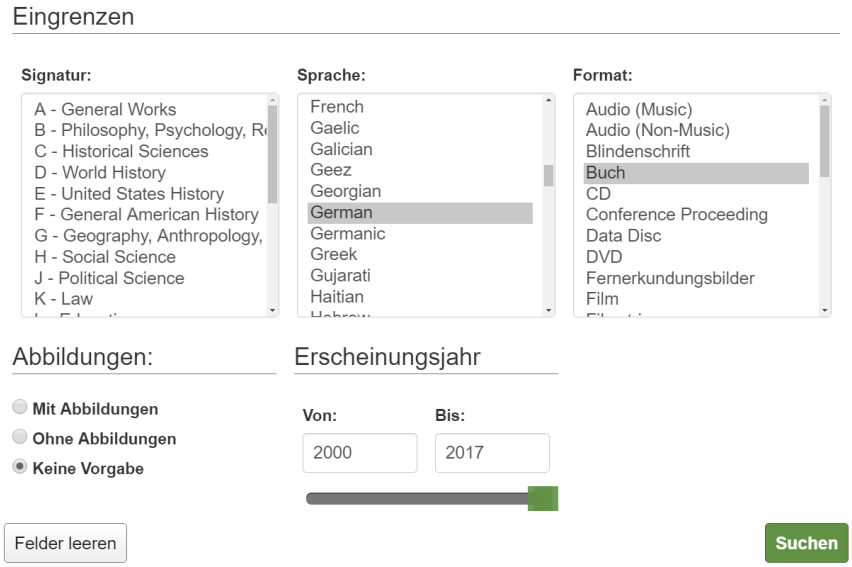


Abbildung 2.3: Erweiterte Suche - Eingrenzung.
Screenshot: http://vufind.org/demo/Search/Advanced, 24.06.16

Autorensuche

Die Autorensuche stellt eine spezialisierte Suchmöglichkeit dar. Es werden nur Felder des Solr-Indexes durchsucht, die Informationen über einen Autor enthalten.

Eine benutzerfreundliche Erweiterung stellt die Einbindung von Wikipedia-Artikeln dar. Wenn ein Autor mit vollem Namen gesucht wird, werden weitere Informationen über diesen Autor oberhalb der Ergebnisliste eingeblendet. Jedoch wird zur Bestimmung des Artikels nur der Name verwendet. Dadurch kann es passieren, dass ein Wikipedia-Artikel über den falschen Autor angezeigt wird. In Bibliothekssystemen werden sogenannte Normdatensätze gepflegt, in denen weitere Informationen zu finden sind. Unter anderem eine eindeutige Nummer, die den richtigen Autor bei Wikipedia identifizieren könnte. Jedoch verarbeitet VuFind keine Normdatensätze und kann deshalb nicht auf die zusätzlichen Informationen zugreifen.

Facetten

Die facettierte Suche beruht auf einer Funktionalität von Solr (siehe Kapitel 2.3.3). Eine Facette beschreibt dabei eine Liste von Eigenschaft, die in den Dokumenten vorkommen. Da zumeist mehrere Dokumente gleiche Eigenschaften haben, kann man diese Eigenschaften zum Filtern der Suchergebnisse nutzen. Eine solche Facette entspricht dem Inhalt eines Feldes des Solr-Indexes.

Dem Nutzer wird für jede bereitgestellte Facette eine Liste von Eigenschaften angezeigt, die nach Häufigkeit des Vorkommens in den Suchergebnissen sortiert ist. Durch die Auswahl einer Eigenschaft in einer Facette wird die Suche verfeinert. Es können verschiedene Facetten gewählt werden, wobei die Reihenfolge keinen Unterschied macht. Teilweise können auch mehrere Eigenschaften einer Facette ausgewählt werden.

Für die Facetten steht eine eigene Konfigurationsdatei zur Verfügung. In dieser kann eingestellt werden, welche Felder als Facette angeboten werden sollen und wo diese Facetten angezeigt werden. So ist es möglich, oberhalb der Ergebnisliste eine spezielle

Facette anzeigen zu lassen, während die Facetten sonst in der Seitennavigation zu finden sind. Des weiteren können die Facetten für die erweitere Suche konfiguriert werden.

Detailanzeige

Die Anzeige eines Dokuments in der Ergebnisliste einer Suche in VuFind ist sehr kurz gehalten, um eine grobe Übersicht über die Ergebnisse zu geben. VuFind verfügt daher über eine umfangreiche Detailansicht, die alle Informationen über ein Dokument anzeigen kann. Diese umfasst Stammdaten wie Titel, Autor und Erscheinungsjahr, kann aber auch um zusätzliche Informationen wie zum Beispiel der Signatur, dem Standort des Dokuments und der Verfügbarkeit an den jeweiligen Standorten erweitert werden.

Die Detailanzeige bietet weiter die Möglichkeit, das Dokument in verschiedene Formate wie BibTex, MARC-21 oder RefWorks zu exportieren, als E-Mail zu versenden oder eine Quellenangabe für Zitate in verschiedenen Zitierstilen zu generieren.

Durchstöbern

Einen anderen Einstieg in die Suche nach Dokumenten bietet das Durchstöbern. Es findet auf einer Metaebene statt, wobei ausgewählte Facetten bereit gestellt werden, die zuerst durchgeschaut werden müssen. Dabei kann man die einzelnen Eigenschaften einer Facette alphabetisch sortieren oder anhand einer anderen Facette gruppieren.

Die Gruppierung gibt dabei einen interessanten Überblick über die Zusammenhänge zwischen einzelnen Eigenschaften. So kann man Autoren mit einer geografischen Komponente gruppieren, um zum Beispiel herauszufinden, welche Autoren Veröffentlichungen in Deutschland oder den USA haben.

Für einen Nutzer kann die alphabetische Sortierung verwirrend sein. Er muss zuerst einen Anfangsbuchstaben wählen, um dann eine Liste von Eigenschaften zu bekom-

men, die mit dem gewählten Buchstaben beginnen. Jedoch ist die Liste nicht alphabetisch sortiert sondern nach Häufigkeit. Das passiert aus technischen Gründen, da je nach Konfiguration nur bis zu 300 Eigenschaften angezeigt werden können. Einerseits bildet Solr dabei eine Begrenzung, weil eine lange Ergebnisliste viel Arbeitsspeicher belegt, andererseits wäre die Anzeige mangels einer Blätterfunktionalität sonst überfüllt.

Hierarchie

Metadaten, die in bibliographischen Datensätzen erfasst werden, können sehr umfangreich sein. Es werden nicht nur die offensichtlichen Informationen wie Titel und Autor erfasst, sondern teilweise auch die Relation zwischen verschiedenen Werken. So kann es einen Datensatz über einen Artikel geben, der Teil einer Zeitschrift ist, die wiederum durch einen anderen Datensatz repräsentiert wird.

VuFind kann diese Art der Relationen abbilden, aber auch Relationen, die zwischen mehrere Ebenen von Werken oder unterschiedlichen Hierarchien bestehen. Dafür müssen im Solr-Index viele Felder korrekt und vollständig angegeben werden[Kat12]:

hierarchy_top_id Die Identifikationsnummer (ID) des Datensatzes, der an der Spitze der Hierarchien steht. Es können auch mehrere IDs angegeben werden, wenn der aktuelle Datensatz mehreren Hierarchien angehört.

hierarchy_top_title Der Titel des Datensatzes, der an der Spitze der Hierarchie steht. Hier können ebenso mehrere Titel angegeben werden, wenn der aktuelle Datensatz mehreren Hierarchien angehört. Die Reihenfolge muss dabei der von `hierarchy_top_id` entsprechen.

hierarchy_parent_id Die ID des Datensatzes, der in der Hierarchie über dem aktuellen Datensatz steht, beziehungsweise die IDs der jeweiligen übergeordneten Datensätze, wenn der aktuelle Datensatz mehreren Hierarchien angehört.

hierarchy_parent_title Die Title der Datensätze, die in den Hierarchien in derselben Reihenfolge wie `hierarchy_parent_title` über dem aktuellen Datensatz stehen.

hierarchy_sequence Ein Wert, der die Position des aktuellen Werkes relativ zu den Datensätzen auf derselben Hierarchiestufe abbildet. Da dieser Wert alphanumerisch sortiert wird, sollten die Zahlenwerte mit vorgestellter Null angegeben werden.

is_hierarchy_id Die ID des aktuellen Datensatzes. Dieser Wert muss derselbe sein, wie der Wert des `id`-Feldes aus dem Solr-Index.

is_hierarchy_title Der Titel des aktuellen Datensatzes. Dieser Wert muss derselbe sein, wie der Wert des `title`-Feldes aus dem Solr-Index.

Templates

Da VuFind auf den Funktionalitäten von `Zend Framework` basiert, steht ein sehr umfangreiches und mächtiges Template-System zur Verfügung.

Es werden einige Hilfsmittel mitgeliefert, mit denen zum Beispiel innerhalb eines tief verschachtelten Templates ein Skript oder ein Stylesheet in das `<head>`-Tag eingefügt oder als Link angehängt werden kann, sowie Funktionen zum Maskieren von HTML-Kontrollzeichen (Escaping) oder für die Internationalisierung.

Mobile Entgeräte

Das Standard-Theme von VuFind basiert auf der `Bootstrap` Bibliothek. `Bootstrap` ist eine Sammlung von Komponenten und Erweiterungen, die einem Web-Entwickler die Arbeit erleichtern können. Des weiteren wird die Technik des responsiven Webdesigns unterstützt. Beim responsiven Webdesign werden Webseiten so aufgebaut, dass sie auf physikalische Aspekte des Gerätes reagieren können. HTML5 und CSS3 bieten

dafür die technische Grundlage durch die Einführung von Media Queries mit denen es möglich ist, bestimmte Abschnitte eines Stylesheets nur dann zu aktivieren, wenn gewisse Voraussetzungen des Endgerätes gegeben sind. So kann man eine Webseite an jedes Gerät ausliefern, anstatt serverseitig den Gerätetyp zu identifizieren und zwischen verschiedenen Anzeigemodi zu wechseln.

Benutzerverwaltung

VuFind bietet einige personalisierte Funktionalitäten an, die einen Benutzer-Account voraussetzen. Dabei werden die bei der Registrierung angegebenen Nutzerinformationen in einer Datenbank gespeichert.

Da größere Organisationen meist eigene Authentifizierungsserver betreiben und diese die Benutzerinformationen verwalten, bietet VuFind einige Schnittstellen an, die es ermöglichen, die werksseitige Datenverwaltung direkt zu nutzen. Unter anderem werden LDAP und Shibboleth unterstützt.

Die personalisierten Funktionalitäten sind das öffentliche Kommentieren und das Verschlagworten von Dokumenten mit Tags sowie das Speichern von Dokumenten in Favoritenlisten.

Open Archives Initiative

Die Open Archives Initiative (OAI) ist im Jahre 2000 aus der Open Access Bewegung entstanden und strebt die einfache und freie Weitergabe von Metadaten über Dokumente im Internet an. Dafür entwickelte die OAI ein Protokoll, das den Austausch von Metadaten standardisieren soll.

Das `Open Archives Initiative Protocol for Metadata Harvesting` (OAI-PMH) beschreibt dabei wie Metadaten von verschiedenen Servern gesammelt werden und auf welche Art ein Server Metadaten anzubieten hat.[LVdS01]

VuFind bietet die Möglichkeit, mit Hilfe des OAI-PMH Metadaten von mehreren

Servern zu indizieren. Dazu werden separate Skripte und Konfigurationen angeboten, die mit Hilfe eines Cron Jobs automatisiert werden können.

Reserve für Kurse

Bibliographische Systeme können Listen über Bücher bereit stellen, die für bestimmte Kurse reserviert sind. Mit VuFind können diese Listen direkt vom bibliographischen System ausgelesen werden oder mit in den Solr-Index eingepflegt werden, so dass Nutzer mit Hilfe des Kursnamens und dem Namen des Dozenten die Liste der reservierten Dokumente durchsuchen können.

Tools zur Indizierung

Die Erstellung des Solr-Indexes ist eine intellektuell und zeitlich aufwändige Arbeit. Mit SolrMarc, das mit VuFind ausgeliefert wird, steht eine Möglichkeit bereit, diese Aufgabe zu vereinfachen. Hierfür werden Konfigurationsdateien angelegt, die beschreiben, wie bibliographische Daten in den Solr-Index übernommen werden sollen.

CLI Tools

Für die Arbeit mit VuFind werden einige Kommandozeilen-Tools angeboten, um wiederkehrende Aufgaben schnell und vollständig zu erledigen, zum Beispiel das Kompilieren von CSS-Dateien oder das Anlegen und Registrieren von neuen Controllern.

Administrationsbereich

VuFind hat einen kleinen Administrationsbereich, der jedoch recht minimalistisch ist. Dieser wird bei der Installation verwendet, um die Zugangsdaten zu einem Datenbanksystem einzustellen und den Zugang zu einem möglichen Integrierten Bibliothekssystem zu hinterlegen. Dabei werden auch noch einige Konfigurationen vorgenommen

und Schreib- sowie Leserechte geprüft. Der Administrationsbereich wird auch zum Einzuspielen von Updates genutzt.

Jedoch verfügt VuFind nicht über eine Rollenverwaltung, so dass jeder Nutzer, ob registriert oder nicht, auf diesen Bereich zugreifen kann. Der einzige Schutz ist eine spezielle Einstellung in der Hauptkonfigurationsdatei. Daher ist der Administrationsbereich auch nicht dafür gedacht, für sonstige Betriebseinstellungen oder Wartungsarbeiten über die Web-Oberfläche genutzt zu werden.

Tests

In der modernen Softwareentwicklung hat sich gezeigt, dass das Schreiben von automatisierten Tests die Qualität und Zuverlässigkeit von Programmen erhöhen kann. Daher wurde für VuFind eine Reihe von Tests implementiert, die viele Bereiche der Software abdecken. Dabei werden in den jeweiligen Modulen von VuFind einerseits einzelne Klassen und deren Methoden durch Unit-Tests, andererseits zusammenhängende Funktionalitäten mit Integrationstests kontrolliert. Diese Tests werden von der Community gepflegt und können beliebig erweitert werden, um eigene Funktionalitäten mit automatisierten Tests abzudecken. Als Test-Framework wird PHPUnit genutzt, das als OpenSource-Projekt frei verfügbar ist.

2.2 Zend Framework

Das `Zend Framework` ist ein modulares, objektorientiertes Framework für die Entwicklung von Webseiten und -applikationen, das im März 2006 in der Version 1.0 veröffentlicht wurde und im Juni 2016 in der Version 3.0 erschienen ist. VuFind nutzt das `Zend Framework` in der Version 2.4.6. Es besteht aus einer Sammlung von vielen lose gekoppelten Modulen. Der Hersteller, `Zend Technologies`, bietet schon über 60 Module an. Außerdem werden in der Community um das `Zend Framework` laufend neue Module entwickelt.

Im Umfeld des `Zend Frameworks` sind nicht nur neue, von der Community entwickelte Module entstanden, sondern auch weitere Software und Tools für die Benutzung mit dem `Zend Framework`. So wurde 2009 der `Zend Server` veröffentlicht, der eine für das `Zend Framework` optimierte Version von PHP darstellt. Des weiteren entstand `Zend Studio`, eine proprietäre Entwicklungsumgebung für Webapplikationen mit PHP und dem `Zend Framework`. `Zend Technologies` bietet zusätzlich mehrere Zertifizierungsprogramme für Entwickler an. Dabei werden den Teilnehmern die Arbeitsweisen der einzelnen Module und ihre Arbeitsweise näher gebracht.

Die Lernkurve des `Zend Frameworks` ist sehr steil, da es aus mehreren hundert Klassen besteht, die jeweils eine eigene Funktionalität oder eine Spezialisierung einer Funktionalität bereitstellen. Außerdem müssen mehrere Konfigurationsdateien gewartet und bearbeitet werden. Daher wird an dieser Stelle nur eine kurze Einführung in das `Zend Framework` gegeben.

2.2.1 Konfiguration

Ein Hauptbestandteil der Arbeit mit dem `Zend Framework` besteht darin, die Funktionalität der zu entwickelnden Applikation nach den jeweiligen Bedürfnissen zu beeinflussen. Dies geschieht im Wesentlichen durch die Manipulation der Applikationskonfiguration und der Modulkonfigurationen.

Nach dem Eintreffen einer Anfrage des Benutzers muss die Applikation gestartet werden. Dafür ist der Befehl `Zend\Mvc\Application::init(\$config)->run()` zuständig. Er erwartet eine Applikationskonfiguration, die unter anderem die zu ladenden Module, den `Service Manager` und einige Pfade konfiguriert.

Die Konfigurationen der jeweiligen Module werden daraufhin eingelesen. Dadurch stehen nun die Funktionen der einzelnen Module bereit und können verwendet werden.

2.2.2 Service Manager

Der `Service Manager` ist eines der wichtigsten Objekte, das zu Beginn einer Anfrage erstellt wird. Er verwaltet Instanzen von verschiedenen Klassen, die in der Applikationskonfiguration definiert wurden. Diese Klassen können Teile der Model-Schicht abbilden, ein Controller sein oder ein Objekt der Betriebslogik. Dabei weiß der `Service Manager`, wie die jeweiligen Instanzen erstellt werden. Das geschieht meist über `Factory`-Methoden, die in der Applikationskonfiguration angegeben werden, oder durch den Standardkonstruktor einer Klasse.

Abhängigkeiten zu Systemkomponenten wie dem `Service Manager` oder dem `Event Manager` können durch `AwareInterface`s ausgezeichnet werden. Implementiert eine Klasse ein solches Interface, wie zum Beispiel das `EventManagerAwareInterface`, so kann der `Service Manager` diese Abhängigkeit beim Instanziieren der Klasse auflösen.

2.2.3 Event Manager

Das `Observer-Pattern` ist im `Zend Framework` tief verwurzelt und wird durch den `Event Manager` zentral verwaltet. Der `Event Manager` bietet die Möglichkeit, `Listener`-Objekte zu registrieren, die auf benannte Ereignisse reagieren. So wird das `Observer-Pattern`, aber auch ein aspektorientiertes oder eventorientiertes Design umgesetzt.

Das Zend Framework nutzt den Event Manager, um den Ablauf einer Benutzeranfrage zu strukturieren. So gibt es vordefinierte Events, die in der Klasse MvcEvent zu finden sind. Darunter die Events route, das dazu führt, dass die URL-Parameter ausgewertet werden, render für die Generierung der HTML-Seite und finish, das angibt, dass die Verarbeitung der Anfrage beendet wurde. Diese Events können durch eigene Funktionalität erweitert werden, aber es können auch beliebige neue Namen für eigene Events eingeführt werden.

2.2.4 Module Manager

Das Laden der Konfiguration der jeweiligen Module, das in Abschnitt 2.2.1 schon genannt wurde, übernimmt der Modul Manager. Dieser iteriert über eine Liste von Modulnamen und initialisiert die jeweiligen Module durch das Hinzufügen der Modulkonfiguration zur Applikationskonfiguration, sowie das Registrieren von Listeners beim Event Manager. Hierfür wird die Datei Module.php des jeweiligen Moduls genutzt.

2.2.5 Benutzeranfrage und Antwort

Wenn alle Konfigurationen und die wichtigsten Objekte geladen und initialisiert sind, kann eine Benutzeranfrage angenommen werden. Eine Anfrage wird dabei von einem Request-Objekt repräsentiert und durch mehrere Stationen verarbeitet.

Zuerst wertet der Router die angefragte URL aus, um den zuständigen Controller und dessen Action zu ermitteln. Der Controller wird mit Hilfe des Service Managers geladen. Die Action, die durch eine Methode des Controllers repräsentiert wird, wird durch den DispatchListener ausgeführt. Diese Methode ist dafür verantwortlich, die Anfrage zu verarbeiten und eine Antwort zu generieren.

Die Antwort auf eine Benutzeranfrage wird durch eine Action eines Controllers generiert. Dabei kann es verschiedene Arten einer Antwort geben. Einerseits kann die

`Action` eine Zeichenkette, die direkt ausgeliefert wird, oder eine Instanz von der Klasse `Response` zurückgeben. Diese Klasse ermöglicht es, die einzelnen Teile einer Antwort objektorientiert zu generieren. So können Status-Code sowie einzelne HTTP-Header-Angaben gesetzt, aber auch der Seiteninhalt manipuliert werden. Zum Schluss werden die so gesammelten Informationen zu einer Antwort zusammengesetzt.

Dies geschieht meist im Zusammenspiel mit dem Template-System, das durch das `Zend Framework` bereit gestellt wird. Dabei wird das eigentliche Layout von der Logik und den Daten getrennt in Template-Dateien gespeichert. Die `Action` generiert mit Hilfe des Tempalte-Systems die Seite.

2.3 Solr

VuFind selbst ist keine Suchmaschine. VuFind bildet nur die Benutzeroberfläche, welche die Resultate einer Suchmaschine für den Nutzer visuell aufbereitet und mit vielen Features erweitert. Lucene ist dabei die genutzte Suchmaschine und bildet das Herzstück jeder Suchanfrage. Lucene ist eine von Doug Cutting entwickelte Suchmaschine, die 1999 über SourceForge.org veröffentlicht wurde und seit 2001 ein Teil der `Apache Software Foundation` ist[Sch11], die sich durch hohe Performance, umfangreiche Features und Flexibilität auszeichnet.

Jedoch nutzt VuFind nicht Lucene direkt. Solr, auch von Apache entwickelt, bildet eine Zwischenschicht. Es basiert auf Lucene und erweitert die Funktionalität noch weiter. Doch anders als Lucene ist Solr nicht nur eine Suchmaschinenbibliothek, die in ein Programm eingebunden werden kann, sondern bildet eine eigenständige Applikation mit einem Webserver. Zwar gibt es Portierungen von Lucenen in andere Programmiersprachen, wie Perl, C#, C++, Python und Ruby[Sch11], doch kann Solr durch alle Programmiersprachen eingebunden werden, die HTTP und XML oder JSON unterstützen. Außerdem hat Solr zwei weiteren Eigenschaften, die für VuFind sehr wichtig sind: Highlighting und Facettierung.

2.3.1 Suchmaschinenindex und Datenbanken im Vergleich

Nun stellt sich die Frage, warum eine Suchmaschine wie Lucene genutzt wird anstatt einer relationalen Datenbank wie MySQL. Von außen betrachtet haben beide Technologien fast dieselbe Funktionalität: Es können Datensätze gespeichert und ausgelesen werden, wobei man beim Auslesen Einträge filtern und sortieren kann. Außerdem gibt es bei Datenbanken und Suchmaschinen eine Indizierung der Daten, um den Zugriff zu beschleunigen. Doch wo liegt dann der Unterschied?

Datenbanken

Datenbanken bestehen aus Tabellen, die meist sehr viele Zeilen an Datensätzen enthalten. Tabellen können durch Referenzen logisch verbunden werden und beschreiben so ein größeres Bild der Wirklichkeit. Betrachtet man nun eine Tabellen für sich alleine ohne Optimierungen, dann würde bei einer Abfrage jeder Datensatz überprüft werden müssen, ob er den Kriterien der Abfrage entspricht. Da die Einträge einer Tabelle im Allgemeinen nicht sortiert sind, kann keine Voraussage getroffen werden, welche Einträge übersprungen werden können. Und da von Datenbanken erwartet wird, dass immer alle zutreffenden Datensätze liefert werden, kann die Suche auch nicht vorzeitig beendet werden. Abgesehen von der ineffizienten Suchstrategie an sich gibt es noch den Nachteil, dass Datenbanken oft zu groß sind, um vollständig im Arbeitsspeicher vorgehalten zu werden. Ein lineares Durchsuchen bedeutet somit, dass alle Daten einer Tabelle von einem Sekundärspeicher wie der Festplatte in den Arbeitsspeicher gelesen werden müssen. Diese I/O-Operationen verlangsamen die Suche deutlich.

User		
id	name	password
42	⋮ Mustermann ⋮	⋮

)

Article			
id	title	⋮	user_id
23	⋮ Hallo Welt ⋮	⋮ ⋮ ⋮	42

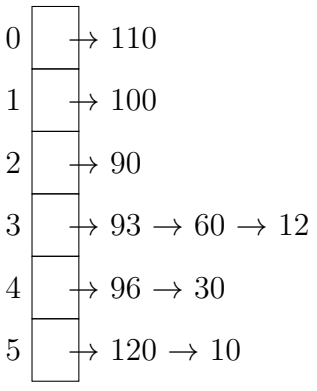
Hier kommen die Indizes der Tabellen ins Spiel. Ein Index ist eine Datenstruktur, welche die jeweiligen Einträge einer Tabellenspalte als Schlüssel vorhält und auf Tabelleneinträge beziehungsweise Datenbereiche verweist. Dabei können Suchoperationen in kürzerer Laufzeit durchgeführt werden als ein linearer Durchlauf. Meist liegt die Komplexität einer Suche in solchen Datenstrukturen in $O(log\ n)$, aber auch $O(1)$ ist unter gewissen Voraussetzungen möglich. Hinzu kommt, dass ein Index weniger Speicher braucht und somit fast vollständig im Arbeitsspeicher vorgehalten werden kann, oder, im Falle von baumartigen Datenstrukturen, nur einzelne Unterbäume auf den Sekundärspeicher ausgelagert werden müssen, wodurch der Zugriff auf den langsamen Sekundärspeicher optimiert wird.

Wenn eine Suche in einem Index fündig wird, gibt der Index den Speicherort der Tabelleneinträge an. Dadurch wird die Datenmenge, die von den Sekundärspeichern geladen werden müssen, weiter reduziert.

Der Nachteil von Indizes ist, dass sie parallel zur eigentlichen Datenbank mit gepflegt werden müssen. Das erfordert zusätzlichen Rechenaufwand und Speicherverbrauch.

Die drei meist genutzten Arten von Indizes für Datenbanksystemen sind Hash-Tabellen, B-Bäume und R-Bäume.

Hash-Tabelle Mit Hilfe der Hash-Funktion wird mittels eines Schlüssels ein Zahlenwert errechnet, der die Position des gesuchten Wertes in der Hash-Tabelle beschreibt. Dabei können Kollisionen entstehen, bei denen zwei verschiedene Schlüssel auf denselben Zahlenwert abgebildet werden. Die Kollisionsbehandlung wird je nach Implementierung unterschiedlich gehandhabt, oft unter Zuhilfenahme von verketteten Listen. Jedoch wird dieser Punkt hier nicht näher behandelt.

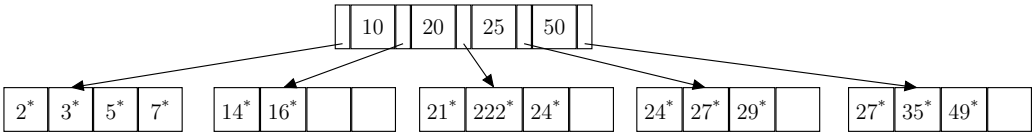


Im Allgemeinen hat der Zugriff auf einen Wert eine
Komplexität in $O(n)$ mit n als Anzahl der Werte in der Hash-Tabelle. Wird eine Hash-Tabelle mit ausreichender Größe initialisiert, so kann der Zugriff in $O(1)$ liegen. Datenbanksysteme können dies durch das regelmäßige Neuerstellen und Neubefüllen einer größeren Hash-Tabelle optimieren. Das Einfügen und Löschen von Elementen liegt auch in $O(1)$.

Daher ist diese Art von Index sehr schnell. Jedoch kann eine Hash-Tabelle nur genutzt werden, wenn der Schlüssel vollständig bekannt ist, zum Beispiel bei der Suche nach einem einzigartigen Identifikationsmerkmal, das als Fremdschlüssel genutzt wird. Eine unscharfe Suche mit einem Wildcard-Operator kann mit einer Hash-Tabelle nicht durchgeführt werden. Auch eine Bereichssuche kann mit einer Hash-Tabelle nicht abgebildet werden.

Je nach Anforderung führt eine Hash-Tabelle zu einem weiteren Nachteil: Durch die Abbildung auf Hash-Werte geht die Sortierung verloren. Dadurch müssen Ergebnislisten nachträglich sortiert werden.

B-Baum Dieser Indextyp besteht aus einer baumartigen Datenstruktur mit Knoten, die mehrere Schlüssel enthalten und mehrere Kinder haben können. Dabei hat jeder Knoten immer ein Kind mehr, als er Schlüssel enthält. Die Schlüssel innerhalb eines Knotens sind sortiert und geben den Bereich der Werte der Kinderknoten an. Bei der Suche nach einem Eintrag in einem B-Baum müssen nur die Schlüssel in einem Knoten verglichen werden, um dann zu entscheiden, entlang welches Kindes man den Baum hinabsteigt. Daraus folgt, dass die Höhe des Baumes der dominierende Faktor für die Komplexität der Suche ist. Daher liegt der Suchoperator in $O(log\ n)$, da Bäume logarithmisch zu der Anzahl von Elementen wachsen. Das Einfügen und Löschen von Elementen liegt auch in $O(log\ n)$.



Damit ist ein B-Baum zwar von der Laufzeit her langsamer als eine Hash-Tabelle, jedoch deutlich schneller als das lineare Durchsuchen der Datenbank. Außerdem sind die Schlüssel sortiert. Somit können auch Abfrageoperatoren, die eine Ordnung voraussetzen, wie $>$ oder $<$, genutzt werden, und eine sortierte Datenausgabe ist automatisch gewährleistet. Zudem können Wildcard-Operatoren genutzt werden, solange dieser Operator nicht als erstes Zeichen im Suchwort genutzt wird.

R-Baum Ein R-Baum ist ähnlich strukturiert wie ein B-Baum. Diese Art von Index wird für räumliche Daten genutzt. Dabei werden die Koordinaten eines minimal umgebenden Rechtecks, das alle Kindknoten und Objekte umfasst, als Schlüssel genutzt. Somit sind diese Indizes darauf spezialisiert, räumliche Anfragen schnell zu beantworten. Dabei kann nicht nur das Enthaltensein eines Objektes in einem anderen Objekt abgefragt werden, sondern auch Überschneidungen, Berührungen oder die Nachbarschaft von Objekten.

Suchmaschinenindex

Für Suchmaschinen werden meist invertierte Indizes verwendet. Diese Art von Index enthält eine Wortliste über alle Wörter, die in den indizierten Dokumenten vorkommen, und für jedes Wort Verweise auf die jeweiligen Vorkommen in den Dokumenten. Oft werden auch weitere Informationen pro Wort vorgehalten wie die Häufigkeit oder der Kontext, in dem das Wort vorkommt.

Die Wortliste wird beim Indizieren von Dokumenten durch eine Vorverarbeitungsfunktionalität zusammengestellt. Dabei werden die einzelnen Wörter durch linguistische Methoden manipuliert und vereinfacht. Bei den linguistischen Methoden handelt es um:

Erkennen von Wortgrenzen Zum Extrahieren einzelner Wörter aus einem Dokument.

Stammformreduktionen Zum Reduzieren der Wörter auf ihren Wortstamm.

Erweiterungen wie zum Beispiel Synonyme.

Außerdem werden Majuskeln durch Minuskeln ersetzt, sowie diakritische Zeichen und Sonderzeichen entfernt. Zuletzt wird die Wortliste alphabetisch sortiert und mit den dazugehörigen Daten persistent gespeichert.

Werden nun neue oder geänderte Daten indiziert, so durchlaufen diese dieselbe Vorverarbeitung und werden wiederum persistent gespeichert, jedoch separat zu dem älteren Index. Beim Durchsuchen werden beide Indizes genutzt, jedoch übersteuern die Einträge des neuen Indexes die des älteren, um nur die aktuellen Informationen zu präsentieren. Daher müssen Suchmaschinenindizes mit der Zeit optimiert werden. Dabei werden die verschiedensten Versionen der Indizes zusammengeführt, alte Daten werden verworfen, neue Daten übernommen und das Ganze als ein Index wieder persistiert. Dies wird inkrementelles Indexieren genannt.

Wird nun eine Suchanfrage an einen solchen invertierten Index gestellt, so wird zunächst die Anfrage durch dieselben Methoden normalisiert. Nur Funktionalitäten zum Anreichern und Erweitern des Wortschatzes, wie im Falle der Synonyme, sind nicht notwendig, da diese Funktionen unnötig Rechenzeit verbrauchen und das Suchergebnis verfälschen können. Die Wortliste des invertierten Indexes wird dann mittels Suchalgorithmen, wie die Binärsuche oder Intervallsuche, nach den Wörtern der Suchanfrage durchsucht.

Vergleich

Weiterhin scheinen die beiden Technologien sehr ähnlich. Doch der Unterschied liegt im Detail. Datenbanken sind durch die Unterteilung mit Tabellen in der Lage, die verschiedensten Arten von Daten und ihre Struktur abzubilden, zu speichern und zu manipulieren. Außerdem wird versucht, Redundanz zu vermeiden, und es wird eine exakte Suche vorausgesetzt, um Datensätze effizient zu finden. Außerdem liegt er Optimierungsschwerpunkt auf den Durchsatz beim Abrufen von Daten aus vielen Datenquellen.

Suchmaschinen sind für freie, unstrukturierte Texte konzipiert, die sich nicht ändern, und Daten werden zusätzlich angereichert, um eine freiere Suche anbieten zu können. Es gibt nur eine Datenquelle und die Inhalte können nicht durch logisch verknüpft werden, wie es bei einer Datenbank möglich ist.

Daraus ergeben sich unterschiedliche Anwendungszwecke: Datenbanken bieten eine performante Möglichkeit zur Verwaltung strukturierter Daten und exakter Suchanfragen, Suchmaschinen sind für die freie Suche konzipiert.

2.3.2 Aufbau von Solr

Solr ist ein modulares System, das aus drei Hauptkomponenten besteht: der Anfrage-Handler (Request Handler), die Suchkomponente (Search Component) und die Antwortkomponente ResponseWriter (vgl. [Tar13b] und [Tar13d]). Diese drei Hauptkomponenten operieren über die Datein eines Indexes, der auch Solr-Core gennant wird, und können beliebig erweitert werden, da Solr sie als Plugins einbindet. Für jeden Solr-Index können in der jeweilig zugehörigen `solrconfig.xml`-Konfigurationsdatei die jeweiligen Komponenten eingebunden, vorkonfiguriert und erweitert werden.

Zusätzlich gibt es für die Struktur des Indexes eine weitere Konfigurationsdatei, in der die einzelnen Felder, sowie Vorverarbeitungsschritte beim Indizieren und Abfragen definiert werden können.

Request Handler

Die Kommunikation mit Solr erfolgt über HTTP-Anfragen. Die Request Handler sind dafür da, solch eine Anfrage entgegenzunehmen und zu verarbeiten. Dabei hat jeder Request Handler eine eigene URL und Aufgabe. Solr liefert schon einige solcher Request Handler mit.

Zum Beispiel gibt es den `SearchHandler` und den `DisMaxRequestHandler`, die Suchanfragen entgegennehmen, oder den `XmlUpdateRequestHandler` und den `JsonUpdateRequestHandler`, die das Importieren und Indexieren von Daten übernehmen.

Die jeweiligen Request Handler werden in der solrconfig.xml eingestellt. Abbildung 2.4 zeigt, wie man einen Search Handler angibt. Dabei beschreibt die Eigenschaft `name` die Pfaderweiterung der URL des Solr-Servers. Innerhalb des `<requestHandler>`-Tags können noch weitere Angaben gemacht werden. In diesem Beispiel wird noch angegeben, dass maximal 200 Dokumente geliefert werden sollen, es sei denn, der Wert wird bei der Anfrage überschrieben.

```
<requestHandler name="/search" class="solr.SearchHandler">
   <lst name="defaults">
        <int name="rows">200</int>
   </lst>
</requestHandler>
```

Abbildung 2.4: Search Handler in solronfig.xml.

Das `class`-Attribut gibt die zu verwendende Implementierung eines Request Handlers an. So kann man eigene Implementierungen einbinden und das Verhalten von Solr anpassen.

Suchkomponente

Die eigentliche Arbeit übernimmt die Suchkomponente. Ein Request Handler ruft diese mit den in der Anfrage angegebenen Parametern auf.

Jede Suchkomponente wird durch einen Namen identifiziert und muss, ähnlich wie ein Request Handler, in der solrconfig.xml definiert werden.

Einem Request Handler werden ohne weitere Angaben folgende Standardsuchkomponenten automatisch zugeordnet [Tar13c]:

Name	Klasse	Siehe
debug	solr.DebugComponent	
expand	solr.ExpandComponent	
facet	solr.FacetComponent	
highlight	solr.HighlightComponent	
mlt	solr.MoreLikeThisComponent	
query	solr.QueryComponent	
stats	solr.StatsComponent	

Abbildung 2.5: Standardsuchkomponente.

Sollen weitere Suchkomponenten wie zum Beispiel `myComponent` registriert werden, können diese durch ein `arr`-Element mit dem Namen `first-components` oder

`last-components` angegeben werden. Oder sie können durch ein `arr`-Element mit dem Namen `components` selbst definiert werden, wie in der Abbildung 2.6. Mit `components` werden die Standardkomponenten nicht automatisch registriert und müssen mit Hilfe ihrer Namen händisch registriert werden.

```
<requestHandler name="/search" class="solr.SearchHandler">
   <lst name="defaults"> <int name="rows">200</int> </lst>
   <arr name="components">
      <str>query</str>
      <str>myComponent</str>
   </arr>
</requestHandler>
```

Abbildung 2.6: Suchkomponenten eines Search Handlers.

Antwortkomponente

Die Antwortkomponente (Response Writer) generiert aus dem Resultat der Suchkomponente eine formatierte Ausgabe. Es werden Response Writer für einige weitverbreitete Formate wie CSV, XML und JSON mitgeliefert, aber auch für die direkte Verwendung von Skript- und Programmiersprachen wie PHP, Python und Ruby. So generiert der PHP Response Writer PHP-Code, der direkt evaluiert werden kann [Tar13d], wie Code-Beispiel 2.7 zeigt.

```
$code = file_get_contents(SOLR_URL . '/select?q=VuFind&wt=php');
eval("$result = " . $code . ";");
print_r($result);
```

Abbildung 2.7: Evaluation in PHP

2.3.3 Facetten

Das Facettieren einer Suche kann mit weiteren Parametern in der Suchanfrage durchgeführt werden. So können bei einer normalen Suchanfrage durch das Setzen des Parameters `facet` auf `true` und der möglicherweise wiederholten Angabe von `facet.field` die anzubietenden Facetten zusätzlich abgefragt werden.

```
/select?q=VuFind&facet=true&facet.field=language&facet.field=format
```

```
[...]
"facet_fields" : {
   "language" : [
      "german" , 17,
      "english" , 15,
      "french" , 9,
      "spanish" , 2 ],
   "format" : [
      "book" , 32,
      "article" , 28,
      "review" , 23 ] }
[...]
```

Abbildung 2.8: Abfrage von Facetten (1)

Die in Abbildung 2.8 durchgeführte Anfrage liefert unter anderem den Eintrag `facet_fields`. Dieser Eintrag enthält die in der Anfrage spezifizierten Facetten und deren Eigenschaften. In einer weiteren Anfrage kann das Suchergebnis mit den so ermittelten Eigenschaften eingeschränkt werden. Dabei gibt die Zahl, die nach der jeweiligen Eigenschaft angegeben wird, die Anzahl von Ergebnissen in der Ergebnisliste an, die man durch das Filtern mit einer weiteren Facette angezeigt bekommt.

Durch die Angabe des `fq`-Parameters in der folgenden Anfrage werden die Suchergebnisse dementsprechend gefiltert und die Angaben der Facetten aktualisiert.

```
/select?q=VuFind&facet=true&facet.field=language&facet.field=format
                &fq=language:german
```

```
[...]
"facet_fields" : {
   "language" : [
      "german" , 17,
      "english" , 0,
      "french" , 0,
      "spanish" , 0 ],
   "format" : [
      "book" , 12,
      "article" , 5,
      "review" , 0 ] }
[...]
```

Abbildung 2.9: Abfrage von Facetten (2)

Zur Erstellung der Facetten wird wieder der invertierte Index des Solr-Cores genutzt. Jedoch kann die Anzahl von Dokumenten, die zu einer Eigenschaft in der Facette angegeben wird, nicht vorausgesagt werden. Daher müssen die Dokumente für jede Eigenschaft gezählt werden. Durch effiziente Caching-Verfahren kann jedoch das ganze System ausreichen beschleunigt werden.

2.3.4 Solr Cloud

Trotz der effizienten Algorithmen, die in Solr implementiert sind, ist eine einfache Solr-Instanz nicht in der Lage, für jeden Anwendungsfall genug Leistung zu bringen. Das kann der Fall sein, wenn die Last durch viele Benutzeranfragen die Systemkapazität ausreizt oder wenn die Indizes so groß werden, dass diese nicht mehr effizient durchsuchbar sind. Aber auch wenn eine Ausfallsicherheit gefordert wird, der ein einzelnes System nicht nachkommen kann.

Solr liefert mehrere Systeme mit, mit denen diese Probleme angegangen werden können. Darunter fällt das Sharding, also das Aufteilen eines Solr-Cores, um die Da-

tenmenge klein zu halten, und die Replikation, also das Duplizieren eines Solr-Cores, um die Ausfallsicherheit zu erhöhen. Gemeinsam mit einem Lastverteiler und einem Verwaltungssystem werden diese Komponenten als `Solr Cloud` bezeichnet.

Replikation

Bei der Replikation wird ein bestehender Solr-Core dupliziert, so dass dieser durch mehrere Server angeboten werden kann. Durch das Replizieren kann einerseits die Last durch Benutzeranfragen auf mehrere Server verteilt werden, andererseits führt der Ausfall eines oder mehrerer Server nicht zu Beeinträchtigungen des angeboten Services, da Anfragen an den ausgefallenen Solr-Cores an ein anderes Replikat umgeleitet und dann verarbeitet werden können.

Das Replizieren eines Solr-Cores führt zu einem Inkonsistenz-Problem. Beim Einspielen von neuen Datensätzen müssen die Änderungen an allen Replika gleichermaßen vorgenommen werden. Ist ein Replikat zum Zeitpunkt der Indizierung nicht verfügbar, so werden Benutzeranfragen mit veralteten Informationen beantwortet.

Solr Cloud löst dieses Problem durch die Ernennung einer Hauptinstanz, die diese Verwaltungsaufgaben übernimmt. Das bedeutet, dass neue Daten in die Hauptinstanz eingespielt werden und diese die Änderung an alle anderen Replika weitergibt. Sollte ein Replikat erst nach der Aktualisierung aktiv werden, so gleicht dieses den Datenstand mit der Hauptinstanz ab. Fällt die Hauptinstanz aus, so wird ein anderes Replikat zur Hauptinstanz gewählt.

Sharding

Das Aufteilen eines Solr-Cores in einzelne Shards ermöglicht es, den Solr-Index auf mehrere Server zu verteilen. Diese Partitionierung führt dazu, dass ein Shard nicht alle Dokumente enthält und eine Anfrage somit parallel an alle Shards gestellt werden muss. Die gefunden Dokumente des jeweiligen Shards müssen danach zu einer Antwort zusammen gefasst werden.

Beim Indizieren erfolgt die Aufteilung der Dokumente auf die einzelnen Shards anhand des Ergebnisses eines `Murmur`-Hashes auf die Identifikationsnummer der jeweiligen Dokumente.

Der Verlust eines Shards führt unweigerlich dazu, dass nur noch ein Teil des ursprünglichen Indexes erreichbar ist. Die Ausfallwahrscheinlichkeit kann durch das Replizieren der Shards reduziert werden, da ein Shard erst dann nicht mehr erreichbar ist, wenn alle Replika des Shards ausgefallen sind.

Verwaltung von Shards und Replikas

Um ein verteiltes System konsistent und effizient zu betreiben, müssen mehrere Kontrollstrukturen eingerichtet werden, die dafür zuständig sind, die Last optimal zu verteilen, auf das Aktivieren und Ausfallen von Knoten im Server-Cluster zu reagieren und korrekt mit neu zu indizierenden Daten umzugehen.

Eine Kontrollstruktur stellt ZooKeeper dar, das eine zentralisierte Infrastruktur für die Verwaltung von Konfigurationen bildet. Die einzelnen Server des Clusters registrieren sich bei ZooKeeper und gelangen so an die nötigen Konfigurationen und Informationen, die für den Betrieb nötig sind. So sind die in Kapitel 2.3.4 genannte Hauptinstanz der Replika eines Shards abfragbar.

Des weiteren können Anfragen an jeden Server des Clusters gestellt werden. Dieser verteilt die Anfragen auf alle Shards und stellt mit Hilfe von ZooKeeper sicher, dass nur aktive Knoten angesprochen werden.

Die Aufgabe der Lastverteilung wird dabei an die anfragende Applikation übertragen. SolrJ bildet als Java-Entwicklungsbibliothek eine Schnittstelle zu Solr und stellt sicher, dass die Anfragen an Hand eines Rundlauf-Verfahren (Round-Robin) an die Server verteilt werden.

Einsatzgebiet von Solr Cloud

Apache Solr ist für den Enterprise-Sektor konzipiert. Es ist darauf ausgelegt, hocheffizient zu arbeiten. Die Nutzung von Solr Cloud erweitert Solr durch die Fähigkeit, auf verteilten Systemen zu operieren und dadurch annähernd beliebig zu skalieren. Dem Einsatz von Solr Cloud für große Suchmaschinenindizes steht daher nichts im Weg. Jedoch ist die Frage, ab wann die Verteilung auf mehrere Server ratsam ist, nicht objektiv zu beantworten, da dies von einigen Faktoren abhängig ist.

Faktoren dafür sind einerseits die Hardware-Gegebenheiten, die Einstellungen von Solr, wie das Index-Schema, aber auch Anforderungen, wie die maximale Antwortzeit einer Anfrage.

Eine Faustregel kann da die Größe des Arbeitsspeichers darstellen, da optimalerweise der komplette Index und die Laufzeitumgebung von Solr gemeinsam in den verfügbaren Arbeitsspeicher passen sollte. Ist dies nicht der Fall, könnte das Erweitern des physikalischen Arbeitsspeichers, das verkleinern des Indexes oder das Aufteilen mit Hilfe von Solr Cloud eine mögliche Lösung darstellen.

2.4 SolrMarc

Von Anfang an wurde mit VuFind eine Möglichkeit zum Indizieren ausgeliefert. In den ersten Versionen von VuFind wurden zwei PHP-Skripte angeboten, die etwa sechs Stunden zum Indizieren von einer Millionen Datensätzen benötigt haben[KN12] [1]. Ein Skript (import-solr.php) spaltete eine MarcXML-Datei in einzelne Dokumente auf, um sie dann per XSL in ein für Solr verständliches Format zu transformieren und an Solr zu schicken. Das zweite Skript (import-night.php) übersetzte Marc-Felder direkt zu Solr-Feldern, jedoch mussten die zu importierenden Daten in einer bestimmten Datei gespeichert werden. In beiden Fällen mussten Anpassungen durch Änderungen am Code umgesetzt werden.

[1] VuFind: Solr Power in the Library, Katz, Demian ; Nagy, Andrew, 2013

Wayne Graham, vom William and Mary Collage, entwarf daraufhin ein eigenständiges Tool mit dem Namen SolrMarc, das im Juni 2008 als Open-Source-Projekt veröffentlicht wurde. Es ist in Java geschrieben und konnte so die API von Solr direkt ansprechen. Damit wurde die benötigte Rechenzeit drastisch von sechs Stunden auf etwa eineinhalb Stunden gesenkt. Eine grundlegende Überarbeitung von SolrMarc brachte einen weiteren Performance-Gewinn, so dass nur noch etwa eine Stunde für eine Millionen Datensätze benötigt wurde.

Die PHP-Skripte wurden durch SolrMarc ersetzt, so dass VuFind nun standardmäßig SolrMarc ausliefert.

Da SolrMarc stark um das Eingabeformat MARC 21 und MARC-XML herum entwickelt wurde, ist es wichtig, einen Überblick über die Struktur dieser beiden Formate zu geben.

2.4.1 MARC-21

Zum Austausch von bibliographischen Metadaten ist ein standardisiertes Dateiformat vonnöten. In den letzten 50 Jahren sind verschiedene Formate entstanden. Aus dem europäischen Raum stammen zum Beispiel die Dateiformate PICA (Project of Integrated Catalogue Automation), MAB (Maschinelle Austauschformat für Bibliotheken) und In den USA wurden Formate wie MODS (Metadata Object Description Schema), Z39.50 (ein Netzwerkprotokoll zur Abfrage von bibliographischen Informationssystemen) und MARC (MAchine-Readable Cataloging) entwickelt.

Bei der Entwicklung von VuFind und SolrMarc hat man sich für MARC-21 als Austauschformat entschieden. Dieses Format wurde 1966 in einem Pilotprojekt der Library of Congress [Avr75], auf Grund einer Studie[2], entworfen. In den folgenden Jahrzehnten wurde das Format weiterentwickelt und angepasst. Es entstanden Schwesterprojekte, die das Metadatenformat an Bedürfnisse von Bibliotheksverbünden und nationale In-

[2]Hochspringen ? Gilbert W. King u.a.: Automation and the Library of Congress. A Survey Sponsered by the Council on Library Resources, Inc., Library of Congress, Washington 1963.

teressen anpassten und solche, die ein universelles Format entwickeln wollten. So entstanden DENMARC, USMARC und CMARC in Dänemark, USA und der Volksrepublik China, IDSMARC des Informationsverbunds Deutschschweiz, UNIMARC (Universal Machine Readable Cataloging) und zuletzt MARC-21, als Vereinigung von USMARC, CANMARC und UNIMARC, mit der XML-Portierung MARC-XML.

Marc-21 ist ein Binärformat, das für die maschinelle Verarbeitung entwickelt wurde, mit dem die Metadaten eines Werkes abgelegt werden können. Dieses Format wurde von der Library of Congress standardisiert. Dabei werden Informationen zu einem Datensatz anhand von Feldern abgelegt, die durch sogenannte Tags identifiziert werden können. Ein Tag beschreibt dabei eine bestimmte Eigenschaft des Datensatzes. Felder können durch Unterfelder feiner gegliedert werden.

Das Format enthält Trennzeichen, die durch nicht nicht-alphanumerische Zeichen repräsentiert werden. Daher wird das Format als Binärformat angesehen, obwohl Address- und Längenangaben nicht durch eine Binärzahl angegeben werden, sondern durch Zeichenketten in dezimaler Schreibweise. Durch die Angabe der Länge eines Datensatzes sind weitere Trennzeichen überflüssig und die Datensätze können somit hintereinander in eine Datei geschrieben werden.

Jeder Datensatz besteht dabei aus drei Bereichen: Einem Kopf, dem Datenverzeichnis und den Feldern, welche die eigentlichen Daten enthalten (vgl. [oC06c]).

Kopf

Der Kopf besteht aus den ersten 24 Zeichen eines Datensatzes.

Im Folgenden werden die Indizes angegeben, so dass das erste Zeichen an der nullten Stelle zu finden ist. Bereichsangaben sind als einschließend zu verstehen.

Länge des Datensatzes (Zeichen 0 bis 4) Die als ASCII-codierte Angabe der Länge beschreibt die Anzahl von Bytes, die der vollständige Datensatz benötigt. Da nur fünf Dezimalstellen angegeben werden können, kann ein Datensatz nur 99.999 Bytes groß werden. Eine binäre Angabe, die bei einem Binärformat zu erwarten

wäre, bei der der Datensatz 128 GiB hätte groß werden können, ist im MARC-21 Standard nicht vorgesehen.

Status des Datensatzes (Zeichen 5) Dieses Zeichen beschreibt die Relation des Datensatzes zu der Datei, in dem der Datensatz gefunden wurde. Bei sich wiederholenden Datenabzügen kann so angegeben werden, ob der Datensatz zum Beispiel neu ist oder geändert wurde.

Typ des Datensatzes (Zeichen 6) Gibt Auskunft über die Art des Dokuments und dessen bibliographische Bestandteile.

Implementierungsspezifisch (Zeichen 7 bis 8) Die Standards ANSI Z39.2 und ISO 2709 nutzen diese beiden Angaben für weitere Informationen über den Typ des Datensatzes.

Zeichencodierung (Zeichen 9) Ein `\#` an dieser Stelle gibt an, das der Datensatz mit dem Marc-8-Encoding codiert wurde. Ein `a` zeigt an, dass UCS, beziehungsweise Unicode, genutzt wurde.

Indikator Länge (Zeichen 10) Gibt die Länge der Indikatoren eines Datenfeldes an. Ist im Standard mit zwei Zeichen festgelegt.

Unterfeldcode Länge (Zeichen 11) Gibt an, wie lang ein Unterfeldcode mit dem nötigen Trennzeichen ist. Die Länge des Unterfeldcodes ist im Standard mit zwei Zeichen festgelegt.

Basisadresse für Daten (Zeichen 12 bis 16) Diese Adresse gibt das erste Byte der Datenfelder an, relativ zum ersten Byte des Kopfes dieses Datensatzes.

Implementierungsspezifisch (Zeichen 17 bis 19) Diese Zeichen sind für die Standards ANSI Z39.2 und ISO 2709 reserviert.

Datenverzeichnisdefinitionen (Zeichen 20 bis 23) Diese vier Zeichen definieren die Längen der einzelnen Bestandteile, die im Datenverzeichnis angegeben werden. Das erste Zeichen steht für die Anzahl der Zeichen für die Längenangabe eines Feldes. Das zweite Zeichen gibt an, wie viele Stellen die Positionsangabe hat. Das dritte Zeichen ist für implementationsabhängige Angaben, die bei jedem Eintrag im Datenverzeichnis zu finden sind. Das letzte Zeichen ist für nur ein Füllzeichen und wird nicht verwendet. Dieser Teil des Kopfes ist vom Standard fest auf `4500` definiert.

Datenverzeichnis

Es gibt verschiedene Methoden, einen Text aus einem Datenformat zu extrahieren. Formate wie XML oder JSON nutzen definierte Zeichenfolgen, die vor und nach dem Text erscheinen. Bei Null-terminierten Zeichenketten ist die Position des ersten Zeichens bekannt und nach dem letzten Zeichen ist ein Null-Byte zu finden. Das Marc-21-Format enthält eine Tabelle, in der die Position und die Länge eines Feldes sowie der Tag des Feldes aufgeführt wird.

Ein Eintrag im Datenverzeichnis besteht somit aus drei Teilen: einem Tag, der mit drei Zeichen angegeben wird, die Länge des Feldes, die mit vier Zeichen angegeben wird, und die Position des Feldes, die mit fünf Zeichen angegeben und relativ zur im Kopf definierten Basisadresse angegeben wird. Daher ist ein Eintrag genau zwölf Zeichen lang.

In Abbildung 2.10 sind beispielhaft zwei Einträge des Datenverzeichnisses zu sehen. Das erste Feld hat das Tag 001, eine Länge von 9 Zeichen (davon sind 8 Zeichen Daten und ein Zeichen ist ein Trennzeichen) und beginnt bei der Adresse 0, also direkt bei der Basisadresse.

Der zweite Eintrag hat das Tag 003, eine Länge von insgesamt 3 Zeichen und beginnt bei der Adresse 9, also direkt nach dem Tag 001.

Das Datenverzeichnis wird mit einem Feldterminator von den Feldern getrennt. Der

001000900000 003000300009#

Abbildung 2.10: Datenverzeichniseintrag im MARC 21 Format

Feldterminator hat den Hexadezimalwert `0x1E` (in Abbildung 2.10 mit # repräsentiert). Die Basisadresse zeigt auf das Zeichen nach dem Feldterminator und kann zur Integrationsprüfung genutzt werden.

Kontroll- und Datenfelder

Es gibt zwei Arten von Feldern: Kontrollfelder und Datenfelder. Beide Arten von Feldern werden für die Speicherung von Informationen über den Datensatz genutzt. Der Unterschied liegt in der Verwendung: Kontrollfelder geben Auskunft über technische Eigenschaften des Datensatzes:

001 - Kontrollnummer Eine eindeutige Zeichenfolge, die zum Identifizieren des Datensatzes innerhalb einer Bibliothek oder eines Bibliotheksverbunds genutzt wird. Dabei muss jeder Datensatz eine eigene eindeutige Zeichenfolge bekommen, auch wenn zwei Datensätze dasselbe Buch beschreiben, jedoch zum Beispiel einmal die Druckausgabe und einmal die Online-Ausgabe.

003 - Kontrollnummertyp Dieses Feld beschreibt das Zahlensystem, mit dem diese Identifikationsnummer generiert wurde. So verwendet das Bibliotheks-Zentrum Baden-Württemberg (BSZ) und andere Verbünde die Pica-Produktions-Nummer, stellen aber aktuell zur Gemeinsame-Normdatei-Nummer (GND-Nummer) um. Im amerikanischen Raum wird unter anderem DLC verwendet.

005 - Änderungsdatum und -uhrzeit In diesem Feld wird der Zeitpunkt der letzten Änderung festgehalten und kann so zur Versionsidentifikation genutzt werden. Die Angabe besteht aus vier Zeichen für das Jahr, zwei Zeichen für den Monat, zwei Zeichen für den Tag, zwei Zeichen für die Stunde, zwei Zeichen für

die Minute, ein Dezimalpunkt und ein Zeichen für die Sekunden, die als Bruchteil einer Minute angegeben werden. Die Stunden werden im 24-Stundenformat angegeben.

So steht 201609042046.9 für den 04. September 2016, um 20:47 Uhr und 54 Sekunden.

006 - Typ Dieses Feld beschreibt, für welche Art von Gegenstand dieser Datensatz steht. Die Informationen dazu sind durch einen Buchstabencode hinterlegt. So ist der erste Buchstabe nur eine grobe Kategorisierung, die durch weitere Angaben in diesem Feld weiter spezifiziert wird. Das Format dieser Angaben ist dabei abhängig von der ersten groben Einordnung.

007 - Physikalische Beschreibung Die verschiedenen Möglichkeiten, dass etwas in physikalische Erscheinung tritt, sind sehr vielfältig. In diesem Feld wird die physikalische Beschreibung durch Buchstabencodes ausgedrückt. Dieser Buchstabencode ist dabei abhängig von den Angaben, die im Feld 006 angegeben wurden. Diese sehr umfangreiche Tabelle kann in der Beschreibung zu Marc-21 auf der Webseite der Library of Congress [oC06a] nachgelesen werden.

008 - Allgemeine Informationen In diesem Feld werden als Buchstabencodes weitere Informationen zu dem Dokument gegeben, die wiederum auf der Webseite der Library of Congress [oC06b] nachgelesen werden können.

Die Inhalte dieser Felder werden direkt nach dem Datenverzeichnis und dem Feldterminator in die Datei geschrieben. Dabei wird jedes Feld mit einem Feldterminator beendet.

Datenfelder

Im Gegensatz zu den Kontrollfeldern sind Datenfelder komplexer im Aufbau. Datenfelder besitzen nicht nur einen Tag, sondern auch Indikatoren, die zur genaueren

Inhaltsbestimmung genutzt werden. Die Anzahl der Indikatoren werden im Marc-21-Kopf angegeben und sind im Standard auf zwei Zeichen definiert.

In Abbildung 2.11 ist ein Datenfeld mit zwei Unterfeldern zu sehen. Das Datenfeld beginnt mit den beiden Indikatoren, gefolgt von den Unterfeldern. Ein Unterfeld beginnt mit dem Unterfeldcode, der aus dem Trennzeichen mit dem Hexadezimalwert `0x1F` (hier als $ dargestellt) und dem Namen des Unterfeldes besteht. Da der Standard eine Unterfeldcodelänge von zwei Zeichen vorgibt, werden Unterfelder mit einzelnen Buchstaben oder Zahlen benannt. Nach dem Unterfeldcode folgen die Daten des Unterfeldes. Ein Feld wird wieder abgeschlossen mit einem Feldterminator, der den Hexadezimalwert `0x1E` hat. Das Feld in Abbildung 2.11 hat eine Länge von 26 Zeichen.

20$aUniversität$vTübingen#

Abbildung 2.11: Datenfeld im MARC 21 Format

2.4.2 MARC-XML

Für die Informationen, die mit MARC-21 abgelegt werden können, gibt es auch ein alternatives Format, das auf XML basiert. MARC-XML übernimmt dabei die Struktur von MARC-21, so dass die Namensgebung und Funktionalität erhalten bleibt, jedoch der Umgang mit den Daten durch bestehende Werkzeuge wie Validatoren und Transformatoren erleichtert wird.

In Abbildung 2.12 ist ein kurzer Datensatz in MARC-XML zu sehen. Erwähnenswert ist hier die Längenangabe des Datensatzes im `leader`-Tag. MARC-XML hat, im Gegensatz zu MARC 21, keine Längenbeschränkung, da XML als solches keine Längenbeschränkung kennt. Das von der Library of Congress herausgegebene XSD zum Validieren eines MARC-XML Dokuments gibt zwar vor, dass diese Stelle mit Zahlen gefüllt sein muss, jedoch hat die dort angegebene Länge keinen Bezug zu den XML-Daten und wird vor allem deshalb verlangt, damit bestehende Programmteile, die den

Kopf auswerten, wiederverwendet werden können.

```
<collection xmlns="http://www.loc.gov/MARC21/slim">
   <record>
      <leader>00142cam 2200301 a 4500</leader>
      <controlfield tag="001">220990</controlfield>
      <datafield tag="999" ind1="2" ind2="0">
         <subfield code="a">Universität</subfield>
         <subfield code="v">Tübingen</subfield>
      </datafield>
   </record>
</collection>
```

Abbildung 2.12: Beispiel: MARC XML

Da XML die einzelnen Felder durch XML-Tags auszeichnet, wird das Datenverzeichnis von MARC-21 nicht benötigt und komplett verworfen.

2.4.3 Arbeitsweise von SolrMarc mit VuFind

Wie man in Abbildung 2.13 sieht, übergibt SolrMarc Daten an Solr und VuFind wiederum liest Daten aus Solr. Dadurch sind SolrMarc und VuFind sehr lose miteinander gekoppelt, da beide nur mit Solr als einer Art Schnittstelle kommunizieren. Dass SolrMarc zum Importieren von Marc-Daten genutzt wird, liegt eher daran, dass Teile der VuFind-Community das SolrMarc-Projekt ins Leben gerufen haben und parallel zu VuFind weiter entwickelten. Es könnte aber ohne weiteres ein anderes Programm genutzt werden, um den Solr-Index zu erstellen.

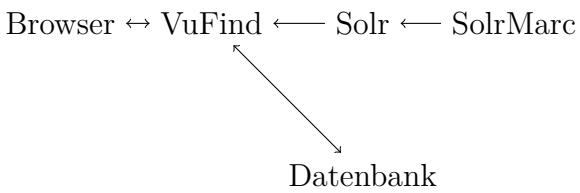


Abbildung 2.13: Datenfluss von VuFind

VuFind enthält ein einfaches Skript, das den Import von Marc-Daten in den Solr-Index stark vereinfacht. Durch das Aufrufen von `import-marc.sh` mit einer Marc-Datei als Parameter wird die Indizierung gestartet. Die Dauer der Indizierung ist stark abhängig von der verwendeten Hardware, der Größe der Marc-Datei, Anzahl von Feldern des Solr-Indexes sowie dem Rechenaufwand, der nötig ist, um die Daten aus den Marc-Daten zu extrahieren. Erfahrungswerte über die Indizierungsgeschwindigkeit liegen beim IxTheo-Projekt zwischen 60 und 100 Datensätzen pro Sekunde. Da SolrMarc auch Volltextindizierung unterstützt, kann die Indizierungsgeschwindigkeit auf unter 10 Datensätze pro Sekunde fallen, wie Erfahrungswerte aus dem KrimDok-Projekt zeigen.

2.4.4 Konfiguration

SolrMarc kennt zwei Arten von Konfigurationsdateien. Die eine Konfigurationsdatei wird meist `import.properties` genannt und enthält allgemeine Einstellungen von SolrMarc. Die andere heißt oft `marc.properties` oder `marc_local.properties` und wird zur Konfiguration der Indizierung genutzt. Beide Arten von Konfigurationen sind sehr umfangreich und teilweise auch komplex. Da es selbst im englischsprachigen Raum kaum komplette Dokumentationen über die verschiedenen Möglichkeiten dieser Konfigurationen existieren, werden hier die einzelnen Einstellungsmöglichkeiten beschrieben.

Allgemeine Konfiguration

Die allgemeine Konfigurationsdatei besteht nur aus Angaben, die das Verhalten von SolrMarc unabhängig von den Indizierungsregeln bestimmen. Es wird pro Zeile ein Schlüsselwort angegeben und der einzustellende Wert, getrennt durch ein Gleichheitszeichen. Kommentare können durch eine Raute eingeleitet werden.

Es folgt die Liste von möglichen Schlüsselwörtern und ihre Bedeutung.

solrmarc.solr.war.path Gibt an, wo Solr und dessen Bibliotheken gefunden werden können.

solrmarc.path Der Pfad zu SolrMarc. In diesem Ordner wird nach weiteren Konfigurationen, Translation Maps und BeanShell-Skripten gesucht. Relative Pfade werden relativ zur ausgeführten Jar-Datei von SolrMarc interpretiert.

solrmarc.site.path Wird dieser Wert angegeben, so werden in diesem Order die weiteren Konfigurationen gesucht.

solrmarc.log.level Dieses Feld gibt das Log-Level von SolrMarc an. Es kann genutzt werden, um SolrMarc zu debuggen, führt jedoch bei zu feiner Einstellung zu Performance-Einbußen.

solrmarc.use_solr_server_proxy Da die Möglichkeit, mit Solr direkt zu kommunizieren, noch nicht vollständig implementiert und nur auf eine ältere Version von Solr angepasst ist, hat diese Einstellung keine Auswirkungen.

solrmarc.use_streaming_proxy Gibt an, ob ein StreamingUpdateSolrServer [sol08b] anstatt des CommonsHttpSolrServer [sol08a] genutzt werden soll.

solrmarc.use_binary_request_handler Da die Möglichkeit, mit Solr direkt zu kommunizieren, noch nicht vollständig implementiert und nur auf eine ältere Version von Solr angepasst ist, hat diese Einstellung keine Auswirkungen.

solr.path Gibt an, wo Solr gefunden werden kann. Ein lokaler Pfad zum übergeordneten Verzeichnis des Solr-Indexes bedeutet, dass SolrMarc den Solr-Index direkt ansprechen soll. Dafür darf der Solr-Index nicht durch ein laufendes Solr genutzt werden. Das Schlüsselwort `REMOTE` bedeutet, dass SolrMarc über das Web-Interface Solr angesprochen werden soll.

solr.core.name Der Name des Solr-Indexes.

solr.data.dir Der Pfad zum Datenverzeichnis des Solr-Indexes. Meist der `data`-Ordner innerhalb des Solr-Core-Ordners.

solr.hosturl Die URL zum Solr-Web-Interface.

solr.updateurl Die URL zur Update-Funktionalität von Solr. Wenn dieser Wert nicht gesetzt wird, wird die `solr.hosturl` durch ein `/update` erweitert.

solr.log.level Gibt an, welches Log-Level für die Ausgaben von Solr genutzt werden soll. Dadurch können Log-Ausgaben von Solr in die Ausgaben von SolrMarc übernommen werden. Angegeben werden dürfen die Log-Level von `java.util.logging.Level`.

solr.indexer SolrMarc kann durch weitere Funktionalitäten erweitert werden. Dafür muss die `SolrIndexer`-Klasse überschrieben werden. Die neue Klasse, oder, wenn es keine Erweiterungen gibt, die `SolrIndexer`-Klasse, muss dann hier mit der Package-Angabe eingetragen werden.

solr.indexer.properties Eine kommaseparierte Liste von Konfigurationsdateien, die beschreiben, wie ein MARC21-Dokument zu einem Solr-Document transformiert werden soll. Dabei werden die Einstellungen von Konfigurationsdateien am Anfang der Liste von später genannten Konfigurationsdateien überlagert.

solr.commit_at_end Wenn dieser Wert auf `false` gesetzt wird, so wird nach dem Indizieren kein `commit`-Befehl an Solr geschickt. Dadurch können mehrere MARC21-Dateien nacheinander indiziert und am Schluss mit einem einzigen `commit`-Befehl in den Solr-Index geschrieben werden.

marc.ids_to_delete Eine kommaseparierte Liste von Dateien (Löschlisten), in denen pro Zeile ein Identifikationswert eines Solr-Dokumentes steht, das gelöscht werden soll.

marc.delete_record_id_mapper Ein Mapping, das angibt, wie der Identifikationswert aus den Löschlisten transformiert werden soll. Zum Beispiel würde $(.*) =>$

u\$1 jedem Identifikationswert ein `u` voranstellen.

marc.just_index_dont_add Wird dieser Wert gesetzt, so durchläuft zwar SolrMarc die Transformation der MARC21-Dokumente zu Solr-Dokumente, jedoch werden diese nicht an Solr geschickt. Damit können die Konfigurationsdateien, die BeanShell-Skripte, die Translation Maps und die nativen Funktionen getestet werden.

marc.combine_records Marc-Datensätze haben eine Längenbeschränkung von 99.999 Zeichen. Eine Möglichkeit diese Begrenzung zu umgehen, ist es, zu große Datensätze in kleinere zu zerlegen. SolrMarc bietet die Möglichkeit, diese Teildatensätze vor dem Indizieren wieder zu einem großen Datensatz zusammenzufügen. Dafür müssen die Teildatensätze in einer Sequenz hintereinander in der Marc-Datei stehen.

Dafür wird in dieser Einstellung ein Regex angegeben, der beschreibt, welche Felder der Teildatensätze zu dem ersten Teildatensatz in der Sequenz hinzugefügt werden sollen. In den meisten Fällen wird das `(.*)` sein.

marc.combine_records.left_field Mit dieser Eigenschaft kann durch die Angabe eines Tags bestimmt werden, welches Feld zur Identifizierung von Teildatensätze benutzt werden soll. Dabei identifiziert dieser Tag das Feld des ersten Teildatensatzes.

marc.combine_records.right_field Mit dieser Eigenschaft kann durch die Angabe eines Tags bestimmt werden, welches Feld zur Identifizierung von Teildatensätze benutzt werden soll. Dabei identifiziert dieser Tag das Feld der folgenden Teildatensätze.

marc.permissive

marc.default_encoding Angabe über die Kodierung der MARC-21-Datei. Die Angabe von `BESTGUESS` bringt SolrMarc dazu, die Kodierung zu erraten.

marc.verbose Wird diese Eigenschaft auf `true` gesetzt, so werden mehr Informationen ausgegeben, unter anderem die Marc-Datensätze.

marc.include_errors Wenn dieser Wert gesetzt wird, wird ein neues Feld in die Solr-Dokumente eingefügt, das mögliche Fehlermeldungen, die beim Indizieren gemeldet wurden, enthält.

marc.to_utf_8 Durch Setzen auf `true` versucht SolrMarc, die Daten als UTF-8 zu lesen und zu schreiben.

marc.unicode_normalize Unicode ist eine Zeichenformatierung, die für bestimmte Buchstaben mehr als eine Darstellung kennt. Um diese Zeichen gleich zu behandelt, kann hier eine Normalisierungsmethode ausgewählt werden, die durch den Unicode-Standard definiert ist [Dav03].

marc.source Gibt an, welche Quelle genutzt werden soll. Valide Angaben sind dabei `FILE`, `Z3950` und `DIR`, wobei `FILE` der Standardwert ist.

marc.override Durch die Angabe einer von `org.marc4j.marc.MarcFactory` abgeleiteten Klasse, kann die Erstellung von neuen Datensätzen, Feldern und Unterfeldern manipuliert und gesteuert werden.

marc.include_if_present Durch Angabe eines Tags werden nur Datensätze indiziert, die ein Feld mit diesem Tag enthalten. Wird nach dem Tag, getrennt durch ein `/`, noch ein Regex angegeben, so wird zusätzlich auch geprüft, ob dieses Feld oder eines seiner Unterfelder auf den Regex passen.

marc.include_if_missing Durch Angabe eines Tags werden nur Datensätze indiziert, die kein Feld mit diesem Tag enthalten. Wird nach dem Tag, getrennt durch ein

/, noch ein Regex angegeben, so wird zusätzlich auch geprüft, ob dieses Feld oder eines seiner Unterfelder auf den Regex passen.

marc.delete_subfields = nomap Wird diese Konfiguration mit einer Liste von durch Doppelpunkt getrennten Tags mit Unterfeld-Codes angegeben, so werden die Felder und Unterfelder aus dem Datensatz gelöscht und nicht indiziert. Dabei ist die Angabe der Unterfeld-Codes optional. Ohne Unterfeld-Code wird das ganze Datenfeld gelöscht.

config.file.dir Der Pfad zu den Konfigurationsdateien.

marc.reader.remap = nomap Hier kann man eine Mapping-Datei angeben, die beim Einlesen der Datensätze die Werte eines Dokuments entsprechend der Mapping-Datei filtert oder transformiert. Diese Mapping-Datei hat ein spezielles Format, da es Befehle enthalten kann.

Die Mapping-Datei enthält pro Zeile eine Aktion, die ausgeführt werden soll. Eine Aktion besteht aus der Erweiterung des Feld-Tags, auf den die Aktion ausgeführt werden soll, eine Bedingung und den eigentlichen Befehl zur Transformation. In der Abbildung 2.14 ist ein solches Mapping zu sehen. Hier wird dem Unterfeld `a` des Datenfeldes `993` die Zeichenkette `Hallo Welt` angehängt, wenn das Unterfeld `a` existiert. Das Datenfeld `993` wird gelöscht, wenn dieses ein Unterfeld `a` mit dem Text `Hallo Welt` enthält.

```
993 =
993_0 = subfieldexists(a), append(a, " Hallo Welt")
993_1 = deleteotherfield(994, a, "Hallo Welt")
```

Abbildung 2.14: Beispiel: Reader Remap

Folgende Bedingungen sind möglich:

true() Immer wahr.

and(expr1, expr2) Wahr, wenn die Bedingung `expr1` und die Bedingung `expr2` gleichzeitig wahr sind.

or(expr1, expr2) Wahr, wenn die Bedingung `expr1` oder die Bedingung `expr2` wahr sind.

not(expr) Negiert den Wahrheitswert von der Bedingung `expr`.

indicatormatches(indicator1, indicator2) Testet, ob die Indikatoren des Feldes den angegebenen Indikatoren entsprechen. Ein `*` kann als Wildcard angegeben werden.

subfieldmatches(subfield-code, regex) Wenn das Feld ein Unterfeld `subfield-code` enthält, dessen Inhalt mit dem Regex übereinstimmt, ist diese Bedingung wahr.

subfieldcontains(subfield-code, needle) Überprüft, ob der Text `needle` in einem Unterfeld `subfield-code` vorkommt.

subfieldexists(subfield-code) Testet, ob ein Unterfeld mit dem Code `subfield-code` existiert.

fieldexists(tag, subfield-code, needle) Überprüft, ob im Datensatz ein Feld mit dem Tag `tag` vorkommt, das ein Unterfeld `subfield-code` besitzt, dessen Text identisch ist mit `needle` oder mit dem Regex `needle` übereinstimmt.

Es gibt folgende Befehle:

replace(subfield-code, old, new) Erstetzt im Unterfeld `subfield-code` alle Vorkommen von `old` mit `new`.

append(subfield-code, content) Hängt an die Daten der Unterfelder `subfield-code` den Text `content` an.

prepend(subfield-code, content) Stellt vor die Daten der Unterfelder `subfield-code` den Text `content`.

deletesubfield(subfield-code) Löscht alle Unterfelder mit dem Unterfeld-Code `subfield-code`.

both(cmd1, cmd1) Führt die Befehle `cmd1` und `cmd2` aus.

deletefield() Löscht das Feld.

deleteotherfield(tag, subfield-code, needle) Löscht ein Feld aus dem Datensatz, dessen Tag `tag` ist, ein Unterfeld `subfield-code` enthält, dessen Inhalt identisch ist mit `needle` oder mit dem Regex `needle` übereinstimmt.

insertfield(field-data) Je nach Formatierung von `field-data` wird ein neues Kontrollfeld, Datenfeld oder Unterfeld eingefügt. Dafür muss `field-data` die Daten der jeweiligen Felder als lesbare Zeichenkette formatiert enthalten.

insertparameterizedfield(field-data, subfield-code, regex) Erstellt, wie `insertfield(field-data)`, ein neues Feld, wobei das neue Feld parametrisiert sein kann, so dass Daten aus dem Unterfeld `subfield-code` des aktuellen Feldes mit Hilfe des Regex `regex` extrahiert und übernommen werden können.

reject() Löscht den aktuellen Datensatz.

2.4.5 BeanShell-Skript

BeanShell ist ein in Java geschriebener Interpreter für eine Skriptsprache, die auf Java basiert. Dadurch kann der BeanShell-Interpreter Java-Sourcecode einlesen und ausführen, verfügt jedoch noch über einige Spracherweiterungen.

Die Skripte werden in derselben Java Virtual Machine ausgeführt wie das aufrufende Java Programm. Dadurch können Werte und Objekte zwischen dem Java Programm und dem BeanShell-Skript ausgetauscht und gegenseitig manipuliert werden.

Es werden einige vordefinierten Funktionen angeboten [Niea]:

source() Liest ein BeanShell-Skript in die Ausführungsumgebung ein und erweitert so die verfügbaren Funktionen.

run() Führt ein BeanShell-Skript in einer eigenen Ausführungsumgebung aus.

eval() Führt eine als String vorliegende BeanShell-Anweisung aus.

Außerdem werden einige Spracherweiterungen angeboten, die die Art und Weise ändert, wie BeanShell arbeitet:

setAccessibility() Überschreibt die Sichtbarkeit von privaten und geschützten Variablen und Methoden, so dass auf alle Bestandteile jeder Klasse zugegriffen werden kann, ohne dass eine Fehlermeldung geworfen wird.

show() Gibt die Ergebnisse aller Gleichungen auf die Standardausgabe aus.

setStrictJava() Erzwingt ein Java-konformes Skript und verbietet eine dynamische Typisierung sowie undefinierte Variablen.

Des weiteren ist man nicht auf die Umgebung einer Klasse angewiesen wie bei Java. BeanShell erlaubt Funktionen auch verschachtelt zu definiert. Verschachtelte Funktionen können dadurch als Objekte angesehen werden.

Eine interessante Spracherweiterung ist, dass man auf den Kontext des Aufrufers zugreifen kann. Dadurch kann eine Funktion eine Variabel auslesen, ändern oder erstellen, die außerhalb des Sichtbereichs der Funktion liegt, jedoch im aufrufenden Sichtbereich. Zur Veranschaulichung hier ein kleines Beispiel:

```
void sayHello() {
   print("Hallo " + this.caller.name);
   this.caller.saidHello = true;
}

name = "Max Mustermann";
sayHello();
print("saidHello: " + saidHello);
```

Die Ausgabe ist:

```
Hallo Max Mustermann
saidHello: true
```

Es ist allerdings fraglich, ob ein solches Feature sinnvoll ist. Es generiert starke Abhängigkeiten zwischen dem Aufrufer und der Funktion/Methode, die nicht durch Sprachkonstrukte wie die Argumentenliste einer Funktion oder Methode dokumentiert werden.

Ein größeres Problem stellt das Projekt BeanShell selber da. Nach der Veröffentlichung von Version 2.0 im Jahre 2005 scheint das Projekt lange Zeit nicht weiter gepflegt worden zu sein. Inzwischen wurde es wieder auf Github.com [Nieb] aufgenommen. Jedoch konnte BeanShell in Sachen Sprach-Features Java nicht einholen. In der aktuellen Version (2.0b6) wird weiterhin nur die Semantik von Java 1.4 unterstützt. Daher ist es nicht möglich, modernen Java-Code zu schreiben.

Außerdem hat BeanShell durch seine Natur als interpretierte Skriptsprache star-

ke Leistungsschwächen gegenüber seinem Vorbild Java. Klar zeigt sich das an einem Beispiel von SolrMarc, das für die Bestimmung des Formates ein BeanShell-Skript [Kat16] einsetzt. Mit dem GetFormatBenchmark wird eine modifizierte Version des BeanShell-Skripts mit dem Java-Pendant verglichen[3]. Dafür wurde jeweils für Java und BeanShell die Zeit von 2,6 Millionen Funktionsaufrufen gemessen. Für statistisch relevante Werte wurde dieser Test jeweils 20 mal wiederholt. Die Tabelle 2.1 gibt die gemessenen Werte in Sekunden an. Daraus folgt, dass in diesem Beispiel BeanShell-Skripte um etwa den Faktor 500 langsamer sind.

	Minimum	Durchschnitt	Maximum
BeanShell	130,2	140	168,3
Java	0,238	0,262	0,465
Faktor	547,1	534,4	361,9

Tabelle 2.1: Benchmark-Ergebnisse zwischen Java und BeanShell in Sekunden

Die veraltete Technik und die fehlende Performance von BeanShell kann man durch die direkte Nutzung von Java umgehen. Möchte man jedoch denselben Entwicklungs-Workflow wie mit den BeanShell-Skripten, ohne händisches Kompilieren behalten, kann man es leicht automatisieren [Obe16].

[3]Die Abhängigkeit zu Marc4J wurde entfernt.

2.5 Struktur und Konzepte von VuFind

In den vorheringen Kapiteln wurden die Grundlagen, auf denen VuFind aufbaut, näher beschrieben. In diesem Kapitel

2.5.1 Erweiterbarkeit

Viele Systeme im Bereich der Webentwicklung stellen nur eine funktionsfähige Basis dar. Zum Beispiel ist Wordpress ein Blogsystem, das für sich alleine zwar schon viele Eigenschaften einer vollständigen Webseite hat, wie zum Beispiel einen Blog, statische Seiten und eine Nutzerverwaltung. Jedoch muss es durch Plugins, Themes und selbstgeschriebene Anpassung erst zu einer allumfassenden Lösung für Webpräsenzen erweitert werden. Django, Drupal, Joomla, Typo3 und VuFind folgen demselben Prinzip. Keines dieser Content Management Systeme ist dafür gedacht, ohne Anpassungen direkt genutzt zu werden. Für kleine Webseiten reicht meist die Anpassung des Themes, für große Webauftritte werden jedoch auch ganz spezielle Features gefordert, die ein Mehrzweck-Web-Framework nicht anbieten kann.

Daher spielt die Erweiterbarkeit bei solchen Systemen eine große Rolle. Die dafür notwendigen Schnittstellen und dazugehörigen Dokumentationen bilden den Kern. Weiterführende Designentscheidungen müssen kritisch bewertet und abgewogen werden.

VuFind, das selbst ein Modul des `Zend Frameworks` darstellt, erbt einen Großteil seiner Schnittstellen direkt vom `Zend Framework`. Dies betrifft die Erweiterbarkeit von Modulen und Templates. Zusätzlich erweitert VuFind sein Angebot an Schnittstellen um eine Funktion zur speziellen Behandlung von Konfigurationsdateien, die eingeführt wurden, um das Suchverhalten bestimmen.

Im Folgenden wird betrachtet, wie Erweiterungen von Konfigurationsdateien, Themes und Module genau funktionieren, und diskutiert, welche Auswirkungen die jeweiligen Designentscheidungen mit sich bringen.

Konfigurationsdateien

VuFind liefert einen Satz von Konfigurationsdateien mit, in dem alle möglichen Einstellungen zu finden sind. Diese Dateien liegen unter `config/vufind/` im VuFind-Ordner. Dort sind unter anderem Konfigurationsdateien für die Grundeinstellungen, wie zum Beispiel für den Titel, das Theme und die Sprache, aber auch für die Suche, Facetten und ILS-Treiber zu finden.

Teile der Konfigurationsdateien, vorwiegend die ILS-Treiber, sind nur Schablonen und enthalten keine Werte. Sie müssen an die jeweilige Umgebung noch angepasst werden. Andere Einstellungsdateien enthalten sehr viele Optionen, von denen nur einige wenige geändert werden müssen, um den Bedürfnissen zu entsprechen.

Hierfür hat VuFind einen Mechanismus implementiert, bei dem Konfigurationsdateien aus verschiedenen Ordnern eine Hierarchie bilden, so dass eine Konfigurationsdatei aus einem Ordner die Einstellung der gleichen Konfigurationsdatei aus einem anderen Ordner erweitert oder überschreibt. Diese Hierarchie wird durch ein Vererbungskonzept realisiert. Ähnlich wie man es aus objektorientierten Programmiersprachen kennt, wird in einer Konfigurationsdatei die übergeordnete Konfigurationsdatei angegeben, von der geerbt werden soll, indem man eine `Parent_Config`-Sektion einführt, in welcher der `path`-Wert, für einen absoluten Pfad, oder der `relative_path`-Wert, für einen relativen Pfad zur übergeordneten Konfigurationsdatei, angegeben wird. Außerdem kann man durch den `override_full_sections`-Wert bestimmen, welche Sektionen aus der übergeordneten Konfigurationsdatei nicht geerbt werden sollen.

Hinzu kommt, dass VuFind ein weiteres Verzeichnis kennt, in dem lokale Änderungen abgelegt werden sollen. Standardmäßig ist das der `local/`-Ordner. Wenn eine Konfigurationsdatei geladen werden soll, kümmert sich VuFind darum, dass versucht wird, diese Datei aus dem `local/config/vufind/`-Ordner zu laden, bevor die Datei im `config/vufind/`-Ordner gesucht wird. Somit kann man mit Hilfe der Hierarchie- und Vererbungswerkzeuge eine Konfigurationsdatei unter `local/config/vufind/` an-

legen, die von der Standardkonfiguration erbt, um dann die Werte zu ändern, die nötig sind.Der Vorteil dieser Methodik liegt in der Flexibilität und der Einfachheit. Ein weiterer Vorteil ist ihre geringere Fehleranfälligkeit bei Updates von VuFind, da neue oder geänderte Standardkonfigurationswerte automatisch übernommen werden und keine weitere Anpassung nötig ist.

Themes

Themes haben eine ähnliche Hierarchie wie die Konfigurationsdateien, jedoch auf Dateiebene. Jedes Theme muss eine Datei mit dem Namen `theme.config.php` haben. Diese Datei enthält ein PHP-Array mit den Einstellungen des Themes. Mit dem Schlüsselwort `extends` kann man den Namen des übergeordneten Themes angeben. Dies führt dazu, dass, wenn eine Template-Datei gerendert werden soll, die Hierarchie nach oben durchlaufen wird, bis die gesuchte Datei in einem Themes in der Hierarchie gefunden wurde.

Auch hier ist ein Vorteil, dass eine implizite Dokumentation der geänderten Templates entsteht und man nur mit einer Untermenge der Templates arbeiten muss, was den Überblick erleichtert.

Anders als bei den Konfigurationsdateien ist diese Methodik jedoch teilweise problematisch in Hinsicht auf Updates. Um ein Template des Themes an einer Stelle anzupassen, muss die entsprechende Datei kopiert werden. Sollte bei einem Update das Template in einem übergeordneten Theme geändert worden sein, so wird diese Anpassung durch die kopierte alte Version übersteuert und tritt nicht in Erscheinung. Das kann bedeuten, dass neue Features nicht angezeigt werden. Aber auch, dass veralteter, teilweise nicht mehr existenter Code aufgerufen wird, was zu Fehlern führt.

Module

Die Nutzung des `Zend Frameworks` als Grundkonstrukt für VuFind ist vor allem an der Aufteilung des Codes in Module erkennbar, da die Ordnerstruktur und teilweise auch die Benennung einzelner Dateien vorgegeben werden.

Ein Modul bildet dabei einen Teilbereich der Applikation ab. So ist VuFind in fünf Module unterteilt:

VuFind Dies ist das Hauptmodul, in dem alle Bestandteile von VuFind zu finden sind, die nicht durch die folgenden Module abgedeckt werden.

VuFindAdmin Dieses Modul enthält die Controller für den Administrationsbereich.

VuFindConsole VuFind bietet einige Kommandozeilen-Tools an. Diese werden in diesem Modul gebündelt.

VuFindSearch Grundlegende Funktionen für die Suche und die Anbidnung an ein ILS (siehe 2.5.2) werden in diesem Modul umgesetzt.

VuFindTheme In diesem Modul sind Erweiterungen des Template-Systems zu finden, dass das `Zend Framework` mit liefert.

Desweitern sind die beiden Module `VuFindDevTools` und `VuFindLocalTemplate`. Ersteres beherbergt eine Hilfskomponente, die für die direkte Entwicklung von VuFind genutzt wird. Zweiteres ist die Grundlage zum Erstellen eines neuen Moduls. So wird dieser Ordner kopiert, wenn bei der Installation angegeben wird, dass ein lokales Modul angelegt werden soll.

Neue Module müssen in der Umweltvariable `VUFIND_LOCAL_MODULES` genannt werden, damit sie geladen werden. Innerhalb eines Modules wird eine Konfigurationsdatei erwartet, die neue Objekte beim `ServiceManager`, aber auch neue Routen registrieren kann.

2.5.2 Anbindung an ein integriertes Bibliothekssystem

Bibliotheken nutzen oft ein integriertes Bibliothekssystem (ILS, aus dem Englischen: integrated library system). Dieses System ist für die Verwaltung interner Prozesse, des Bibliotheksbestandes, sowie Aus- und Fernleihe zuständig.

VuFind bietet die Möglichkeit über eine Schnittstelle diese Systeme einzubinden um zusätzliche Informationen über den Bestand anzeigen zu können, aber auch um das Bestellen und Ausleihen von Büchern zu ermöglichen.

Hierfür wird entweder eine vorgefertigte Konfigurationsdatei mit den nötigen Informationen ausgefüllt, oder ein neuer `Driver` erstellt, der auf das ILS angepasst werden muss. Ein `Driver` bildet hierbei die Middleware zwischen VuFind und dem ILS ab und konvertiert Befehle und Daten zwischen den beiden Systemen.

2.5.3 Konfiguration

Einstellungen werden im `ini`-Format angegeben. Das sind Textdateien, die aus einzelnen Sektionen bestehen. Jede Sektion beginnt mit einem Namen, der in eckigen Klammern angegeben wird. Darauf folgen die Einstellungen im Format 'Einstellung = Wert'.

Die einfache Anpassung von VuFind durch diese textbasierten Konfigurationsdateien ist ein sehr großes Thema. Dabei trennt VuFind die Standardeinstellungen von den lokalen Einstellungen. Diese Trennung erfolgt durch die vorgegebene Ordnerstruktur. Die Datein zum Konfigurieren von VuFind werden zuerst im Ordner 'local/config/vufind' gesucht und dann im Ordner 'config/vufind', wenn sie im 'local'-Ordner nicht existieren. Dadurch werden die Standardeinstellungen durch die lokalen Einstellungen komplett überlagert. Durch die Angabe einer 'Parent_Config'-Sektion mit der Einstellung 'path' oder 'relative_path' kann man eine andere Konfigurationsdatei angeben, die als Grundlage genommen wird. So kann man einzelne Werte überschreiben, ohne die Standardwerte zu duplizieren.

2.6 VuFind-Version im Vergleich

Zwischen der ersten Veröffentlichung von VuFind unter der Versionsnummer 0.5 am 18. Juli 2007 und der aktuellen Version 3.0.0 ist viel Zeit vergangen, und auch VuFind hat sich stark weiterentwickelt. Der Wandel der populären Technologien und Softwarebibliotheken ist deutlich zu erkennen.

VuFind-Version 0.5 bis 1.4

Die ersten Versionen von VuFind enthielten nur wenige Softwarebibliotheken, die eine Basis für die jeweiligen Funktionalitäten von VuFind bilden konnten. So wurde Smarty in der Version 2.6.18 als Template-System eingesetzt, sowie PEAR als eine Erweiterung von PHP. Apache Solr, in der Version 1.2.0, wurde schon von Beginn an als Suchmaschine genutzt und Tomcat als Java-Container-Webserver. Es gab bereits Funktionalitäten, die eine relationale Datenbank benötigten. Hierfür wurde die Berkeley-XML-Datenbank von Oracle eingesetzt. Sonstige grundlegenden Funktionen wurden selbst implementiert. So gab es keine Klasse, die einen Controller repräsentierte und dadurch mehrere Aktionen einer Ressource bündelte, sondern jede Aktion benötigte eine eigene Klasse. Auch das Routing von einer URL auf die jeweilige Aktion wurde durch eigene Rewrite-Regeln umgesetzt.

Anpassungen waren einfach, da das ganze System sehr einfach zu verstehen und überschaubar war. Jedoch mussten diese Anpassungen direkt an VuFind vorgenommen werden. Es gab kein Plugin-System, mit dem es möglich gewesen wäre, neue Funktionalitäten hinzuzufügen oder bestehende zu überladen, obwohl es schon eine Schnittstelle gab, um Informationen mit verschiedenen Bibliothekssystemen auszutauschen. Dadurch führten Updates von VuFind meist zu fehlerhaftem oder inkorrektem Code.

Über die sechs Jahre, die es bis zur Version 1.4 von VuFind brauchte, wurden die Abhängigkeiten Stück für Stück auf den jeweils neusten Stand gebracht. So wurde

beim Release 1.3 von VuFind bereits Solr in der Version 3.5 verwendet. Außerdem wurde SolrMarc für die Indizierung eingeführt und parallel zu den Anforderungen von VuFind mitentwickelt.

Vufind-Version 2.0 bis 2.5.4

Die größte Neuerung gegenüber der vorherigen Version ist die Verwendung des `Zend Frameworks` als Grundgerüst für die ganze Applikation. Dies führte dazu, dass Erweiterungen und Anpassungen von den vorherigen Versionen nicht übernommen werden konnten, sondern komplett neu geschrieben werden mussten. Die Nutzung des `Zend Framework` hob VuFind nicht nur auf aktuelle Standards der Webentwicklung, sondern ersetzte auch veraltete Funktionen, die mit neueren Versionen von PHP nicht mehr genutzt werden sollten.

Die Abhängigkeiten zu Bibliotheken von Drittanbietern werden nun nicht mehr durch ein Installationsskript aufgelöst, stattdessen werden die nötigen Softwarepakete direkt mitgeliefert.

In der zweiten Hauptversion von VuFind wurden hauptsächlich Fehler beseitigt, bestehende Funktionen erweitert und Funktionalitäten von Solr, die durch Updates von Solr hinzukamen, bereit gestellt.

Vufind-Version ab 3.0

Das Update von Solr auf Version 5.5.0 brachte viele Änderungen im mit sich. Diese wirkten sich hauptsächlich durch Modularität auf VuFinds interne Prozeduren aus. Die dadurch notwendigen Anpassungen wurden zum Anlass genommen, den Code von VuFind zu bereinigen.

Abhängige Softwarepakete werden nun wieder nachträglich durch die Installation mithilfe von `Composer` geladen.

2.7 Community und Soziales

Open-Source-Projekte leben von der Mitwirkung Dritter. Das Beisteuern von neuen Ideen sowie das Prüfen und Überarbeiten von bestehenden Features lässt ein solches Projekt auf eine Art und Weise wachsen, wie es die Projektverantwortlichen alleine meist nicht ermöglichen könnten.

VuFind, das auch ein Open-Source-Projekt ist, wird stark durch Dritte unterstützt.

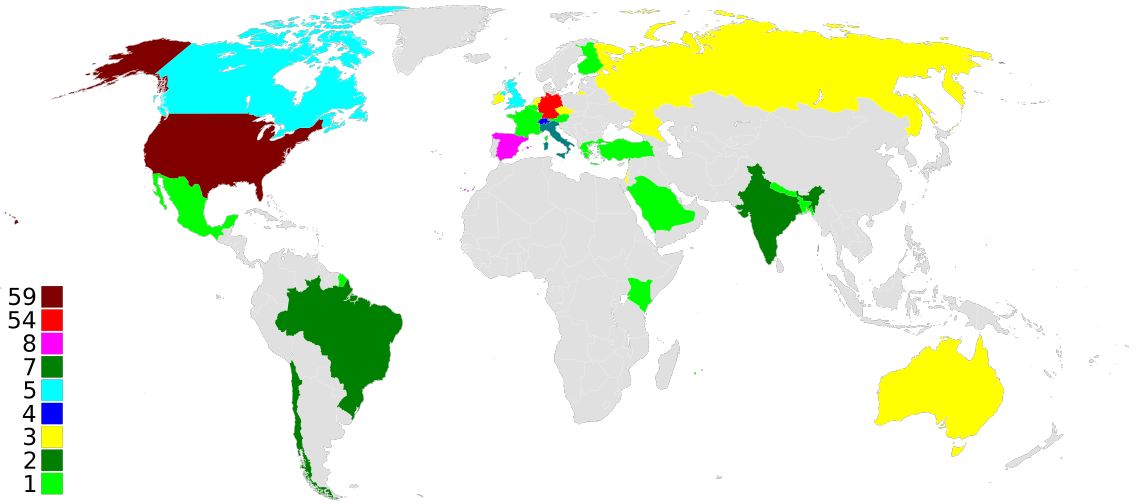


Abbildung 2.15: Weltweite Verbreitung von VuFind im Dezember 2016
Bild: [AMK08]

VuFind wird weltweit eingesetzt. Eine wahrscheinlich unvollständige Tabelle ist im Wiki von VuFind zu finden[VC16]. Von dieser Tabelle ist die Grafik 2.15 abgeleitet, auf der zu sehen ist, dass sich die Anzahl aktiver VuFind-Installationen in den USA und Deutschland ballt. Daher ist es nicht verwunderlich, dass sich die deutschsprachige Community, die aus VuFind-Betreibern aus Deutschland, Österreich und den deutschsprachigen Kantonen der Schweiz besteht, immer stärker engagiert, um ihre Bedürfnisse in den Entwicklungsprozess von VuFind einzubringen. So gibt es seit 2011 jährlich ein VuFind-Anwendertreffen in Deutschland. Bei diesem zweitägigen Event werden von verschiedenen Betreibern einer VuFind-Installation neue Ideen, Fortschritte in der eigenen Entwicklung sowie Wünsche und Verbesserungsvorschläge präsentiert.

Außerdem werden Workshops angeboten, in denen in kleineren Gruppen Themen über oder rund um VuFind vertieft werden. Das Interesse an diesen Treffen steigt stetig. Im Jahr 2016 haben schon etwa 80 Personen verschiedener Universitäten und Bibliotheken daran teilgenommen. Des weiteren sollen neue Kanäle für die Kommunikation zwischen den Mitgliedern der deutschsprachigen Community geschaffen werden. Wurden bisher verschiedene Google-Docs-Dokumente zum Austausch genutzt, soll nun eine eigenständige Plattform für diesen Zweck geschaffen werden.

Der Kontakt mit der internationalen Community wird einerseits über einen E-Mail-Verteiler, andererseits über regelmäßige Meetings per Video-Telefonie realisiert.

Der E-Mail-Verteiler eignet sich vor allem für Fragen zu Problemen bei der Entwicklung, aber auch um zu Erfahren, welche neune Features in der Community entwickelt und möglicherweise Nachgenutzt werden können. Antworten bekommt man meist innerhalb weniger Stunden, oft auch von Demian Katz persönlich, der immer gute Tipps und Vorschläge parat hat.

Die `VuFind Developers Call` genannten Meetings finden meist alle zwei Wochen ab und werden von Demian Katz über die Plattform WebEx und Google Hangouts geführt. Diese Meetings werden genutzt um intensiv die nächsten Schritte für die Entwicklung von VuFind zu besprechen.

Abgesehen von den VuFind-Anwendertreffen im deutschsprachigen Raum werden noch weitere Treffen organisiert. Darunter auch das VuFind Summit, dass jährlich in Villanova, USA statt findet.

3 SolrMarc

Die mit VuFind mitgelieferte Lösung zum Indizieren von MARC-21-Datensätzen ist ein wichtiges Werkzeug, um mit VuFind zu arbeiten.

3.1 Problemdefinition

Sowohl für Bibliotheken des Öffentlichen Dienstes als auch für private Bibliotheken gibt es verschiedene Faktoren, die Einfluss auf den Erwerb und den Betrieb eines Suchmaschinensystems haben. Abgesehen von finanziellen Kriterien, wie Anschaffungs- und Betriebskosten, stehen auch betriebliche Gesichtspunkte im Fokus.

Interessant ist die Skalierbarkeit der Softwarelösung. Einerseits sind hierbei die Anforderungen an die Hardware zu beachten, die benötigt wird, um die zu erwartende Anzahl von Benutzeranfragen zu verarbeiten, selbst für den Fall, dass die Nutzerzahl beliebig steigt. Andererseits sind auch die Auswirkungen auf interne Prozesse zu bewerten, wenn es um die Pflege und den Umgang mit bibliographischen Daten geht.

Die Nutzung von SolrMarc zum Indizieren der bibliographischen Daten stellt einen Faktor dar, der sich sowohl auf die internen Prozesse als auch auf die Hardwareanforderungen auswirkt, da der Indizierungsprozess sehr rechenaufwändig ist und eine starke Belastung für die Massenspeicher darstellt.

Die internen Prozesse werden durch SolrMarc beeinflusst, da die Dauer zwischen Änderungen an den Daten durch die Mitarbeiter und den Auswirkungen für die Anzeige der Suchmaschine den Arbeitsrhythmus des ganzen Systems bestimmt, wobei die

Dauer linear zu der zu indizierenden Datenmenge wächst. Da die Anzahl an Datensätzen für eine bibliographische Suchmaschine meist eine feste Größe ist oder tendenziell eher zunimmt, muss das Softwaresystem, das die Daten verarbeitet, genauer betrachtet werden, um den Einfluss zum Beispiel von SolrMarc zu minimieren.

3.2 Analyse

Es gibt verschiedene Methoden, Software zu analysieren. Die Wahl des richtigen Analyseverfahrens hängt stark vom Ziel der Analyse ab und von dem Wissen über die Software an sich. Im Fall von SolrMarc sollen Bereiche identifiziert werden, die den Betrieb verlangsamen. Hier kann ein Profiler, der das Laufzeitverhalten analysieren kann, einen ersten Überblick geben.

Informationen über das Programm helfen, den richtigen Profiler zu wählen, da je nach Programmiersprache oder Plattform ein spezieller Profiler gewählt werden kann. So ist es für ein Java-Programm wichtig, nicht die Java-Virtual-Maschine zu analysieren, sondern das Programm, das in dieser Virtuellen Maschine ausgeführt wird. Java hat außerdem den Vorteil, dass in den Kompilaten weiterhin menschenlesbare Informationen wie Klassen- oder Methodennamen vorhanden sind, so dass weitere Statistiken über das Laufzeitverhalten geführt werden können.

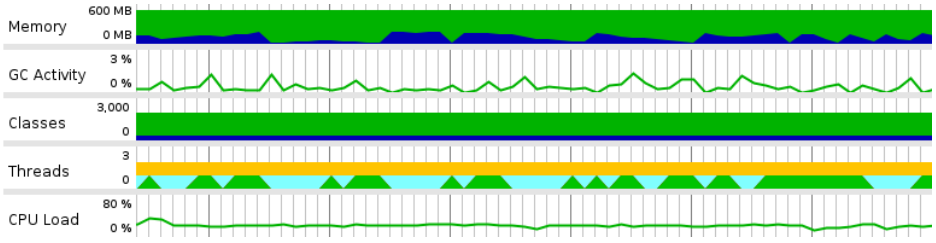


Abbildung 3.1: Profiling von SolrMarc mit JProfile

Für diese Arbeit wurde der JProfiler von ej-Technologies verwendet. Als erstes wurde unter anderem die CPU-Nutzung und der Status einzelner Threads protokolliert. Abbildung 3.1 zeigt das von JProfiler generierte Diagramm.

Die letzte Zeile zeigt die CPU-Auslastung, die mit etwa 25% vermuten lässt, dass SolrMarc durch I/O-Operationen in der Berechnung unterbrochen wird. Das bestätigt die vorletzte Zeile, die den Zustand von insgesamt zwei Threads anzeigt. Der orange Balken bedeutet, dass ein Thread dauerhaft wartet. Der andere Balken gibt an, dass der zweite Thread in den grünen Phasen Berechnungen anstellt und in den blauen Phasen auf Netzwerk-I/O wartet. Dies ist ein wichtiger Hinweis, dem im nächsten Schritt nachgegangen wird.

Da SolrMarc ein Open-Source-Projekt ist, ist es möglich, den Quelltext direkt zu untersuchen und mit zusätzlichen Ausgaben wie den Ergebnissen von Zeitmessungen zu erweitern.

Die Abbildung 3.2 zeigt die Struktur der wichtigsten Klassen von SolrMarc. Zentral ist hierbei `MarcHandler`, da diese Klasse den Ablauf vom Einlesen der Datensätze bis zur Übersetzung in ein Zwischenformat steuert. `MarcImporter` erweitert `MarcHandler` mit dem Wissen, wie dieses Zwischenformat an Solr weitergegeben werden kann. Zusätzlich wird die Funktion für die Nutzung über ein Kommandozeilenprogramm implementiert.

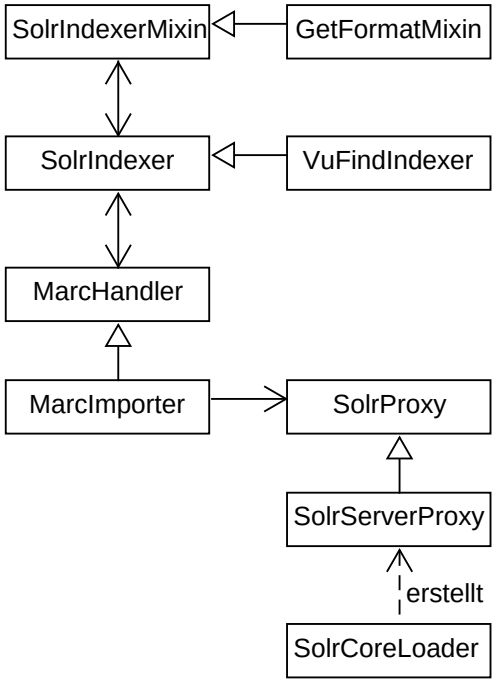


Abbildung 3.2: UML-Diagramm zentraler Klassen von SolrMarc

Die Übersetzung in das Zwischenformat wird von `SolrIndexer` behandelt. Diese Klasse übersetzt die Angaben der Konfigurationsdatei in Übersetzungsroutinen, die dann für jeden Datensatz durchlaufen werden. Die Klasse nutzt so genannte `Mixins`, die durch die Klasse `SolrIndexerMixin` repräsentiert werden, um die Über-

setzungsroutinen zu erweitern. `GetFormatMixin` ist dabei das einzige `Mixin`, dass von SolrMarc stammt und hart kodiert ist.

Für die Verwendung mit VuFind wird die Klasse `SolrIndexer` durch die Klasse `VuFindIndexer` erweitert und in den Konfigurationsdateien ersetzt. In dieser Spezialisierung werden spezifische Methoden angeboten, um weitere Informationen aus den Datensätzen zu extrahieren. Unter anderem werden Informationen über das Indizierungsdatum in einer separaten Datenbank gespeichert, so dass diese über mehrere Indizierungen hinweg persistent vorgehalten werden.

Durch die Analyse der Struktur ist zuerkennen, dass die Klasse `MarcHandler` geeignet ist, den Ablauf mit Hilfe von Zeitmessungen zu untersuchen. Hierbei ist die Methode `addToIndex(Record)` aus der Klasse `MarcImporter` prädestiniert, da sie den Ablauf zwischen dem Extrahieren der Daten aus dem Marc-Datensatz und dem Übertragen der gewonnenen Daten an Solr steuert. In der Tabelle 3.1 werden die Ergebnisse von zehn Messungen aufgezeigt, die die Dauer der beiden Bereiche ermittelten.

	Minimum	Durchschnitt	Maximum
Extrahieren	57,3	59,3	71,9
Übertragen	59,8	65,9	89,4

Tabelle 3.1: Zeitmessung zweier Hauptbestandteile von SolrMarc in Minuten

Die Messungen ergeben, dass während etwa der Hälfte der Laufzeit von SolrMarc darauf gewartet wird, dass die extrahierten Daten an Solr weitergeleitet werden. Eine Parallelisierung könnte die Gesamtlaufzeit also halbieren.

Durch die Zeitmessung ist die Klasse `SolrIndexer` als zentraler Punkt für Optimierungen identifiziert worden. Diese Klasse enthält 84 Methoden. Sieben Methoden sind für das Auswerten der Konfigurationsdateien vorgesehen, die restlichen können beim Konfigurieren der Indizierung durch einen Nutzer angegeben werden.

Abbildung 3.3 listet diese Namen der Methoden auf, wobei überladene Methoden durch die Anzahl in den Klammern repräsentiert wurden.

`fillMapFromProperties` ist die Methode, welche die Angaben der Konfigurationsdateien aufspaltet und vorhält, damit einerseits der Ablauf der einzelnen Übersetzungsroutinen daraus abgeleitet werden kann, andererseits aber die jeweiligen Routinen weitere Informationen extrahieren können.

SolrIndexer	
+addFieldValueToMap	+getSubfieldDataCollector (2x)
+findMap	+getSupplUrls
+findMixin	+getTitle
+getAllAlphaExcept	+getWithOptions
+getAllAlphaSubfields (2x)	+indexerFromProperties
+getAllSearchableFields	+isControlField
+getAllSearchableFieldsAsSet	+loadTranslationMap
+getAllSubfields	+map (4x)
+getAllSubfieldsAsList	+reinitFromProperties
+getAllSubfieldsCollector	+removeTrailingPunct
+getCurrentDate	+addField (2x)
+getDataFromVariableField	+addFields (2x)
+getDate	+fillMapFromProperties
+getEra (2x)	+getAlphaSubfldsAsSortStr
+getErrorHandler	+getInd2AsInt
+getFieldList	+isSupplementalUrl
+getFieldListAsList	+loadTranslationMap
+getFieldListCollector	+perRecordInit
+getFieldSetMatchingTagList (2x)	+writeJson
+getFieldVals	+writeRaw
+getFirstFieldVal (2x)	+writeXml
+getFormat	-addIndexerExceptionAsField
+getFormatMapped	-dequote
+getFullTextUrls	-finishCustomOrScript
+getLinkedField	-getInterpreterForScript
+getLinkedFieldCombined	-getStd
+getLinkedFieldValue	-getXPathFieldList
+getLinkedFieldValueAsList	-handleCustom
+getLinkedFieldValueCollector	-handleScript
+getPublicationDate	-loadTranslationMapValues (2x)
+getSingleIndexEntry	-localParseInt
+getSolrId	-parseSinglefieldSpec
+getSortableAuthor	-perRecordInitMaster
+getSortableTitle	-verifyCustomMethodExists
+getSubfieldDataAsList (2x)	-verifyCustomMethodsAndTransMaps
+getSubfieldDataAsSet (2x)	

Abbildung 3.3: UML-Diagramm von SolrIndexer

Aus der Analyse des Quelltextes und den oben beschriebenen Eigenschaften ergeben sich vier Punkte, die als ineffiziente Implementierung identifiziert wurden, wobei der letzte Punkt der kritischste ist:

- Im Bereich der so genannten `Mixin`s, die Plugins zum Erweitern des Funktionsumfangs darstellen, werden diese vorgehaltenen Konfigurationen genutzt, um mittels Reflections die aufzurufende Methode zu identifizieren.
- Es werden vorwiegend Reguläre Ausdrücke genutzt, um die Konfigurationen in die benötigten Informationsfragmente zu zerlegen.
- Viele Bereiche sind durch eigenständige Try-Catch-Blöcke abgesichert.
- Die Ergebnisse der vorherigen drei Punkte werden nicht gespeichert, sondern für jedes zu indizierende Feld und jeden Datensatz neu berechnet.

Reflections, Reguläre Ausdrücke und die Fehlerbehandlung mit Try-Catch-Blöcken sind sehr rechenaufwändig und sollten daher in frequentierten Bereichen mit Bedacht eingesetzt werden. Da bei SolrMarc die Konfigurationen nur einmal gelesen und dann nicht geändert werden und daher für jeden zu indizierenden Datensatz gleich sind, ist es möglich, diese Operationen zu Beginn des kompletten Indizierungsvorgangs durchzuführen und vorzuhalten.

3.3 Lösungsvorschläge

Um eine Verbesserung der Situation um SolrMarc zu erreichen, können verschiedene Wege beschritten werden. In dieser Arbeit werden mehrere Ansätze dargelegt. Dabei wird eine Lösung gesucht, die auf der einen Seite das Indizieren von Marc-Datensätzen effizienter gestaltet, auf der anderen Seite aber nur ein Update ohne weitere Anpassungen benötigt, um eine hohe Akzeptanz bei bestehenden VuFind-Installationen zu erreichen.

Zuerst werden zwei exemplarische Alternativen aufgezeigt, um zu prüfen, ob bestehende Lösungen als Ersatz von SolrMarc genutzt werden können. Der zweite Ansatz versucht SolrMarc durch parallele Verarbeitung zu optimieren, ohne SolrMarc anzupassen. Als letztes werden die Ergebnisse der Analyse dazu genutzt, SolrMarc zu überarbeiten.

Um eine Vergleichbarkeit der Ergebnisse zu erlangen, werden alle Tests auf demselben Computer durchgeführt, der mit einem Intel Core i7-4790 mit 3,6 GHz, 32 GB Arbeitsspeicher und einer Solid State Drive (SSD) ausgerüstet ist. Am Ende dieses Kapitels wird noch darauf eingegangen, welche Einflüsse eine Festplatte (HDD), weniger Arbeitsspeicher oder ein langsamerer Prozessor auf die Ergebnisse haben.

Die Ergebnisse werden, soweit nicht anders angegeben, durch die Anzahl von Dokumenten angegeben, die pro Sekunde indiziert wurden. Dadurch sind die Ergebnisse von der Anzahl der Dokumente unabhängig, die für den jeweiligen Test indiziert wur-

den. Da die Indizierung von der Anzahl an indizierten Feldern pro Datensatz und der Komplexität der einzelnen Indizierfunktionen abhängt, werden die Indizierungskonfigurationen und der Solr-Index vom `IxTheo`-Projekt als Grundlage genutzt.

3.3.1 XML Transformation

Durch das Marc-XML-Format, das aus Marc-21 generiert werden kann, können existierende Tools für die Verarbeitung von XML verwendet werden. Mit `XSLT` und `xPath` ist es möglich, Daten im XML-Format in viele verschiedene Formate zu überführen. Dadurch ist es möglich, Marc-XML in das Import-Format von Solr zu konvertieren und in Solr einzuspielen. Positiv zu bewerten ist, dass `XSLT` eine bewährte Technologie ist. Es ist unwahrscheinlich, dass Bugs in `XSLT` existieren, welche die Arbeit mit diesem Werkzeug behindern, da `XSLT` weit verbreitet ist. Aus demselben Grund kann man davon ausgehen, dass die Algorithmen hinter `XSLT` effizient arbeiten.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:marc="http://www.loc.gov/MARC21/slim"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:output method="xml"/>
   <xsl:template match="/">
      <add><doc><xsl:apply-templates/></doc></add>
   </xsl:template>
   <xsl:template match="marc:controlfield[@tag=001]">
        <field name="id"><xsl:value-of select="text()"/></field>
   </xsl:template>
</xsl:stylesheet>
```

Abbildung 3.4: Beispiel für XLST

Das Beispiel 3.4 extrahiert aus einem einzelnen Marc-XML-Datensatz nur die Identifikationsnummer und erstellt eine XML-Datei, mit der diese Identifikationsnummer zu einem Solr-Index hinzugefügt werden kann.

Dieser Ansatz hat allerdings zwei Nachteile:

- `XSLT` ist eine für den allgemeinen Zweck konzipierte Sprache. Das führt dazu, dass komplexe, schwer lesbare Ausdrücke genutzt werden müssen, um Transformationen durchzuführen, die mit Hilfe eines spezialisierten Systems, wie SolrMarc, durch kurze, gut lesbare Befehle beschrieben werden können.

- Marc-XML benötigt etwa fünf mal mehr Speicherplatz als Marc-21. Dies führt zu höheren Hardwareanforderungen im Bereich der Massenspeicher und zu einer langsameren Verarbeitung, da das Lesen und Verarbeiten der Daten um etwa denselben Faktor mehr Zeit benötigt.

3.3.2 Traject

Neben SolrMarc und technischen Lösungen wie `XSLT` gibt es weitere Projekte aus der Open-Source-Welt, mit denen sich Marc-Datensätze in Solr importieren lassen. Traject ist ein Projekt, das vom Verwendungszweck und vom Funktionsumfang am besten mit SolrMarc vergleichbar ist, da es seinen Ursprung in SolrMarc hat. Das Projekt startete 2011 unter dem Namen marc2solr und ist in `Ruby` geschrieben. Ende 2013 wurde marc2solr in Kooperation mit einem zweiten Entwickler komplett überarbeitet und ist seitdem als Traject bekannt.

Auch Traject nutzt mindestens eine Konfigurationsdatei, über die viele Einstellungsmöglichkeiten gegeben werden, um den Ablauf der Indizierung zu bestimmen. Diese Datei kann beliebig aufgeteilt werden, so dass logische Partitionen erstellt und ausgetauscht werden können.

Die Konfigurationsdatei ist ein Ruby-Skript. Das bedeutet, dass Felder des Solr-Indexes mit den Rückgabewerten von Ruby-Funktionen gefüllt werden. Diese Funktionen können durch spracheigene Konstrukte hinzugeladen werden, wodurch eine einfache Erweiterbarkeit und Anpassbarkeit gewährleistet wird.

Minimum	Durchschnitt	Maximum
1199,22	1257,83	1289,20

Tabelle 3.2: Durchsatz von Traject in Dokumente pro Sekunde.

Tabelle 3.2 gibt einen Überblick über die Anzahl an Dokumenten, die pro Sekunde von Traject indiziert wurden. Gegenüber SolrMarc ist Traject etwa um den Faktor 4 leistungsfähiger und zeigt das Potential, dass in SolrMarc steckt.

Traject ist jedoch trotz seiner Leistung und der einfachen Anpassbarkeit kein praktischer Ersatz für SolrMarc, da ein Umstieg auf Traject einen hohen Aufwand bedeuten würde. Abgesehen davon, dass die Konfigurationen zum Indizieren neu geschrieben und möglicherweise personalisierte Indizierungsfunktionen übertragen werden müssten, würde die Überprüfung des kompletten Systems Zeit und Geld in Anspruch nehmen, die meist nicht zur Verführung stehen. Zusätzlich bedeutet ein Umstieg auf Ruby eine Änderung in den Kompetenzansprüchen, die einige Entwicklerteams dann nicht mehr erfüllen können.

3.3.3 Parallelität ausnutzen

Solr ist in der Lage, mehrere Updates gleichzeitig zu empfangen und zu verarbeiten. SolrMarc hingegen besitzt keine eingebauten Routinen, um das Indizieren von Marc-Daten zu parallelisieren. Jedoch ist es ohne weiteres möglich, mehrere Instanzen von SolrMarc zu starten und verschiedene Marc-Dateien gleichzeitig zu indizieren.

SolrMarc sollte hierfür nach Beendigung der Indizierungsphase einer Instanz keine Optimierung des Solr-Cores anstoßen. Die Optimierung sollte erst nach dem Beenden

aller Instanzen von SolrMarc durchgeführt werden, da sonst die Belastung für Solr steigt und sich die Gefahr eines Timeouts erhöht.

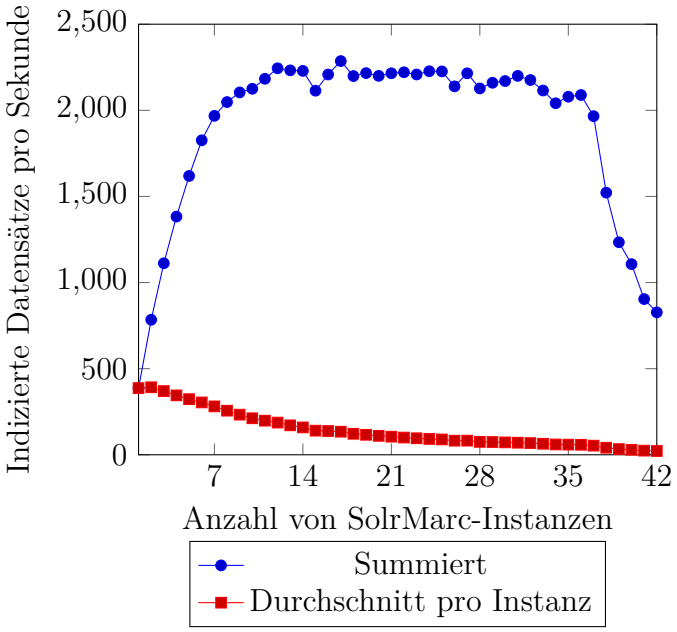


Abbildung 3.5: Durchsatz bei paralleler Ausführung von SolrMarc

In Abbildung 3.5 ist der durchschnittliche Durchsatz einer Instanz und der Gesamtdurchsatz in Abhängigkeit der Anzahl von Instanzen aufgezeigt. Wie nach der Analyse zu erwarten war, ist es durch diese Art der Parallelisierung möglich, den Durchsatz zu steigern, da die jeweilige Instanz von SolrMarc etwa die Hälfte der Laufzeit auf I/O-Operationen wartet. Es zeigt sich eine Leistungssteigerung um den Faktor 5,7. Der Leistungseinbruch bei den letzten sechs Messungen wird durch das Ausreizen des gesamten verfügbaren Arbeitsspeichers hervorgerufen.

3.3.4 Überarbeiten von SolrMarc

Bei der Analyse der Struktur von SolrMarc und deren Import-Konfigurationsdateien wurde ersichtlich, dass es drei Stufen gibt:

- Die erste Stufe extrahiert Daten aus dem aktuellen Marc-Datensatz und überträgt diese Daten als Zeichenkette oder als Liste von Zeichenketten an die nächste Stufe. Sie besteht aus vielen kleinen Funktionen, die jeweils für das Extrahieren von bestimmten Datenfragmenten zuständig sind. Außerdem ist vorgesehen, dass neue Funktionen hinzugefügt werden können, sodass eine Anpassung auf eigene Bedürfnisse ermöglicht wird.

- Das Mapping überführt als zweite Stufe Zeichenketten, die in der vorherigen Stufe extrahiert wurden, mit Hilfe einer Mapping-Datei oder einem Regulären Ausdruck in eine andere Zeichenkette.

- Zuletzt kann noch entschieden werden, ob aus einer Liste von Zeichenketten eine einzelne Zeichenkette gemacht wird, indem nur die erste Zeichenkette genommen wird, oder ob alle Zeichenketten mit einem angegebenen Zeichen konkateniert werden.

Diese drei Stufen und eine Vielzahl von Funktionen der ersten Stufe sind in SolrMarc in einer einzigen Klasse implementiert. Das führt zu unübersichtlichem Code und einer heterogenen Implementierung der einzelnen Features. Das ist daran ersichtlich, dass bestimmte Funktionalitäten an mehreren Stellen auf ähnliche Art und Weise implementiert wurden, ohne den redundanten Code zu vereinheitlichen. Ein Beispiel dafür stellt die Analyse von Parameterangaben für bestimmte Funktionen dar, die in den Konfigurationsdateien für die Indizierung angegeben werden können. Die fehlende Struktur führte auch dazu, dass die genannte Analyse für jedes Feld und jedem Marc-Datensatz erneut durchgeführt werden musste, obwohl diese Informationen für einen

Indizierungsdurchlauf unveränderlich sind und nur ein einziges Mal analysiert werden müssen.

In Abbildung 3.6 ist ein UML-Diagramm gezeigt, das als Vorschlag für eine Umstrukturierung von SolrMarc genutzt wird. Zusehen ist die Klasse `AbstractValueIndexer`, welche die Indizierung für ein Solr-Feld repräsentiert. Ein solches Objekt besitzt drei weitere Arten von Objekten, die die drei Stufen darstellen. Jedes dieser Objekte wird zum Beginn von `Factories` erstellt. Diese `Factories` besitzen dabei alle Informationen aus den Konfigurationsdateien. Diese Informationen werden zu Indizieren kompiliert und gehen danach verloren. So soll sichergestellt werden, dass die jeweiligen Indizierer mit allen statischen Daten versorgt werden und nicht während der laufenden Indizierung immer wieder Konfigurationsdaten analysieren. Außerdem verringert sich dadurch die Menge des verwendeten Arbeitsspeichers, da keine unnötigen Informationen gehalten werden müssen.

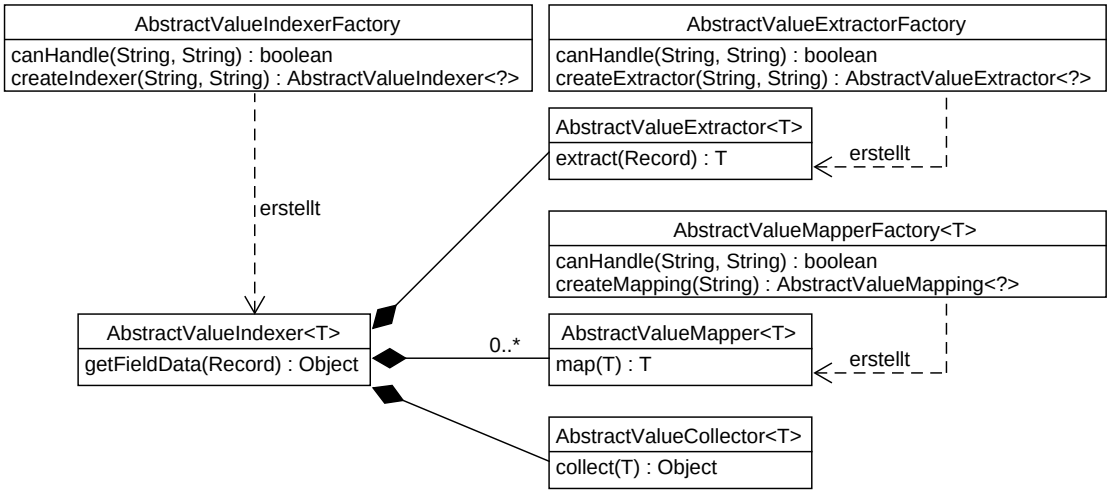


Abbildung 3.6: Custom Level

Die `Factories` werden durch `Reflections` im Klassenpfad gesucht. Dadurch können dynamisch neue `Factories` hinzugefügt werden, ohne den Quelltext anpassen zu müssen. Jede `Factory` besitzt deshalb eine `canHandle`-Methode, die angibt, ob

ein Eintrag in der Indizierungskonfiguration durch diese `Factory` verarbeitet werden kann.

Jedoch kommt von Solr eine Anforderung, welche die Komplexität erhöht: Die an Solr gesendeten Daten müssen dann Listen von Zeichenketten sein, wenn in der `solrconfig`-Konfigurationsdatei das jeweilige Feld als `multivalue` markiert ist, andernfalls nur einfache Zeichenketten. Daher muss von Beginn an klar sein, welche Art von Rückgabewert der jeweilige Indizierer erzeugt. Dadurch kommen die in Abbildung 3.7 gezeigten Kinderklassen zustande.

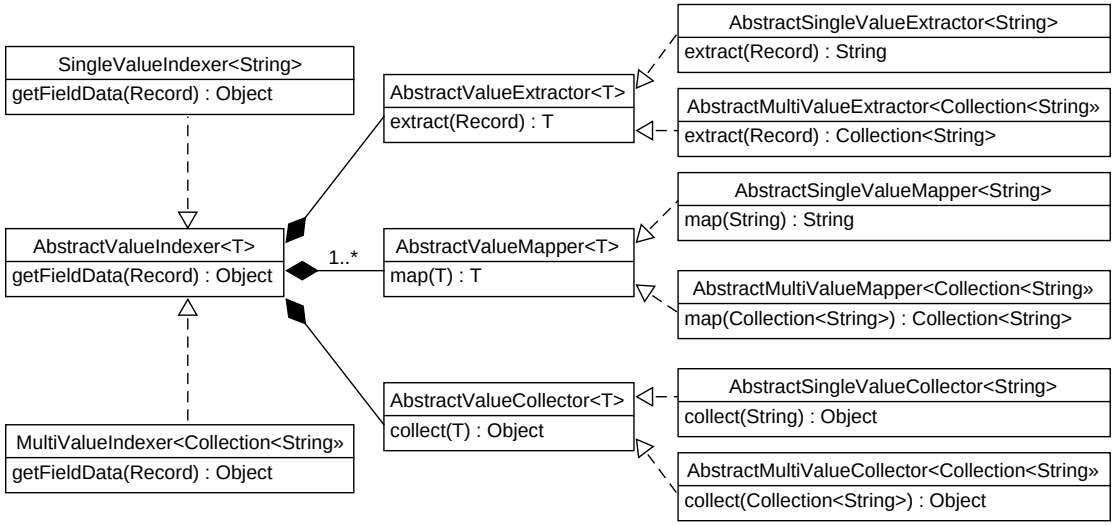


Abbildung 3.7: Custom Level

Das führt einerseits dazu, dass Funktionen, die auf einfache Zeichenketten und für Listen von Zeichenketten angewendet werden können, doppelt implementiert werden müssen. Andererseits ergeben sich dadurch Dokumentations- und Optimierungsmöglichkeiten. Als Beispiel extrahiert der `DirectMultiValueExtrator` mehrere Unterfelder von verschiedenen Marc-Feldern. Daher wird immer eine Liste von Zeichenketten erzeugt, die dann mit einem passenden `AbstractValueCollector` nachverarbeitet werden muss.

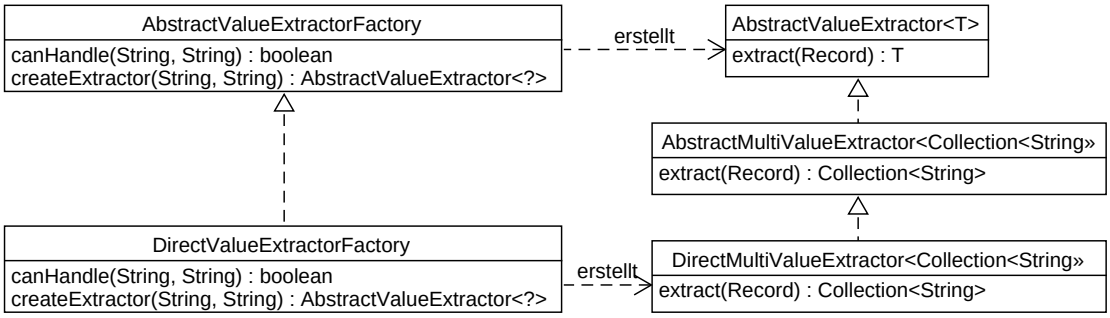


Abbildung 3.8: Custom Level

Abbildung 3.8 zeigt die einfache Struktur eines solchen `Extractors`. Durch die Klassenstruktur ist sofort ersichtlich, welche Art von Ergebnis geliefert wird.

SolrMarc verfügt über die Möglichkeit, so genannte `Mixins` zu benutzen. Das sind Klassen, die Methoden enthalten, die einen Marc-Datensatz und sonst nur Zeichenketten als Argument nehmen und eine Zeichenkette oder eine Liste von Zeichenketten zurückliefern. Die `Mixin`-Funktionalität ist in SolrMarc recht umständlich zu nutzen. Es gibt zwei Möglichkeiten:

- Man kann eine abgeleitete Klasse der Klasse `SolrIndexer` erstellen, in der die Funktionen implementiert werden. Dadurch können Mixins von dritten nur durch Anpassung der Vererbungsreihenfolge eingebunden werden.

- Es kann eine Klasse, die von `SolrIndexerMixin` abgeleitet wurde, in einem Java Archiv (JAR-Datei) abgelegt und in den Konfigurationen benannt werden. Jedoch muss der Name dieser Klasse bei jedem Methodenaufruf in der Konfiguration mit der vollen Nennung des Pakets angegeben werden, wodurch die Konfigurationsdatei wieder unübersichtlich wird.

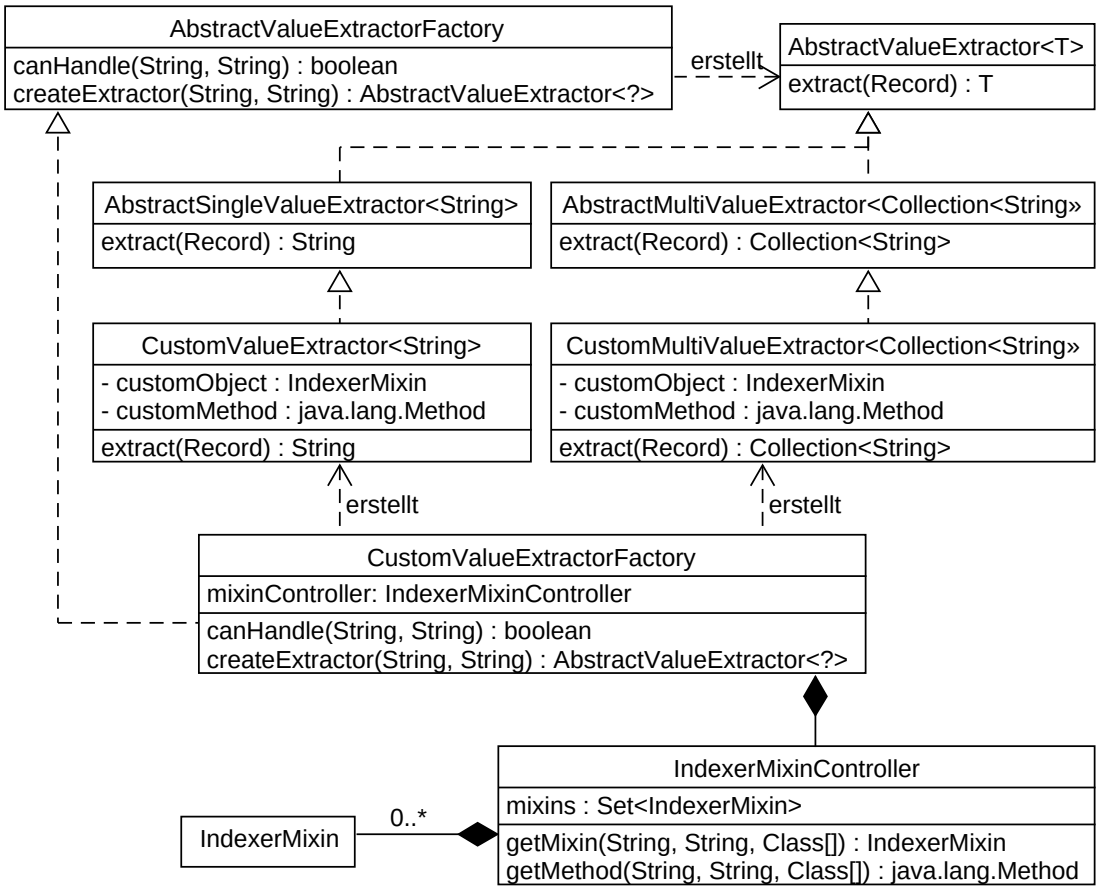


Abbildung 3.9: Custom Level

Wie in Abbildung 3.9 zu sehen ist, können diese beiden Wege mit Hilfe von Reflections vereinfacht werden, da im Klassenpfad direkt nach Implementationen der Klasse `IndexerMixin` gesucht werden kann. So können diese Klassen dynamisch nachgeladen und initialisiert werden. Weiter wird Reflections genutzt, um innerhalb der Implementierung nach passenden Methoden zu suchen.

Methoden, die für die Nutzung als Indizierungsroutinen infrage kommen, müssen drei Eigenschaften haben:

- Es muss eine öffentliche, nicht-statische Methode sein.
- Sie muss als erstes Argument ein `org.marc4j.Record` entgegen nehmen und sonst nur `String`s.
- Sie muss einen `String` oder eine `Collection<String>` zurückgeben.

Der Kern dieser Funktionalität stellt die Klasse `IndexerMixinController` bereit. Diese hält alle Instanzen von `IndexerMixin` und extrahiert aus den Klassen dieser Instanzen die passenden Methoden. Wird in der Konfiguration nun eine Methode eines `IndexerMixin`s durch den Klassen- und Methodennamen angegeben, so wird der `IndexerMixinController` nach einer passenden Instanz und Methode durchsucht. Nur im Fall, dass mehr als eine passende Methode gefunden wurde, muss der vollständige Paket- und Klassenname in der Konfiguration angegeben werden.

Für die Kommunikation mit Solr kommt SolrJ in der Version 5.3.1 zum Einsatz. Im Gegensatz zu der Version, die in SolrMarc genutzt wird, bieten neuere Versionen die Möglichkeit, Anfragen in separaten Threads auszuführen. Außerdem können in einem Batch-Vorgang mehrere Datensätze gebündelt an Solr übergeben werden, anstatt jeden Datensatz einzeln zu übermitteln.

Diese Vorschläge wurden implementiert und durch dieselben Tests, wie SolrMarc und Traject, analysiert. Die Ergebnisse, die in Tabelle 3.3 aufgeführt sind, zeigen eine Steigerung des durchschnittlichen Durchsatzes um Faktor 5,8 gegenüber SolrMarc und 1,8 gegenüber Traject.

Minimum	Durchschnitt	Maximum
2213,35	2256,49	2295,13

Tabelle 3.3: Durchsatz der überarbeiteten Version in Dokumente pro Sekunde.

Die Überarbeitung sah keine Parallelisierung auf der Ebene der Indizierer oder auf der Ebene der Datensätze vor. Durch dieselbe Methode, mit der SolrMarc in Kapitel 3.3.3 parallel ausgeführt wurde, kann geprüft werden, ob eine Parallelisierung der überarbeiteten Version eine Leistungssteigerung bewirkt.

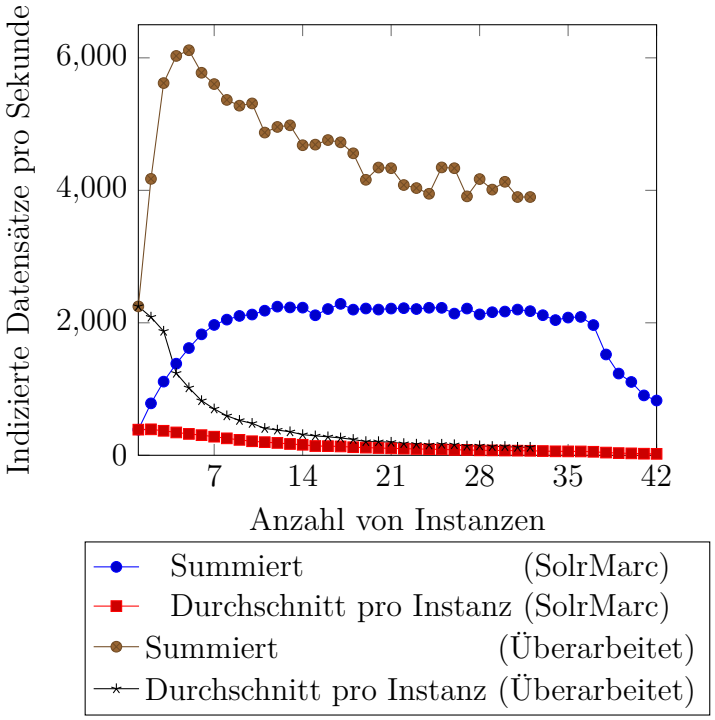


Abbildung 3.10: Durchsatz bei paralleler Ausführung im Vergleich

Das Diagramm von Abbildung 3.10 zeigt, dass durch die parallele Ausführung der Durchsatz um etwa Faktor 2,7 erhöht werden kann. Daher liegt eine Parallelisierung auf der Ebene der Indizierer oder auf der Ebene der Datensätze nahe.

3.3.5 Auswirkungen anderer Hardware

Das Testsystem, das mit einem Intel Core i7-4600U mit 3,6 GHz, 32 GB Arbeitsspeicher und einer Solid State Drive (SSD) ausgerüstet ist, ist sehr leistungsstark. Im Serverbereich sind jedoch durch den geringeren Preis und höheren Kapazität eher Festplatten als SSDs zu finden. Auch kann bei kleineren Bibliotheken von weniger

leistungsstarken Prozessoren und weniger Arbeitsspeicher ausgegangen werden. Daher muss untersucht werden, welcher dieser Faktoren den Durchsatz von SolrMarc und der überarbeiteten Version beeinflusst.

Also wurden die Tests auf einem System mit einem Intel Core i7-4600U mit 2,1 GHz, 12 GB Arbeitsspeicher und einer SSD wiederholt, um die Einflüsse der schwächeren CPU und dem geringeren Arbeitsspeicher zu untersuchen.

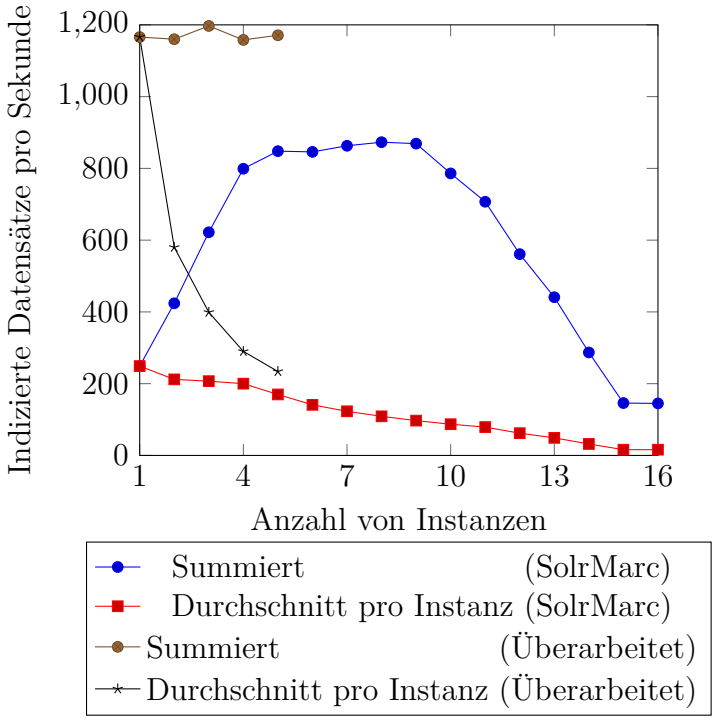


Abbildung 3.11: Durchsatz mit langsamerer CPU und geringerem Arbeitsspeicher im Vergleich

Prozessor

Beim Indizieren von bibliographischen Daten wird vor allem Zeichenkettenmanipulation betrieben. Diese Operationen sind sehr rechenintensiv. Das Diagramm aus Abbildung 3.11 zeigt, dass durch die langsamere CPU der Durchsatz geringer ausfällt. Außerdem zeigt sich bei der überarbeiteten Version, dass die Beschränkung von vier auf zwei Cores, trotz Hyperthreading-Technologie die Steigerung beim parallelen Aus-

führen komplett zunichte macht, da Solr während der Indizierung gleichzeitig laufen muss, um die Datensätze in den Index einzutragen.

Arbeitsspeicher

Während der Ausführung wird pro Instanz etwa 700 MB Arbeitsspeicher belegt, wobei die tatsächliche Menge sowohl von der Größe der einzelnen Datensätze als auch von der Komplexität des Indexes abhängt. Da anzunehmen ist, dass Computersysteme aktuell über mehr als 2 GB Arbeitsspeicher verfügen, sollte dies keinen Einfluss auf die Ausführungsgeschwindigkeit der Indizierung haben.

Bei der parallelen Ausführung der Tests kam es jedoch zu starken Leistungseinbrüchen, da durch die vielen Instanzen mehr Arbeitsspeicher benötigt wurde, als verfügbar war. Daher musste das Betriebssystem beginnen, Arbeitsspeicher auf die Festplatte auszulagern. Diese Operation bremst das System so stark aus, dass es zu Leistungseinbürchen bei der Indizierung kommt.

Festplatte

Solid State Drives erreichen aktuell eine maximale Lese- und Schreibgeschwindigkeit von etwa 500 MB/s. Bei magnetischen Festplatten variiert diese Zahl stark. Die genutzte Festplatte erreichte eine maximale Lese- und Schreibgeschwindigkeit von etwa 100 MB/s.

Die Marc-21-Datensätze hatten ein Gesamtvolumen von 3,6 GB und enthielten 1,6 Millionen Datensätze. Selbst auf dem Originaltestsystem, das 2248 Datensätze pro Sekunde verarbeitet, werden im Schnitt etwa 5 MB/s und beim maximalen Durchsatz von 6115 Datensätzen pro Sekunde 13,6 MB/s gelesen.

Einflüsse auf den Durchsatz durch die Festplatte sind daher auszuschließen.

4 Betriebskonzepte

4.1 Hessische Bibliotheks Informations System (HeBIS)

Das HeBIS verwaltet als Bibliotheksverbund in Hessen und Rheinhessen einen gemeinsamen Katalog mit über 38 Millionen Dokumenten und bietet zusätzlich diverse Dienstleistungen an.

In Zusammenarbeit mit der HeBIS-IT, die als Rechenzentrum die technische Kompetenz stellen, wird das HeBIS Discovery System (HDS) bereitgestellt. Das HDS ist ein Projekt, dass Ende 2011 ins Leben gerufen wurde, um das Verbundportal sowie die Portale der dem Verbund angehörenden Bibliotheken abzulösen und zu vereinheitlichen. Die technische Umsetzung wurde auf der Basis von VuFind vorgenommen. Durch die Möglichkeit zur Einbindung verschiedener Schnittstellen in VuFind konnten bestehende Systeme nachgenutzt werden. So wird Shibboleth für die Authentifizierung, DAIA für den Austausch mit den jeweiligen lokalen Ausleihsystemen und eine Modul für die Einbindung der Bestellfunktionen von Pica-OUS genutzt. Außerdem wird der von Regionale Datenbank-Information Baden-Württemberg (ReDI) entwickelte Link-Resolver genutzt und das Elektra-Protal für die Fernleihe eingebunden[Bib].

Für das HDS wird ein stark angepasstes VuFind in der Version 1.3 genutzt, wobei ein Umstieg auf die aktuellste Version vorbereitet wird. Dieses wird auf den Servern der HeBIS-IT gehostet. Dabei kommen insgesamt drei Virtuelle Maschinen (VM) zum

Einsatz. Die erste VM beherbergt vier Solr-Indizes. Dabei wird der erste Index zum Einspielen von Updates genutzt, während die drei anderen Indizes für die Recherche bereit stehen. Diese VM verfügt über 16 CPUs, 256 GB Arbeitsspeicher und SSDs als Massenspeicher um die Recherche beim Indizierungsprozess nicht zu beeinträchtigen. Die zweiten VM wird ausschließlich für VuFind und somit für die Oberfläche genutzt. Diese ist mit 8 CPUs und 32 GB ausgestattet. Die letzte VM beherbergt einen Load Balancer, der die Last auf die drei Solr-Indizes verteilen, die für die Recherche verwendet werden[Reh17].

Seit der Testphase im Jahr 2013 wird das HDS inzwischen von mehr als 30 Bibliotheken genutzt.

Die fünf größten Mitglieder bezüglich der Anzahl an Titel sind [Reh17]:

- UB Frankfurt 7,4 Millionen Titel
- UB Gießen 4,2 Millionen Titel
- UB Meinz 3,9 Millionen Titel
- UB Marburg 3,7 Millionen Titel
- ULB Darmstadt 3,5 Millionen Titel

Die Mitglieder des HDS organisieren sich in einem Gremium, indem Entscheidungen über größere Anpassungen getroffen werden. Kleinere Änderungen, wie das Anpassen des Rankings und von statischen Texten können jedoch unabhängig vom Gremium vorgenommen werden. Das HDS-Projekt ist dabei dem HeBIS-Verbundrat unterstellt, dass die Entscheidungsgewalt über das Bestehen des Projektes hat.

Für den Aufbau des Indexes sind mehrere Schritte notwendig:

- Die Datensätze werden aus dem Zentralkatalog im Pica- und Marc-Format abgezogen. Dies benötigt 20 Stunden.

- Die Daten durchlaufen einen Vorverarbeitungsschritt, der Informationen aus den Titeldaten in Hilfsdateien auslagert, die für den nächsten Verarbeitungsschritt benötigt werden. Dies dauert etwa 2 Stunden.

- Die Titel werden in diesem Schritt angereichert und indiziert. Dies braucht etwa 8 Stunden.

Somit benötigt die Indizierung aller Daten etwa 30 Stunden. Da der Index jedoch Tagesaktuell sein soll wird eine vollständige Indizierung nur bei Änderungen an der Struktur des Indexes vorgenommen. Im Normalfall werden täglich nur Datenänderungen, Neueinträge und Löschungen in den Index eingespielt. Hierbei ist die Dauer abhängig von der Anzahl der Änderungen.

4.2 Finc

Im Jahr 2011 wurde von der Universitätsbibliothek Leipzig ein Projekt zur Entwicklung einer suchmaschinenbasierten Katalogoberfläche für die sächsische Hochschulbibliotheken unter dem Namen finc ins Leben gerufen. [Wei11] Der Europäische Fonds für regionale Entwicklung hat dieses Projekt bis ins Jahr 2014 mit einem Fördervolumen von 1,3 Millionen Euro gefördert. Inzwischen wird das Projekt durch die Mittel der Mitglieder getragen. Der flinc-Gemeinschaft gehören aktuell 16 Mitglieder aus Sachsen, Nordrhein-Westfalen und Schleswig-Holstein an. Darunter folgende Bibliotheken [fin]:

- UB Leipzig

- Christian-Albrechts-Universität zu Kiel

- Robert Schumann Hochschule Düsseldorf

- Berufsakademie Sachsen

- TU Freiberg

Geleitet wird das finc-Projekt durch die UB Leipzig. Auch das Hosting und die Entwicklung findet in Leipzig statt. Dabei können die Mitglieder auf verschiedene Art Einfluss auf finc nehmen. Sie können Vorschläge für Neuerungen aber auch Eigenentwicklungen einbringen. Außerdem sind alle Mitglieder dazu aufgerufen den gemeinsamen Index zu erweitern.

Zu Bemerken ist, dass Mitglieder teilweise nicht alle Bereiche des finc-Projekts nutzen. Zwar teilen sie sich einen Index, die Rheinisch-Westfälische Technische Hochschule Aachen betreibt jedoch ihre eigene Weboberfläche und hat keine Lokalsystemanbindung ebenso wie fünf weitere Mitglieder.

Der Suchmaschinenindex des finc-Projekts ist ein aggregierter Index. Hierbei werden die Datensätze der lokalen Bestände der jeweiligen Bibliotheken zusammengeführt und durch den kommerziellen Index Primo Central der Firma Ex Libris erweitert [Sei12]. Dadurch enthält der Index über 100 Millionen Datensätze [Mus14].

Seit Ende 2016 wird das finc-Projekt auf Grundlage der aktuellen Version von VuFind betrieben. Dabei ist der Source-Code des gesamten Projektes öffentlich auf GitHub.com einsehbar.

4.3 Integriertes Bibliothekssystem Baden-Württemberg (IBS|BW)

Im Zuge des Projektes IBS|BW wurde das integrierte Bibliothekssystem aDIS/BMS der Firma a|S|tec an vielen Bibliotheksstandorten in Baden-Württemberg eingeführt. Ende 2014 haben 62 Bibliotheken dieses Produkt eingesetzt. Die Administration, der Support und Schulungen von aDIS übernimmt das Bibliotheksservice-Zentrum Baden-Württemberg (BSZ), das in der Universität Konstanz ansässig ist. Das Zentrum für Datenverarbeitung der Universität Tübingen betreibt hingegen die Server-

Systeme[Lan17].

Mittlerweile sind zwei Folgeprojekte entstanden, die unabhängig voneinander jeweils ein Resource Discovery System (RDS) auf der Grundlage von VuFind entwickeln. Dabei bildet ein RDS mit aDIS und dem BSZ eine Dreiecksbeziehung. Das BSZ wird für die Aggregation der Datensätze genutzt, aDis weiterhin als integriertes Bibliothekssystem, über das der Ausleih- und Fernleihservice abgewickelt wird, und das RDS als Einstieg in die Titelsuche.

BSZ One Stop Search (BOSS) Das BOSS-Projekt versucht die angebotenen Leistungen einer Bibliothek unter den Deckmantel einer Benutzeroberfläche zu vereinen und zu vereinfachen. Normalerweise ist der Zugriff auf den Volltext, Ausleihsystem und Fernleihfunktion durch unterschiedliche Softwaresysteme realisiert zu denen ein Nutzer weitergeleitet wird. [BWb].

Das vom BSZ in Konstanz entwickelte System basiert auf VuFind in der Version 2.5. Die Nutzung von Vufind in der Hauptversion 3 ist zur Zeit in Planung.

BOSS bietet Schnittstellen zu kommerziellen Katalogen wie EBSCO, Ex Liebris oder ProQuest, aber auch andere Datenbanken können eingebunden werden.

BOSS wird aktuell von 22 Bibliotheken verwendet [BWc], wobei 30 weitere Bibliotheken das System testen [BWa]. Darunter fallen folgende Institutionen [Lan17]:

- Südwestdeutscher Bibliotheksverbund (SWB)
- HTWG Konstanz
- WLB Stuttgart
- HS Ulm
- PH Heidelberg

KatalogPlus Das zweite Projekt wird an der Universität Freiburg entwickelt und basiert auf VuFind in der Version 1.3. Der Umstieg auf eine aktuelle Version von VuFind wird gerade realisiert, ist jedoch durch die großen Anpassungen sehr aufwändig [Bor17]. Für den Suchindex werden Daten vom SWB abgezogen und durch einen Vorverarbeitungsschritt bereinigt und angereichert. Dieser basiert auf einer SQL-Datenbank, in die alle Datensätze eingespielt und durch Datenbank-Operationen manipuliert werden. Das Einspielen von 5,5 Millionen Datensätzen in die Datenbank dauert etwa vier Stunden, die Manipulation benötigen etwa 10 Stunden, und das Ausspielen und Zusammensetzen der Datensätze benötigt weitere vier Stunden[1]. Dieser Ansatz erreicht somit einen Datendurchsatz von 80 Datensätzen pro Sekunde, wobei fast 45% der Zeit auf das Ein- und Ausspielen der Daten verwendet wird.

Zum Aktualisieren des Indexes müssen die 5,5 Millionen Datensätze nicht erneut eingespielt werden. Es müssen nur die geänderten Datensätze des Teilabzuges zur Datenbank hinzugefügt werden, manipuliert und wieder ausgespielt werden.

Zusätzlich können bestehende Indizes über einen Proxy eingebunden und Bestandsnachweise und Verfügbarkeiten durch die DAIA-Schnittstelle von bestehenden aDIS abgefragt werden. Zusätzlich können kommerzielle Kataloge wie EBSCO, Summon und Swets eingebunden werden. Als Link-Resolver wird die Lösung von ReDI eingesetzt, darunter:

Aktuell wird KatalogPlus von sieben Einrichtungen genutzt:

- Badische Landesbibliothek
- UB Tübingen
- UB Ulm
- UB Stuttgart
- UB Freiburg

[1] Wiedergabe eines Gesprächs mit Hannah Born auf dem VuFind-Anwendertreffen 2016

4.4 IxTheo

An Hand des Projektes IxTheo, das den Index Theologicus der Theologischen Fakultät Tübingen auf einen modernen technischen Stand bringt, werden nun infolge einige Strukturen und Vorgänge beschrieben, die für den Entwicklung und den Betrieb eines solchen Systems notwendig sind. Wobei die genaue Umsetzung je nach Projektziele, Vorgaben, Umfang und Voraussetzungen unterschiedlich stark ausgeprägt sind.

4.4.1 Was ist IxTheo

Der Index Theologicus ist eine Sammlung von theologischer und religionswissenschaftlicher Literatur, die seit 1973 in regelmäßigen Intervallen veröffentlicht wurde. Anfangs wurde der Index abgedruckt und als Heft monatlich publiziert. Ab dem Jahr 1994 wurde der Index Theologicus in eine Datenbank überführt, die die analoge Publikation zum Jahr 2001 ablöste. Die Datenbank war auf CDs erhältlich und über das Internet abrufbar.

Mittlerweile umfasst diese Sammlung über 1,6 Millionen Titel. Es werden Festschriften und Kongressschriften sowie über 2000 Zeitschriften ausgewertet.

Das öffentlich zugängliche IxTheo-Portal ist zwar in der Lage diese Datenmenge zu verarbeiten, jedoch soll diese Plattform auf den aktuellen Stand der Technik gebracht werden.

4.4.2 Struktur

Für den Betrieb von einer Suchmaschine wie IxTheo gibt es vier Teilarbeitsbereiche:

Politik Ein Gremium ist nötig, um politische aber auch inhaltliche Fragen zu klären. Im Falle des IxTheos besteht dieses Gremium aus nur wenigen Personen.

Intellektuelle Erfassung Die Bereitstellung und Pflege der Datensätze bedarf eines geschulten Personals, das in der Lage ist Inhalte intellektuell zu Erfassen und

langfristig Verantwortung über die Daten zu übernehmen. Dabei müssen Änderung bei der Katalogisierung an ein Entwickler-Team weitergeleitet werden, um möglicherweise notwendige Anpassungen bei der Indizierung vornehmen zu können.

Entwicklung Ein Team aus Softwareentwicklern ist nötig um die technischen Grundlagen zuschaffen, aber auch um langfristig Softwarefehler zu beheben, neue Features umzusetzen und inkonsistente Daten, die bei der Indizierung entdeckt wurden, an das Team für die intellektuelle Erfassung zu melden.

Hardware Für einen Internetauftritt wird eine passende Infrastruktur benötigt. Im Bereich der Universität Tübingen ist das Zentrum für Datenverarbeitung (ZDV) dafür zuständig die nötige Hardware bereitzustellen und den reibungslosen Betrieb zu gewährleisten.

4.4.3 Datenlieferung

Die Datenquelle des IxTheos ist das Bibliotheksservice-Zentrum Baden-Württemberg (BSZ). Das BSZ aggregiert eine große Menge an bibliographischen Daten, die durch verschiedene Institutionen zusammengetragen und aktualisiert werden. Um eine fachspezifische Sammlung wie den IxTheo mit Daten zu versorgen werden Filter eingesetzt, die eine Untermenge definieren.

Es gibt zwei Möglichkeiten, wie diese Untermenge vom BSZ an den IxTheo geliefert wird:

- Durch einen Vollabzug werden alle Datensätze, die den Filtern genügen, zu einem Datenarchiv zusammen gefasst und an den IxTheo ausgespielt. Diese Art von Abzug ist vollständig, jedoch mit Kosten verbunden.

- Mit einem Teilabzug werden nur die Datensätze ausgeliefert, die seit dem letzten Voll- oder Teilabzug geändert wurden. Zusätzlich werden auch Löschlisten

bereit gestellt, in denen die Identifikationsnummer von gelöschten Datensätzen aufgeführt werden. Durch Teilabzüge fallen keine Kosten an, jedoch können in seltenen Fällen beim Abzugsprozess Datensätze durch das Raster fallen.

Dem IxTheo werden täglich Teilabzüge bereitgestellt, die mit einem älteren Vollabzug zu einem neuen 'Pseudovollabzug' zusammengeführt werden. Halbjährlich oder nach Absprache mit der Leitung des IxTheos werden neue Vollabzüge beantragt. Diese Vollabzüge sind in längeren, regelmäßigen Abständen von Nöten, da zwar theoretisch nach der Zusammenführung eines Vollabzug und allen folgenden Teilabzügen die Datenmenge identisch zu einem neuen Vollabzug sein sollte, jedoch in der Praxis dies nicht zuverlässig der Fall ist. Einzelne Datensätze können verloren gehen, da die Erstellung eines Teilabzugs einige Zeit benötigt und gleichzeitig eingespielte Änderungen teilweise nicht berücksichtigt werden können.

Die vom BSZ ausgelieferten Daten werden im MARC-21-Format ausgespielt. Innerhalb eines MARC-21-Datensatzes werden dabei sogenannte Lokaldatensätze angehängt. Diese enthalten Zusatzinformationen einzelner Institutionen. Dadurch können einige Datensätze nicht ausgespielt werden, da sie die vom MARC-21-Standard vorgeschriebenen Längenbeschränkung nicht einhalten. Aktuell können so von 1,6 Millionen Datensätzen etwa 120 Datensätze im IxTheo nicht gefunden werden.

4.4.4 Datenaufbereitung

Der IxTheo verfügt über eine hochperformante, sogenannte MARC-Pipeline. Dies ist eine Sammlung von Programmen, welche die vom BSZ ausgespielten Daten aufbereiten und durch zusätzliche Informationen anreichern, um alle Features von IxTheo zu betreiben.

Diese Pipeline umfasst etwa 20 verschiedene Programme, die jeweils als eigene Phase in der Pipeline angesehen werden und modular in beliebiger Reihenfolge ausgeführt werden können. Jedes Programm liest dabei eine MARC-21-Datei und schreibt die

geänderten Daten in eine neue MARC-21-Datei, die dann wiederum von dem nächsten Programm in der Pipeline gelesen wird. Im Schnitt dauert eine Phase etwa eine Minute, so dass ein Programm einen Durchsatz von etwa 25.000 Datensätzen pro Sekunde erreicht. Da es etwa 20 Pipeline-Phasen gibt benötigt die komplette Pipeline etwa 20 Minuten, wodurch ein Durchsatz von ca. 1.300 Datensätzen pro Sekunde für den vollständigen Durchlauf der Pipeline entsteht.

5 Diskussion

Im Rahmen dieser Arbeit habe ich die Struktur einer Suchmaschine für bibliographische Daten auf der Grundlage von VuFind ausgeführt. Außerdem habe ich durch die Analyse von SolrMarc einen Teilbereich des täglichen Umgangs mit VuFind genauer betrachtet und durch das Erarbeiten von Vorschlägen zur Verbesserung dieses Teilbereichs beigetragen.

Während der Arbeit an diesem doch recht speziellen Thema, dass im Detail sehr Umfangreich ist, kamen einige Fragen auf, die den Gesamtrahmen dieses Dokumentes übersteigen. Daher wird dieses Kapitel einerseits einen Rückblick auf die Arbeit, aber auch einen Ausblick für weiterführende Forschungen geben und einige Fragen ansprechen.

VuFind Ein wichtiger Faktor für das erfolgreiche Arbeiten mit VuFind ist die Community die sich inzwischen um VuFind gebildet hat. Dadurch, dass viele Softwareentwickler, die an einem Projekt auf der Basis von VuFind arbeiten, bereit sind ihr Wissen zu teilen und sogar an manchen Aufgaben gemeinsam zu arbeiten, ist VuFind ein lebendiges Projekt, dass sich weiter entwickelt und gleichzeitig die Bedürfnisse der einzelnen Bibliotheken berücksichtigt. Dabei sind die finanziellen Möglichkeiten des VuFind-Projekts eher überschaubar. Zwar gibt es Mitarbeiter der University of Villanova, USA, welche die Hauptverantwortlichen von VuFind sind und deshalb ein Etat zur Verfügung haben. Dieser Etat ist jedoch nicht vergleichbar mit den Möglichkeiten von kommerziellen Produkten, dennoch weist VuFind ein stetiges Wachstum auf.

Auf technischer Ebene ist es bemerkenswert, wie stark VuFind sich in den Jahren gewandelt und weiterentwickelt hat. Es ist nicht auf einer jahrzehntealten, aber bewehrten Technologie stehen geblieben, sondern wird ständig erneuert und auf einen neuen Stand gebracht. Eine Frage, die wären dieser Arbeit aufgekommen ist, betrifft das `fullrecord`-Feld, dass beim Indizieren mit dem vollständigen MARC-Datensatz gefüllt wird, wobei nicht zwischen MARC-21 und MARC-XML unterschieden wird. Dieser Datensatz wird bei der Volltitelanzeige genutzt um Informationen über den Datensatz anzeigen zu können. Jedoch ist fraglich, warum diese Informationen nicht in unterschiedliche Felder des Solr-Indexes indiziert werden, anstatt den Datensatz für die Anzeige erneut zu parsen. Da es einige Felder gibt, die für den korrekten Betrieb von VuFind vorausgesetzt werden, könnte man weitere Felder zu dieser Liste hinzunehmen.

Zend Framework Das `Zend Framework` zeigt bei VuFind seine Stärken aber auch seine Schwächen. Für Entwickler, die erfahren sind mit dem `Zend Framework`, ist der Einstieg in die Weiterentwicklung von VuFind sehr einfach. Doch durch die vielen Module und deren jeweilige Klassen, sowie Spezialerweiterungen von einzelnen Klassen ist es schwer, den Überblick über die Funktionalität dieser Bibliothek zu behalten. Dadurch wird die Lernkurve für neue Entwickler sehr steil und der Einstieg in das `Zend Framework` deutlich erschwert. Das zeigt sich nicht nur am Umfang des Frameworks, sondern auch an Designentscheidungen. So scheinen beispielsweise `AwareInterface`s zuerst magisch, da die Arbeitsweise nicht offensichtlich ist und ein umfangreiches Verständnis über das `Zend Framework` voraussetzt. Jedoch wird durch diesen großen Umfang an vordefinierten Funktionen und die effiziente und flexible Implementierung des gesamten Frameworks die Entwicklung von großen Applikationen und Webseiten ermöglicht.

Dank der großen Community, die über die Jahre um das `Zend Framework` entstanden ist, gibt es viele umfangreiche Dokumentationen und Tutorials in den verschie-

densten Sprachen zu fast allen Funktionen. Außerdem gibt es auch einige Foren und Mailing-Listen, in denen Fragen hilfsbereit beantwortet werden.

Durch die modulare Struktur und den `Modul-` sowie `Service Manager` lassen sich neue Funktionalitäten recht einfach zu einer bestehenden Zend-Applikation hinzufügen. Bei Projekten, die auf einem `Zend Framework` basierten System wie VuFind aufbauen, zeigen sich weitere Vorteile: VuFind wird in einem Git-Repository verwaltet. Um eigene Anpassungen vornehmen zu können kann das VuFind-Repository dupliziert werden. Nun kann in dem Duplikat ein Modul angelegt werden, das die nötigen Erweiterungen bereitstellt. Die Module von VuFind müssen dabei nicht manipuliert werden. Dadurch können neue Änderungen aus dem original VuFind-Repository mit geringem Aufwand in das Duplikat eingespielt werden.

SolrMarc Bei der Überarbeitung von SolrMarc wurde die Historie dieses Nebenprojektes ersichtlich. Durch die anfängliche Lösung, die durch ein unflexibles Skripts umgesetzt wurde, dass feste Pfade und Datenfelder voraussetzte, lag die Aufmerksamkeit der Entwicklung fast vollständig auf VuFind. Selbst als aus diesem Skript ein eigenständiges Projekt wurde fehlte es an einer kompetenten und technisch versierten Weiterentwicklung, wie sie bei VuFind zu finden ist.

Die in der Analyse prognostizierte Verdoppelung des Durchsatz wurde in dieser Arbeit weit übertroffen. Außerdem konnte gezeigt werden, dass durch moderne Konzepte und die Einführung von einfachen Strukturen die Flexibilität im Umgang mit SolrMarc deutlich verbessert werden konnte.

Jedoch bleiben noch einige Punkte offen, die in dieser Arbeit nicht behandelt wurden und weitere Performance-Gewinne versprechen. So werden die MARC-21-Datensätze durch die Bibliothek `marc4j` eingelesen. Das Parsen der gelesenen Daten ist dabei ineffizient gelöst. Durch die Verwendung von `ByteArrayInputStream`s müssen viele teure Allokationen durchgeführt werden, die durch eine spezialisierte Klasse nicht notwendig sind. Erste Ansätze zeigten eine Steigerung der Lesegeschwindigkeit von

20%. Zusätzlich ist der Zugriff auf einzelne Felder und Unterfelder eines Datensatzes mit jeweils eine Laufzeit in $O(n)$ umgesetzt, wobei n die Anzahl von Feldern des Datensatzes oder die Anzahl von Unterfeldern eines Feldes ist. Da dies nur einen kleinen Teil der Laufzeit von SolrMarc ausgemacht hat, wurden diese Anpassungen nicht in dieser Arbeit berücksichtigt. Die einzelnen Methoden zum Extrahieren von Daten aus MARC-21-Datensätzen wurden von SolrMarc direkt übernommen und nicht überarbeitet. Eine Analyse der einzelnen Methoden könnte somit eine weitere Steigerung des Durchsatzes bedeuten. Auch eine Parallelisierung und somit das Ausnutzen von Mehrkernprozessoren ist ein wichtiger Ansatz für weitere Optimierungen.

Betriebskonzepte Für das Kapitel über die Betriebskonzepte habe ich einige Bibliotheken in Deutschland angeschrieben um Informationen über deren Organisationsstrukturren zu bekommen. Die Anzahl an Antworten wahren eher dürftig um ein ausreichend genaues Bild der Situationen an deutschen Bibliotheken wiederzugeben. Interessant sind die doch sehr Ähnlichen Ansätze in der Struktur bei der Zusammenarbeit von mehreren Bibliotheken. So sind meist demokratische Entscheidungen die Grundlage für die Zusammenarbeit. Ebenso interessant scheint die technische Herangehensweise beim Aufbau des Indexes. Das Einspielen der Daten in eine Datenbank zur Aufbereitung scheint viele Möglichkeiten zu eröffnen. Jedoch ist die Verarbeitung durch die MARC-Pipeline sehr viel effizienter. Dadurch unterscheiden sich auch die Update-Strategien: die Datenbank-Lösung spielt nur Teilabzüge ein, die Pipeline-Lösung indiziert immer den kompletten Abzug.

Nach meiner Meinung sollte auf einem der nächsten VuFind-Anwendertreffen genau dieser Punkt in einem Workshop besprochen werden um neue Ideen zu sammeln und bestehende Lösung zu evaluieren.

6 Fazit

Der Erfolg bei der Überarbeitung von SolrMarc ist offensichtlich. Eine Steigerung des Durchsatzes um den Faktor 5,8 hat großen Einfluss auf die Arbeit mit VuFind. Dadurch sinken die Betriebskosten und die Hardwareanforderungen, ohne das Betreiber einer VuFind-Instanz größere Anpassungen vornehmen müssen. Das Teilen der Vorschläge zur Verbesserung von SolrMarc mit der VuFind Community führte zusätzlich zu einem Umdenken und zu einer großen Überarbeitung von SolrMarc, bei der die Ergebnisse dieser Arbeit mit Umgesetzt wurden. Die so entstandene neue Version von SolrMarc führte bei der University of Villanova, USA, sogar zu einer Performance-Steigerung um den Faktor 17,7.

Im Bereich der Betriebskonzepte scheinen noch weitere Untersuchungen von Nöten zu sein um ein klareres Bild zu bekommen. Der erste Einblick zeigt jedoch, dass bekannte Strukturen aus dem Bereich des Projektmanagements sich durchsetzen und zu einem langfristigen funktionierenden Betrieb eines auf VuFind basierende bibliographische Suchmaschine führen.

Literaturverzeichnis

[AMK08] AMK1211. File:BlankMap-World8.svg. https://commons.wikimedia.org/wiki/File:BlankMap-World8.svg, 2008. [Online; letzter Zugriff 16.06.2016].

[Avr75] Henriette D. Avram. Marc; its history and implications, 1975.

[Bib] Hessische Bibliotheksinformationssystem. Aufbau eines Discovery Systems für den HeBIS-Verbund. https://www.hebis.de/de/1ueber_uns/projekte/portal2.php. [Online; letzter Zugriff 16.01.2017].

[Bor17] Hannah Born. Fragen zum Freiburger VuFind. E-Mail, 2017. [Online; Siehe beiliegendem Datenträger].

[BWa] Bibliotheksservice-Zentrum Baden-Württemberg. Bibliotheken mit BOSS. https://wiki.bsz-bw.de/doku.php?id=projekte:boss:bossbibliotheken. [Online; letzter Zugriff 16.01.2017].

[BWb] Bibliotheksservice-Zentrum Baden-Württemberg. BOSS - Konzept. https://wiki.bsz-bw.de/doku.php?id=projekte:boss:bosskonzept. [Online; letzter Zugriff 16.01.2017].

[BWc] Bibliotheksservice-Zentrum Baden-Württemberg. BSZ One Stop Search (BOSS). https://wiki.bsz-bw.de/doku.php?id=projekte:boss:start. [Online; letzter Zugriff 16.01.2017].

[BYGH+08] Ori Ben-Yitzhak, Nadav Golbandi, Nadav Har'El, Ronny Lempel, Andreas Neumann, Shila Ofek-Koifman, Dafna Sheinwald, Eugene Shekita, Benjamin Sznajder, and Sivan Yogev. Beyond basic faceted search. In *Proceedings of the 2008 International Conference on Web Search and Data Mining*, pages 33–44. ACM, 2008.

[Dav03] Martin Davis, Mark und Dürst. Unicode normalization forms. `http://www.unicode.org/reports/tr15/tr15-23.html`, 2003. [Online; letzter Zugriff 12.09.2016].

[ddB] Börsenverein des deutschen Buchhandels. Titelproduktion insgesamt. `http://www.boersenverein.de/sixcms/media.php/976/Titelproduktion_Erst_und_Neuauflage_final.pdf`. [Online; letzter Zugriff 16.01.2017].

[fin] finc. finc - Mitglieder. `https://finc.info/de/mitglieder`. [Online; letzter Zugriff 16.01.2017].

[Kat12] Demian Katz. Solr Setup. `https://vufind.org/wiki/indexing:hierarchies\_and\_collections\#solr\_setup`, 2012. [Online; letzter Zugriff 27.06.2016].

[Kat16] Demian Katz. format.bsh. `https://github.com/solrmarc/solrmarc/blob/release-2.10/examples/GenericVuFind/index_scripts/format.bsh`, 2016. [Online; letzter Zugriff 12.09.2016].

[KN12] Demian Katz and Andrew Nagy. Vufind: Solr power in the library. *Library Automation and OPAC 2.0: Information Access and Services in the 2.0 Landscape: Information Access and Services in the 2.0 Landscape*, page 73, 2012.

[Lan17] René Lange. Informationen zu VuFind im ZDV. E-Mail, 2017. [Online; Siehe beiliegendem Datenträger].

[LVdS01] Carl Lagoze and Herbert Van de Sompel. The open archives initiative: Building a low-barrier interoperability framework. In *Proceedings of the 1st ACM/IEEE-CS Joint Conference on Digital Libraries*, JCDL '01, pages 54–62, New York, NY, USA, 2001. ACM.

[Mus14] Björn Muschall. Personalisierung von Suchräumen in RDS. `https://www.slideshare.net/BjrnMuschall/suchrume-und-personalisierung-dbt2014?ref=https://finc.info/de/Archive/6`, 2014. [Online; letzter Zugriff 16.01.2017].

[Niea] Pat Niemeyer. BeanShell - Simple Java Scripting . `http://beanshell.org/manual/bshmanual.html`. [Online; letzter Zugriff 16.09.2016].

[Nieb] Pat Niemeyer. BeanShell auf GitHub.com. `https://github.com/beanshell/beanshell`. [Online; letzter Zugriff 16.09.2016].

[Obe16] Oliver Obenland. JavaValueExtractorUtils.java. `https://github.com/oobenland/SolrMarc-Indexer-Tests/blob/master/src/playground/solrmarc/index/extractor/impl/java/JavaValueExtractorUtils.java`, 2016. [Online; letzter Zugriff 16.06.2016].

[oC06a] Library of Congress. 007 - Physical Description Fixed Field-General Information. `https://www.loc.gov/marc/bibliographic/bd007.html`, 2006. [Online; letzter Zugriff 06.09.2016].

[oC06b] Library of Congress. 008 - Fixed-Length Data Elements-General Information. `https://www.loc.gov/marc/bibliographic/bd008.html`, 2006. [Online; letzter Zugriff 06.09.2016].

[oC06c] Library of Congress. Introduction. http://www.loc.gov/marc/bibliographic/bdintro.html, 2006. [Online; letzter Zugriff 12.06.2016].

[Reh17] Uwe Reh. Fragen zu HeBIS. E-Mail, 2017. [Online; Siehe beiliegendem Datenträger].

[Sch11] Jürgen Schmidt. Ten years of the Lucene search engine at Apache. http://www.h-online.com/open/news/item/Ten-years-of-the-Lucene-search-engine-at-Apache-1350761.html, 2011. [Online; letzter Zugriff 28.02.2017].

[Sei12] Leander Seige. Der aggregierte Index – eine Entscheidung. https://finc.info/de/Archive/27, 2012. [Online; letzter Zugriff 16.01.2017].

[sol08a] Class CommonsHttpSolrServer. https://lucene.apache.org/solr/3_6_0/org/apache/solr/client/solrj/impl/CommonsHttpSolrServer.html, 2008. [Online; letzter Zugriff 11.09.2016].

[sol08b] Class StreamingUpdateSolrServer. https://lucene.apache.org/solr/3_6_0/org/apache/solr/client/solrj/impl/StreamingUpdateSolrServer.html, 2008. [Online; letzter Zugriff 11.09.2016].

[Tar13a] Cassandra Targett. Overview of Searching in Solr. https://cwiki.apache.org/confluence/display/solr/Overview+of+Searching+in+Solr, 2013. [Online; letzter Zugriff 28.02.2017].

[Tar13b] Cassandra Targett. RequestHandlers and SearchComponents in SolrConfig. https://cwiki.apache.org/confluence/display/solr/RequestHandlers+and+SearchComponents+in+SolrConfig, 2013. [Online; letzter Zugriff 17.07.2016].

[Tar13c] Cassandra Targett. RequestHandlers and SearchComponents in SolrConfig. https://cwiki.apache.org/confluence/display/solr/RequestHandlers+and+SearchComponents+in+SolrConfig, 2013. [Online; letzter Zugriff 17.07.2016].

[Tar13d] Cassandra Targett. Response Writers. https://cwiki.apache.org/confluence/display/solr/Response+Writers, 2013. [Online; letzter Zugriff 17.07.2016].

[Tar13e] Cassandra Targett. Response Writers. https://cwiki.apache.org/confluence/display/solr/Response+Writers#ResponseWriters-PHPResponseWriterandPHPSerializedResponseWriter, 2013. [Online; letzter Zugriff 28.02.2017].

[VC16] VuFind-Community. VuFind Customer Installations. https://vufind.org/wiki/community:installations, 2016. [Online; letzter Zugriff 16.06.2016].

[Wei11] Evelyn Weiser. finc - Projektstart. https://finc.info/de/Archive/32, 2011. [Online; letzter Zugriff 16.01.2017].