Kurzfassung

Suchtechnologien werden heutzutage in allen Bereichen des Internets eingesetzt. Hierbei liegt der Schwerpunkt in der Verarbeitung von semi- oder unstrukturiert vorliegender Information, auf welcher eine Volltextsuche ausgeführt wird. Auf diese Weise werden dem Nutzer einer solchen Technologie in kurzen Zeiten die Informationen zur Verfügung gestellt, welche für ihn relevant sind.

Im Bereich von Electronic Commerce ist die Suchfunktion eine nicht mehr zu ersetzende Grundfunktion, welche zur Ausstattung jener modernen Online-Shops gehört. Der Grund hierfür liegt in der Leichtigkeit, die der Shop-Nutzer durch eine gut funktionierende Suche, mit der er auf schnellstem Weg zu gewünschten Produkten geleitet wird, erfährt. Gerade deswegen ist es für den Shop-Betreiber umso wichtiger eine voll funktionstüchtige Suchfunktion bereitzustellen, um konkurrenzfähig zu bleiben.

Um eine komplette Suchfunktionalität selbst aufzubauen, bedarf es viel Einsatz an Ressourcen. Das kann mit dem Einsatz einer Komplettlösung einer Suchplattform zur externen Anbindung an einen bestehenden Shop vermieden werden. Zwei sehr erfolgreiche Vertreter dieser Branche sind FACT-Finder und Solr, welche auf dem Gebiet der Suche in Online-Shops sehr breite Anwendung finden, sodass an einem Vergleich dieser Produkte ein öffentliches Interesse besteht.

Im Rahmen der vorliegenden Arbeit werden FACT-Finder und Solr zunächst anhand gestellter Anforderungen aus Sicht des Nutzers und Shop-Betreibers verglichen. Anschließend werden die Suchplattformen exemplarisch in den Blog von SCOOBOX, das eine Erweiterung der E-Commerce-Software Enfinity Suite 6 ist, der Firma dotSource GmbH integriert und bezüglich des Integrationsaufwandes und der Performance gegenübergestellt.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Dieses Kapitel erläutert die Motivation und die Zielsetzung der vorliegenden Arbeit. Weiterhin wird dem Leser ein Überblick über den Inhalt der einzelnen Kapitel dieser Diplomarbeit gegeben.

1.1 Einordnung und Motivation der Arbeit

Das Online-Geschäft in Deutschland verzeichnet im Jahr 2010 einen Umsatz von 30,3 Milliarden Euro, was im Vergleich zum Vorjahr eine Steigerung um 4,1 Prozent ist. Der Anteil am gesamten Umsatzes in der Handelsbranche liegt mittlerweile bei 60,4 Prozent, wobei für das Jahr 2011 weiteres Wachstum von 15,5 Prozent vorausgesagt wird. [BV10]

Das positive Wachstumsverhalten dieser Branche kann zum Teil durch die einkaufsfreudigen Kunden, aber auch durch die Bequemlichkeit, die der Kunde bei einem Online-Einkauf erfährt, erklärt werden. Aus diesem Grund wollen viele Shop-Betreiber in den Bereich E-Commerce übergehen, um eigene Umsatzzahlen durch ein breiteres Spektrum an Kunden zu erhöhen. Als logische Konsequenz des Erfolges dieser Branche resultiert zugleich die hohe Konkurrenz zwischen den, in hoher Anzahl vertretenen, Onlineshop-Betreibern. Um diesem Konkurrenzdruck standzuhalten, müssen die Shop-Betreiber u. a. hoch qualitative Services anbieten.

Eine der zentralen Servicekomponenten ist die Produktsuche, welche durch die Integration moderner Web 2.0-Technologien auf weitere Komponenten eines Onlineshops erweitert werden muss. Repräsentative Beispiele hierfür sind Einträge in Foren, Blogs sowie Kommentare und Kundenbewertungen.

Die vorrangigen Ziele der Suche in einem Onlineshop sind die Unterstützung der Suche auf allen Komponenten sowie effizientes Bereitstellen und eine korrekte und übersichtliche Darstellung der Suchergebnisse. Auf diesem Gebiet existieren bereits Anbieter, welche ausgereifte Lösungen in Form von externen Suchplattformen für die Integration in bestehende Systeme bereitstellen. Die bekanntesten Vertreter der E-Commerce-Branche sind FACT-Finder der Firma Omikron Data Quality GmbH und Solr von Apache Software Foundation. Hierbei ist es interessant die Anforderungen an die externen Suchplattformen aus dem Blickwinkel der Shoppingportale zu untersuchen, um die Suchmaschinen strukturiert zu vergleichen.

1.2 Zielsetzung

Die Suchplattformen FACT-Finder und Solr werden schrittweise gemäß der folgender Zielsetzung verglichen.

1. Die Anforderungen seitens der Shoppingportale an die externe Suchplattformen sollen möglichst genau erfasst und exakt definiert werden.
2. Anhand der aufgestellten Anforderungen sollen FACT-Finder und Solr gegenübergestellt werden, um eine objektive Bewertung dieser Suchplattformen abzugeben.
3. Der theoretische Vergleich soll mittels der Ergebnisse aus der praktischen Integration in das Blog-System von SCOOBOX der Firma dotSource GmbH überprüft und gefestigt werden. Auf der Grundlage der technischen Integration wird die Indizierung sowie Qualität und Geschwindigkeit der Suche gemessen und ausgewertet.

1.3 Inhaltlicher Überblick

Das Kapitel 2 erläutert die Grundlagen für die Hauptkapitel 4 und 5 dieser Arbeit. Dabei werden die theoretischen Bausteine wie Information Retrieval, E- und Social-Commerce sowie Suchalgorithmen und Indizierungstechniken für die Volltextsuche betrachtet. Für den praktischen Teil der Arbeit wird die eingesetzte Entwicklungsumgebung sowie das Softwareprodukt, für das die Integration der Suchplattformen vorgesehen ist, vorgestellt.

Im darauf folgenden Kapitel werden die Anforderungen an externe Suchplattformen im Bezug auf die E-Commerce-Branche gestellt. In diesem Zusammenhang werden die Anforderungen aus der Sicht des Betreibers und aus dem Blickwinkel des Nutzers eines Online-Shops betrachtet.

Das Kapitel 4 stellt die Suchplattformen FACT-Finder und Solr vor. Diese werden im zweiten Schritt miteinander verglichen und bewertet. Die Gegenüberstellung basiert auf Anforderungen, welche im Kapitel 3 formuliert wurden.

Die Integration von FACT-Finder und Solr in das Blog-System von SCOOBOX der Firma dotSource GmbH wird im 5 Kapitel beschrieben. Hierbei werden die vorhandenen Implementierungen und für die Integration notwendige Erweiterungen sowie die Ergebnisse der praktischen Umsetzung erfasst. Anschließend werden die integrierten Suchplattformen auf Basis eines exemplarischen Anwendungsfalls bezüglich des Integrationsaufwandes und der Performance miteinander verglichen.

Im letzten Kapitel werden die Ergebnisse in Bezug auf die Zielsetzung dieser Arbeit zusammengefasst und ein Ausblick über die weiterführenden Arbeiten gegeben.

Kapitel 2

Grundlagen

Das Grundlagenkapitel ist in zwei Abschnitte unterteilt. Im ersten Teil werden die Details von Information Retrieval und der damit im Zusammenhang stehenden Prozesse erläutert. Der zweite Abschnitt enthält einen informativen Überblick zu E-Commerce und den Softwareprodukten, welche für die Implementierung im Rahmen dieser Arbeit eingesetzt werden.

2.1 Information Retrieval

Information Retrieval (IR) ist ein Teilbereich der Computerwissenschaften, deren Übersetzung ins Deutsche für Informationsbereitstellung, -beschaffung oder auch weiter gefasst für die Informationssuche steht. Die technische Bestimmung von IR wird in [BYRN11] wie folgt definiert.

„*Information retrieval deals with the representation, storage, organization of, and access to information items such as documents, Web pages, online catalogs, structured and semi-structured records, multimedia objects. The representation and organization of the information items should be such as to provide the users with easy access to information of their interest.*“

IR ist also ein komplexer Prozess, welcher die *Präsentation*, *Speicherung*, *Organisation* und den *Zugriff* auf Daten beinhaltet. Die Daten, die in diesem Zusammenhang verarbeitet werden, sind nicht an eine bestimmte Art gebunden. So können beispielsweise Texte in Form von Dokumenten oder Textpassagen, Bildern bzw. Bildausschnitten, Tönen wie Musikstücke und allgemeine Audiospuren sowie Filmmaterial als Informationsquellen dienen. Das primäre Ziel von IR ist die einfache und schnelle Bereitstellung von Information, die den Nutzer interessiert. Damit stellt Information Retrieval die Grundlage für Suchmaschinen aller Art dar, was die Betrachtung von IR im Hinblick auf den Vergleich von *FACT-Finder* und *Solr* im Kapitel 4 begründet.

Die Funktionen zur *Präsentation*, *Speicherung*, *Organisation* von Daten, sowie der *Zugriff* darauf wird von den Datenbankmanagementsystemen (DBMS) ebenso unterstützt. Im Gegensatz zu IR werden in DBMS andere Zielstellungen verfolgt. Die Schwerpunkte werden beispielsweise auf die Bereiche der Datensicherung, Ausfallsicherheit, Transaktionsverwaltung, Hochverfügbarkeit u. a. gelegt.

Anhand der Information Retrieval Dimensionen, welche in [Rij79] ausführlich diskutiert werden, wird der Unterschied zwischen IR und DBMS deutlich. Die Zusammenfassung einzelner Eigenschaften ist in der Tabelle 2.1 dargestellt. Im Folgenden werden die Dimensionen gemäß [Fuh11] erläutert.

Eigenschaft	**DBMS**	**IR**
Matching	exakt	partiell, best match
Inferenz	Deduktion	Induktion
Datenorganisation	strukturiert	semi- bzw. unstrukturiert
Anfragesprache	formal	natürlich
Fragespezifikation	vollständig	unvollständig
gesuchte Objekte	erfüllen die Fragespezifikation	relevante
Reaktion auf Datenfehler	sensitiv	insensitiv

Tabelle 2.1: IR-Dimensionen nach Rijsbergen aus [Fuh11]

Matching: Bei DBMS wird die Suchanfrage exakt in der Form gesucht, wie sie formuliert wurde (*exact match*). In IR sollen auch unsichere Treffer gefunden werden, welche nur teilweise auf die Anfrage zutreffen.

Inferenz: DBMS berechnet die Suchergebnisse anhand fest definierter Regeln, sodass die Schlussfolgerung nach deduktiver Methode vom allgemeinen auf speziellen Fall erfolgt. IR erschließt die Ergebnisse eher induktiv, in dem auf Basis von Beobachtungen und Erfahrungen einzelner Fälle die allgemeine Erkenntnisse gewonnen werden.

Datenorganisation: Die Kriterien für die Erzeugung eines Datenbankschemas erzwingen eine eindeutige Struktur der Daten. Bei IR existiert ein grobes Schema mit ungenauer Abgrenzung, sodass nicht alle Objekte alle Attribute haben müssen, sodass die Daten semi- bzw. nicht strukturiert sind.

Anfragesprache: DBMS verstehen nur formale Sprachen wie z. B. *Structured Query Language (SQL)*. In IR können die Begriffe aus der natürlichen Sprache verwendet werden.

Fragespezifikation: DBMS-Anfragen müssen syntaktisch korrekt sein, um der Sprache, die vollständig spezifiziert ist, zu genügen. In IR werden die Anfragen fast immer unvollständig formuliert. Oft weiß der Nutzer selbst nicht genau wonach er sucht, sodass die Anfrage sukzessive modifiziert wird.

gesuchte Objekte: In DBMS werden Objekte mit der exakten Übereinstimmung gesucht. Bei IR werden *relevante* Ergebnisse, die in Bezug zu der gesuchten Anfrage stehen, erwartet.

Reaktion auf Datenfehler: Fehler im Datenbestand können bei DBMS zu Problemen bei der Wiederfindung der Datensätze führen. IR ist an dieser Stelle durch die *Matching*-Eigenschaft fehlertoleranter als ein DBMS.

In den folgenden Betrachtungen wird die Form von Daten bzw. Dokumenten auf die Textform eingeschränkt. Dies sei damit begründet, dass die Diskussion weiterer oder sogar

aller Datenvariationen den Rahmen dieser Arbeit sprengen würde und die ausschließliche Betrachtung der textuellen Form für weitere Untersuchungen ausreichend ist.

2.2 Information Retrieval System

Im praktischen Umfeld wird Information Retrieval mithilfe eines *Information Retrieval Systems (IRS)* realisiert. IRS ist ein computergestütztes System, welches die Information speichert, organisiert, repräsentiert und gegen eine Nutzeranfrage automatisiert abruft. In der Abbildung 2.1 ist ein IRS schematisch abgebildet. Die Komponenten und ihre Funktionsweise werden im Folgenden gemäß [BYRN11] beschrieben.

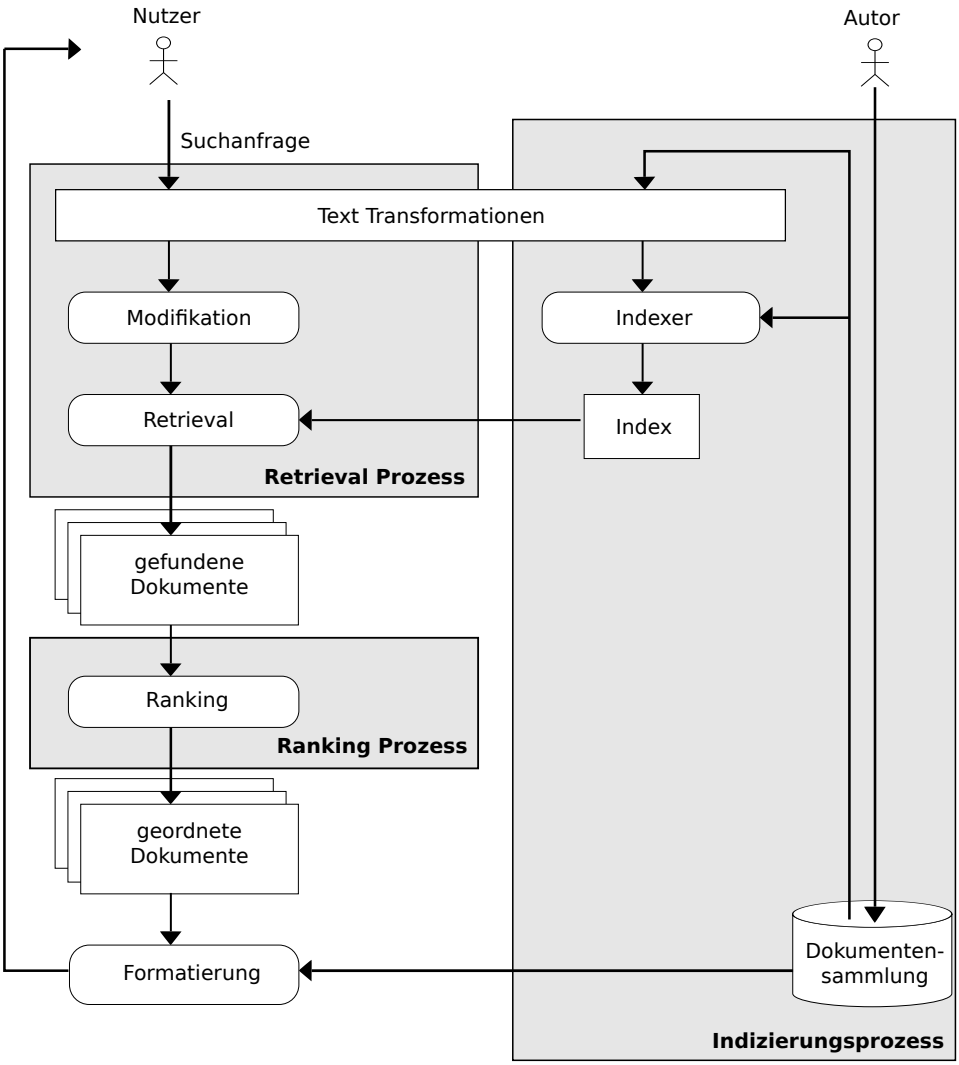

Abbildung 2.1: IRS-Architektur: Komponenten und Prozessablauf aus [BYRN11]

Die Architektur eines IRS besteht im Wesentlichen aus drei Hauptkomponenten, welche für den *Indizierungs-*, *Retrieval* und *Ranking* Prozess verantwortlich sind. Grundlegend für jedes IRS ist die *Dokumentensammlung*, welche z. B. ein *Autor* aktualisiert und pflegt. Wenn die Informationen direkt aus dem Web geholt werden, so wird ein *Web-Crawler*,

welcher die Webseiten analysiert und so die Informationen bereitstellt, vor die *Dokumentensammlung* geschaltet [Lew05]. Die *Dokumentensammlung* ist auf einem Speicher abgelegt, welches ein DBMS oder ein einfaches File-System sein kann. Um die *Retrieval-* und *Ranking-* Prozesse effizient zu gestalten, muss die *Dokumentensammlung* als ein *Index* vorliegen. Der *Index* wird aus der *Dokumentensammlung* durch den *Indexer* entweder direkt oder nach *Texttransformationen*, welche im Abschnitt 2.2.1 vorgestellt werden, aufgebaut. Der *Index* wird oft als eine *invertierte Liste*, deren Beschreibung im Abschnitt 2.2.2 stattfindet, implementiert. Der *Indizierungsprozess* ist aufgrund der Indexerstellung aus i. d. R. großen Dokumentensammlungen ein ressourcenintensiver Prozess, welcher im Offline-Modus durchgeführt werden sollte.

Nachdem der Indizierungsprozess vollzogen ist, kann der *Retrieval Prozess* ausgeführt werden. Initial stell der *Nutzer* eine *Anfrage* an das IRS, welche zunächst mithilfe von *Texttransformationen* für die Weiterverarbeitung aufbereitet wird. Im nächsten Schritt findet die *Modifikation* statt, wobei die *Anfrage* z. B. mit anderen Wortformen oder den Systemvorschlägen, welche bei der Eingabe gebracht wurden, erweitert wird. Die modifizierte *Anfrage* und der *Index* werden als Parameter in die *Retrieval-Funktion* der *Retrieval* Komponente eingegeben. Als Rückgabelwert wird eine Liste von Tupeln, die aus dem Dokument und zugehörigem *Retrieval Status Value (RSV)* bestehen, generiert [Fuh11]. RSV ist das Maß für die *Relevanz*, welche die Beziehung zwischen dem Dokument und der Anfrage beschreibt. Nach dem *Retrieval Prozess* werden die Dokumente mithilfe der *Ranking* Komponente anhand der berechneten *Relevanz* geordnet.

Anschließend werden *geordnete Dokumente* einer *Formatierung* unterworfen und dem *Nutzer* präsentiert. Bei der *Formatierung* besteht der Zugriff auf die *Dokumentensammlung*, sodass die Suchergebnisse darauf abgebildet und beispielsweise mit hervorgehobenen Fundstellen präsentiert werden.

Bei den *Retrieval-* und *Ranking-* Komponenten findet die eigentliche Suche und Anordnung der Dokumente statt. Diese Komponenten sind grundlegend für den eigentlichen Suchprozess und sind durch das verwendete *Information Retrieval Modell (IRM)* weitestgehend definiert. IRM beschreibt weiterhin wie die Repräsentation der *Dokumentensammlung* und der *Anfrage* auszusehen hat und ist somit fundamental für die Implementierung eines IRS. Ein Überblick über die wichtigsten Modelle des Information Retrieval werden im Abschnitt 2.2.3 gegeben.

Aus dem Grund der Übersichtlichkeit wurde in der Abbildung 2.1 die Rolle des Administrators nicht dargestellt. Im praktischen Umfeld konfiguriert dieser die Indizierungs- sowie Retrieval- und Ranking-Komponenten für den konkreten Anwendungsfall.

2.2.1 Texttransformationen

IR weist bei der Wissensrepräsentation der in Volltext vorliegenden Dokumentensammlung ein zentrales Problem auf. Die im Text vorkommenden Worte können im Text auf unterschiedlichste Weise formuliert werden, was in der Literatur als *Synonymie* und *Polysemie* bekannt ist.

- *Synonymie* ist die Ähnlichkeit der Bedeutung von Ausdrücken, welche sprachlich verschieden sind. Beispielsweise sind *„Notebook“* und *„Laptop“* synonym.

- *Polysemie* ist das Gegenstück zu *Synonymie* und steht für die Mehrdeutigkeit eines Ausdrucks. Ein repräsentatives Beispiel stellt das Wort *„Bank“* dar. Dieses Wort kann ohne weiteren Kontext beispielsweise als Sitzmöbel oder als Kreditinstitut interpretiert werden.

Es muss also dafür gesorgt werden, dass verschiedene Formulierungen auf die gleiche Repräsentation abbilden und unklare Formulierungen identifiziert werden. [Fuh11]

Die im Folgenden vorgestellten Texttransformationen bieten Lösungsansätze für das oben beschriebene Problem.

In Abhängigkeit von der gegebenen Dokumentensammlung ist u. U. eine Vorverarbeitung wie die Bestimmung der Kodierung, des Dateiformates sowie der Sprache notwendig. Im nächsten Schritt der *Tokenisierung* wird der Volltext in Worte zerlegt und von Ziffern, Bindestrichen, Satzzeichen und *Stoppwörtern* befreit sowie die Groß-/Kleinschreibung vereinheitlicht. Anschließend werden die einzelnen Zeichenketten weiteren *Normalisierungs-* und *Filterprozessen* wie z. B. die *Stammwortreduzierung* sowie Anwendung der *Thesauren* und *phonetischer Analyse* unterworfen. Das Resultat der vorgenommenen Texttransformationen ist eine Ansammlung von einzelnen Zeichenketten, welche als *Tokens* bzw. *Terme* bezeichnet werden und in dieser Form in den Index aufgenommen werden. [Got10]

Stoppwortliste
Die Stoppwortliste enthält die Stoppwörter, diese wird bei der Texttransformation geladen und auf den Volltext angewendet. Stoppwörter kommen i. d. R. in allen Dokumenten vor und haben demzufolge nur sehr geringen Identifikationspotenzial für den Retrieval Prozess, weil sie in relevanten Dokumenten genauso oft vorkommen können wie in nicht relevanten. Die Stoppwörter tragen also einen geringen auszeichnenden Charakter und bringen aus diesem Grund keinen bedeutenden Informationsgewinn für das IRS. [Sto07]

Die typischen Vertreter von Stoppwörtern sind Artikel oder Präpositionen. Diese sind sprachspezifisch und sollten der Übersichtlichkeit wegen separiert gespeichert und gepflegt werden. Ein Beispiel ist die Zeichenkette *„die“*, welche in deutsche, aber keinesfalls eine englische Stoppwortliste gehört. [Fer03]

In gewissen Anwendungsfällen kann es jedoch sinnvoll sein die Stoppwörter mit in die Suche einzubeziehen. Ein Beispiel ist das berühmte Zitat aus Hamlet *„Sein oder nicht Sein“*. Intuitiv bieten sich die Zeichenketten *„oder“* und *„nicht“* für die Aufnahme in die Stoppwortliste an, was in diesem konkreten Beispiel die Tragödie von William Shakespeare mit hoher Wahrscheinlichkeit als nicht relevant für das Zitat zuordnen würde. Dieses Beispiel zeigt, dass die Stoppwörter in einem gewissen Kontext nicht vollkommen informationsleer sind. So ist es denkbar die Stoppwörter in Abhängigkeit von der Thematik der Dokumentensammlung mit in den Index aufzunehmen und bei dem Retrieval Prozess mit einer gewissen Abwertung einzubeziehen. [Sto07]

Stoppwörter bieten keine direkte Lösung für das zur Einleitung dieses Abschnittes beschriebene Problem der Wissensrepräsentation an. Sie dienen vielmehr als ein Teil der Vorverarbeitungsschritte vor den eigentlichen Filteroperationen. Der positive Effekt neben der Beseitigung nicht auszeichnender Tokens ist ein nicht unnötig aufgeblähter Umfang des Indexes, was einen effizienteren Retrieval Prozess zur Folge hat.

Stammwortreduzierung
Die Stammwortreduzierung (engl. *stemming*) ist ein Vertreter der *Freitextsuche*, bei der

keine weitere Repräsentation erstellt und die Funktion zur Verbesserung der Suche auf die Tokens angewendet wird. [Fuh11]

Durch die Stammwortreduzierung werden die Wortformen auf ihren Wortstamm zurückgeführt. Hierbei handelt es sich um den computerlinguistischen Ansatz, bei dem die Terme in anderen Flexionsformen erkannt werden. Der reduzierte Wortstamm kann von der linguistisch gültigen Form abweichen. [Fer03]

Zu den wichtigsten Lösungsansätzen für die Stammwortreduzierung zählen *Präfix-/Suffix-Entfernung*, *Table Lookup* und die *N-Gram-Methode*. [Lew05]

Bei der ersten Variante werden die Suffixe von der Wortform getrennt. Das klappt in englischer Sprache für die regelmäßige Pluralbildung ganz gut. Für die deutsche Sprache eignet sich dieses Verfahren jedoch nicht, weil die Abwandlung im Vergleich zu Englisch komplexer ist. Die Trennung von Präfixen führt dagegen in gewissen Fällen zu falschen Ergebnissen, weil dabei die Bedeutung des Wortes verloren gehen kann. Beispiele von sinntragenden Präfixen in der deutschen Sprache sind *„anti-“* und *„un-“*. [Sto07]

Table Lookup ist ein wörterbuchbasiertes Verfahren, bei dem alle Abwandlungen eines Wortstamms in einer Liste gespeichert werden. Dieses Verfahren hat sich in der Praxis sehr gut bewährt, jedoch ist der Pflegeaufwand relativ hoch und schwer automatisiert umzusetzen. [Lew05]

Bei der *N-Gram-Methode* werden die Terme in Bestandteile, genannt *Grame*, der Länge *N* zerlegt. Auf diese Weise werden zusammengesetzte Wörter automatisch in ihre Teilworte zurückgeführt. Manchmal kann es jedoch bei der Zerlegung zu einem Teilwort reduziert werden, welches dem Sinn des Ursprungswortes nicht entspricht. Das Beispiel hierfür ist das Wort *Widerspruchsfreiheitsbeweis*, welches bei der Zerlegung in Grame der Länge fünf u. a. *Reihe* als Teilwort enthält, das inhaltlich nicht dem Ausgangswort entspricht. [Lew05, Sto07]

Im Gegensatz zu *N-Gram-Methode* sind die Lösungsansätze *Präfix-/Suffix-Entfernung* und *Table Lookup* sprachspezifisch und dementsprechend aufwändiger für mehrsprachige Systeme zu implementieren, weil für jede Sprache eigene Umsetzung erforderlich ist. [Lew05]

Die Stammwortreduzierung verkleinert den Umfang des Indexes, führt allerdings dazu, dass bei der Suche nach Flexionsformen eines Begriffs das Suchergebnis automatisch auf alle Flexionsformen der zugrundeliegenden Stammform erweitert wird. Dadurch wird es unmöglich nach spezielleren Begriffen zu suchen. [Fer03]

Thesaurus

Im Gegensatz zu der Stammwortreduzierung handelt es sich bei dieser Methode um einen Vertreter von *Dokumentationssprachen*, die den semantischen Ansatz für die Lösung des Problems der Wissensrepräsentation verfolgen. Mit *Dokumentationssprachen* werden mithilfe syntaktischer und symantischer Regeln *Deskriptoren*, welche die Tokens aus dem Text repräsentieren, definiert. [Fuh11]

Ein *Thesaurus* definiert ein kontrolliertes Vokabular, welches die Wörter und deren Beziehungen untereinander erfasst, um ein Wissensgebiet eindeutig zu beschreiben. Jedes Wissensgebiet besteht aus Synonymen und bildet formal eine eigenständige *Äquivalenzklasse*. Jede *Äquivalenzklasse* wird durch ein Vorzugselement, genannt *Deskriptor*, repräsentiert. Mithilfe von Beziehungen werden die synonyme Begriffe durch Äquivalenzrelation und hierarchische Ober- und Unterbegriffsrelationen sowie assoziative Relationen zwischen den Deskriptoren definiert. [Fer03]

In IRS wird nur der *Deskriptor* in den Index aufgenommen, über den der Zugriff auf separat gespeicherte Synonyme im Retrieval Prozess erfolgt. [Got10]

Beispielsweise kann eine Äquivalenzklasse zwischen den Begriffen *„Obstbaum“*, *„Apfelbaum“* und *„Apfelsinenbaum“* im Thesaurus mithilfe der *IS-Relation*, als die Beziehung zwischen zwei Synonymen, dargestellt werden. Als *Deskriptor* kann z. B. der Begriff *„Obstbaum“* verwendet werden, welcher als solches durch die *USE-Relation* kenntlich gemacht wird. Für die Umsetzung sind somit folgende Befehle notwendig.

1. *Apfelbaum IS Obstbaum*; *Apfelsinenbaum IS Obstbaum*
2. *Apfelbaum USE Obstbaum*; *Apfelsinenbaum USE Obstbaum*

Bei den Relationen *IS* und *USE* handelt es sich nicht um eine genormte Sprache zur Definition von Thesauren, diese dienen lediglich zur Veranschaulichung des Beispiels.

Die Konstruktion von Thesauren kann entweder manuell oder automatisiert erfolgen. Bei der manuellen Konstruktion der Thesauren wird gemäß [Fer03] wie folgt vorgegangen.

1. Der Umfang, die thematische Eingrenzung und Definition der Syntax des Thesaurus werden vorgenommen.
2. Dokumente, Textsammlungen oder auch bereits fertige Thesauren werden als Quellen für die Konstruktion ausgewählt.
3. Die Terme des Thesaurus werden der terminologischen Kontrolle unterworfen, um die nicht eindeutigen Relationen der natürlichen Sprache zu beseitigen. Man überprüft das Vokabular mithilfe der Synonym- und Polysemkontrollen.
4. Abschließend wird die begriffliche Kontrolle des Thesaurus vorgenommen. Dabei werden die Deskriptoren in hierarchische und assoziative Relationen gesetzt.

Da die manuelle Konstruktion relativ aufwendig, teuer und langsam ist, wurden automatisierte Verfahren für die Thesaurus-Konstruktion entwickelt. Diese schließen aus dem gemeinsamen Auftreten auf die Ähnlichkeit der Terme in der Dokumentensammlung. Obwohl diese Verfahren schneller und kostengünstiger als die manuelle Konstruktion ist, werden in den meisten Fällen keine strenge Hierarchien in der Struktur solcher Thesauren erreicht. Als Folge können fälschlicherweise relevante Dokumente aus dem Suchergebnis ausgeschlossen und nicht relevante Dokumente in das Ergebnis aufgenommen werden. Damit ist die Qualität der automatisch konstruierten Thesauren erwartungsgemäß schlechter als solcher, die manuell erstellt wurden. [Fer03]

Phonetische Analyse

Die *phonetische Analyse* bildet einen weiteren Lösungsansatz für das Problem der Wissensrepräsentation. Die Zielstellung ist es Rechtschreibfehler mithilfe der Analyse des phonetischen Klanges aufzuspüren.

Bei der phonetischen Suche werden die Zeichenketten auf die Ähnlichkeit der Aussprache hin untersucht. Diese Art von Vergleich wird auch als *Phonetic Matching* bezeichnet und nimmt seinen Anfang im Jahr 1917. Damals entwickelte und patentierte Robert C. Russell ein Verfahren für die phonetische Sortierung eines Namensregisters. Dieses Verfahren wurde später als *Soundex* bezeichnet. [Sto07]

Soundex generiert für die gegebene Zeichenkette eine Zahlenkombination, welche den phonetischen Klang des Wortes repräsentiert. Bei diesem Algorithmus werden alle Vokale gleich gewichtet, sodass der Klang von Konsonanten als das Maß für die Ähnlichkeit angenommen wird. Die Buchstaben werden in Gruppen zusammengefasst, indem ähnlich klingende Konsonanten derselben Gruppe zugeordnet werden. [Hof10]

Code	*Buchstaben*
0	a, e, i, o, u, y, h, w
1	b, p, f, v
2	c, g, j, k, q, s, x, z
3	d, t
4	l
5	m, n
6	r

Tabelle 2.2: Regelsatz von Soundex aus [Hof10]

Bei dem Ablauf wird der erste Buchstabe ohne Änderung übernommen, alle weiteren Buchstaben werden nach dem Ersetzungsschema aus der Tabelle 2.2 mit Beachtung folgender Regel kodiert. [Sto07]

1. Ignoriere Groß- und Kleinschreibung.
2. Wenn ein Buchstabe in einem Wort mehrmals nacheinander auftritt, wird nur das erste Auftreten kodiert und alle weiteren Vorkommen ignoriert.
3. Entferne alle Nullen aus der Kodierung.
4. Gib die ersten vier kodierten Zeichen zurück, fülle gegebenenfalls mit Nullen auf.

Zur Veranschaulichung der Arbeitsweise von *Soundex* werden im Folgenden zwei Beispiele (vgl. Tabellen 2.3 und 2.4) angegeben.

Regel	*„Meier“*	*„Meyer“*
1	Meier	Meyer
2	M06	M06
3	M6	M6
4	M600	M600

Tabelle 2.3: positives Beispiel

Regel	*„Spears“*	*„Superzicke“*
1	Spears	Superzicke
2	S1062	S01062020
3	S162	S1622
4	S162	S162

Tabelle 2.4: negatives Beispiel

Das Ersetzungsschema in der Tabelle 2.2 und das Beispiel in der Tabelle 2.4 mit den Begriffen *„Spears“* und *„Superzicke“* machen deutlich, dass der Soundex in seiner Ursprungsform für den englischsprachigen Raum gedacht war und nicht ohne weitere Modifikationen für alle Sprachen anwendbar ist. Man müsste beispielsweise für die Anwendung in der deutschen Sprache die Umlaute und *ß* hinzufügen. Eine mögliche Spezialisierung von Soundex für die deutsche Sprache wurde im Jahr 1969 von H. J. Postel unter den Namen *„Kölner*

Phonetik“ vorgeschlagen. Der angepasste Algorithmus nutzt ein anderes Ersetzungsschema und beachtet die umliegenden Buchstaben. [Hof10]

Da *Soundex* den ersten Buchstaben unverändert übernimmt, können nur Fehler der darauffolgenden Buchstaben korrigiert werden. Der weitere Nachteil von auf *Soundex* basierten Verfahren ist, dass ausschließlich eine Aussage, ob Ähnlichkeit gegeben ist oder nicht, geliefert wird. Das hat zur Folge, dass die Ergebnisse nicht nach der Ähnlichkeit angeordnet werden können. Aus diesem Grund ist die Kombination mit anderen Verfahren im praktischen Umfeld die Regel. [Sto07]

2.2.2 Invertierte Liste

Nach der Vorverarbeitung der Dokumentensammlung mithilfe von Texttransformationen liegt die dabei extrahierte Information, in Form von Tupeln vor. Jedes Tupel ist aus dem Token und der Information in welchem Dokument es vorkommt, wie z. B. eine *Dokumenten-ID*, aufgebaut. Für die Überprüfung der Relevanz der Dokumente müssen die Terme der Anfrage mit den der einzelnen Dokumente verglichen werden. Wenn Übereinstimmungen mit einer der Listen festgestellt werden, so wird das entsprechende Dokument in das Suchergebnis aufgenommen. Ohne weitere Strukturierung der Tupel ist bei jeder Suchanfrage ein Vergleich zwischen jedem Token der Suchanfrage mit jedem Token der Dokumentensammlung erforderlich. Bei dieser naiven Vorgehensweise ist der Retrieval Prozess aufgrund fehlender Organisation der Tupel sehr ineffizient.

Invertierte Listen bzw. *invertierte Indizes* sind in IRS eine der meist eingesetzten Datenstrukturen, welche die Tupel für den Retrieval Prozess geeignet organisieren und dadurch die Suche auf der Dokumentensammlung effizient umsetzen. Die invertierte Liste besteht aus alphabetisch sortierten Termen der Dokumentensammlung. Der effiziente Zugriff auf die einzelnen Tokens wird über einen Baum oder eine Hash-Tabelle realisiert. Jeder Eintrag in der *invertierten Liste* enthält eine *Posting Liste*, welche auf alle Dokumente, die den jeweiligen Token enthalten, verweist. Die Dokumente innerhalb der *Posting Liste* sind aufsteigend sortiert. [Got10]

Das schrittweise Vorgehen beim Aufbau einer invertierten Liste wird beispielhaft in der Abbildung 2.2 gezeigt, wobei die Wortzerlegung bereits erfolgt ist.

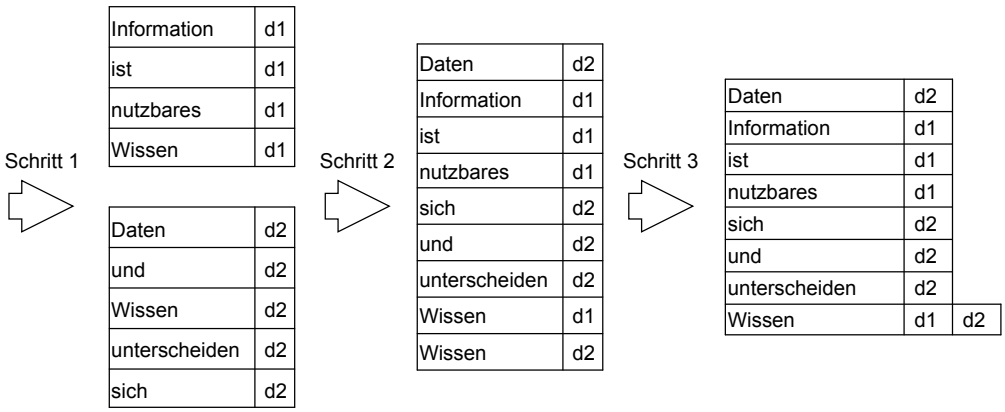

Abbildung 2.2: Beispielaufbau einer invertierten Liste in Anlehnung an [Got10]

Die Ausgangssituation ist in der Abbildung 2.2 nicht dargestellt, wobei folgende Dokumente als Text vorliegen.

Dokument 1 (d1): Information ist nutzbares Wissen

Dokument 2 (d2): Daten und Wissen unterscheiden sich

Nach dem ersten Schritt der Wortzerlegung sind die Dokumente als Tupeln aus Term und Dokument-ID repräsentiert. Im zweiten Schritt werden alle Tupel vermischt und alphabetisch geordnet. Anschließend werden die gleichen Terme zusammengefasst. Auf diese Weise kann bei der Suche effizient auf die Dokumente, welche den jeweiligen Term enthalten, zugegriffen werden. Durch die Zusammenfassung der Terme werden die Dokumente, die gleiche Terme enthalten, strukturiert verwaltet und können somit in einem Schritt über die jeweilige Posting Liste abgerufen werden.

Ein Nachteil von invertierten Listen ist, dass bei einer Aktualisierung, also Hinzunahme von neuen Termen, die komplette Liste neu sortiert werden muss.

2.2.3 Modelle des Information Retrieval

Ein *Information Retrieval Modell* wird nach [Fer03] durch die folgenden Eigenschaften charakterisiert.

1. Repräsentation D für die Dokumente der *Dokumentensammlung*
2. Repräsentation Q für die *Anfragen* an die Dokumentensammlung
3. *Retrieval-Funktion* $R : Q \times D \to \mathbb{R}$, welche dem Dokument für eine Anfrage eine reelle Zahl zuordnet

Im Folgenden wird ein Überblick über die wichtigsten Modelle des Information Retrieval gegeben und die jeweiligen Vor- und Nachteile aufgezählt.

Boolesches Modell
Das boolesche Modell ist unter Verfahren, die nach den Begriffen in exakter Form suchen (exact match), einzuordnen und basiert auf den Gesetzen, welche George Boole im Jahr 1854 aufgestellt hat [Sto07]. Das boolesche Model ist historisch gesehen das erste, das in Information Retrieval eingesetzt wurde [Fuh11].

Die Suchanfragen werden mithilfe von booleschen Operatoren *AND*, *OR* und *NOT* formuliert. Um komplexere Anfrage zu stellen wird weiterhin die Klammerung unterstützt. Die Retrieval-Funktion ordnet den Dokumenten die Werte 1 oder 0 zu. Das heißt ein Dokument ist entweder relevant oder nicht relevant zu der gestellten Suchanfrage, sodass das Ergebnis aus zwei disjunkten Teilmengen besteht. [Lew05]

Wenn man beispielsweise nach dem Begriff *Information* in den Beispieldokumenten aus dem Abschnitt 2.2.2 gesucht wird, dann wird die Retrieval-Funktion das Dokument mit der Dokumenten-ID *d1* als relevant auswerten, weil dieses den Suchbegriff in exakter Übereinstimmung enthält.

Das boolesche Modell hat gemäß [Lew05] und [Fuh11] mehrere Nachteile, die im Folgenden aufgezählt werden.

- Viele Dokumente werden nicht gefunden, weil die Trennung zwischen relevant und nicht relevant streng ist. So werden beispielsweise bei einer UND-Verknüpfung von drei Termen, diejenigen Dokumente mit zwei übereinstimmenden Termen genauso nicht in das Suchergebnis aufgenommen wie solche mit keinen Treffern.
- Das Ranking von Dokumenten ist nicht möglich. Damit kann es sehr schwierig werden bei einem umfangreichen Ergebnis relevante Dokumente zu finden, weil diese nicht zwangsläufig vorne stehen müssen.
- Die Formulierung der Suchanfragen ist für einen ungeübten Nutzer u. U. ein schwieriges Unterfangen.
- Es gibt keine Möglichkeit die Tokens innerhalb des Indexes und der Anfrage zu gewichten.

Fuzzy-Set-Modell

Das Fuzzy-Set-Modell basiert auf der Fuzzylogik, die von L. A. Zadeh im Jahr 1965 erfunden wurde [Dre08]. Die Fuzzylogik nutzt die Theorie der unscharfen Mengen (englisch: *Fuzzy Sets*), welche die traditionelle Mengenlehre um den Grad der Mitgliedschaft in einer Menge verallgemeinert. Der Grad der Mitgliedschaft liegt im abgeschlossenem Intervall zwischen 0 und 1 und kann mithilfe der Zugehörigkeitsfunktion[1] bestimmt werden. [Fer03]

Dieses Modell verfolgt genau wie das boolesche Modell den mengentheoretischen Ansatz für die Wissensrepräsentation. Jedes Dokument wird als eine unscharfe Menge, die aus Tupeln besteht, repräsentiert. Jedes Tupel besteht aus dem Token, der Dokumenten-ID und dem Grad seiner Zugehörigkeit, welche als eine reelle Zahl im Intervall $[0, 1]$ definiert ist, zu dem jeweiligen Dokument. Die Anfragen werden genauso wie bei dem booleschen Modell repräsentiert. Die Operatoren *AND*, *OR* und *NOT* sind im Vergleich zu dem booleschen Modell für die Auswertung der Gewichte erweitert und entsprechen den Definitionen des Durchschnitts, der Vereinigung und des Komplementes aus der Theorie der unscharfen Mengen. [Fuh11]

Für die Bestimmung der Werte einzelner Terme innerhalb des Indexes werden in [CP99] nur Vorschläge gemacht. Die Werte sollen entweder über eine manuelle Bewertung durch Experten oder automatisiert in Abhängigkeit von solchen Parametern wie der Typ des Dokumentes oder die Auftrittshäufigkeit des Terms innerhalb des Dokuments bestimmt werden. Ein konkrete Technik zur Bestimmung der Termwerte wird in [HCL03] beschrieben. Hier wird mithilfe der Termfrequenz und der inversen Dokumentenhäufigkeit die Relevanz von Dokumenten zu der Suchanfrage schrittweise, durch virtuelle Verschiebung der Suchanfrage in Richtung relevanter Dokumente, verbessert. In [Ver05] werden für diese Aufgabe drei Vorgehensweisen vorgestellt, welche die Werteerstellung direkt, indirekt und mithilfe von Zahlentransformationen umsetzen.

Bei diesem Ansatz wird also der Nachteil von nur exakten Übereinstimmungen des booleschen Modells mit der Unschärfe der Zugehörigkeit zu einer Menge behoben, so dass ein Ranking der Dokumente im Suchergebnis möglich ist. Das gilt aber nur für die Gewichtung der Dokumente. Die Formulierung von gewichteten Anfragen ist weiterhin nicht möglich. Der Nachteil mit der komplizierten Anfragesprache bleibt weiterhin bestehen.

In gewissen Fällen kann die Auswertung des UND-Operators in dem Fuzzy-Set-Modell zu fraglichen Ergebnissen führen. Wie bereits erwähnt, entspricht ein logischer UND-Operator

[1] bei der booleschen Logik wird die Zugehörigkeit ausschließlich auf die Intervallgrenzen abgebildet

dem Durchschnitt in der Mengentheorie, welches von zwei unscharfen Mengen als das punktweise Minimum ihrer Zugehörigkeitsfunktionen definiert ist [Fer03]. Das grenzwertige Verhalten wird an einem Beispiel aus [Fuh11] deutlich.

$$T = \{t_1, t_2\}$$

$$q = t_1 \wedge t_2$$

$$d1 = (0.6, 0.6),\ d2 = (0.59, 1.00)$$

$$q(d1) = 0.6,\ q(d2) = 0.59$$

Das Dokument *d2* ist im zweiten Term deutlich höher und nur um 0,1 weniger im ersten Term gewichtet. Im Gesamtergebnis wird aber das Dokument *d1* als relevanter gewertet. Durch dieses Beispiel wird deutlich, dass die Benutzung das Fuzzy-Set-Modells im praktischen Umfeld zumindest in der Standarddefinition problematisch sein kann.

Vektorraummodell

Das *Vektorraummodell (VRM)* wurde von Gerard Salton in den 1960er und 1970er Jahren entwickelt und am *System for the Mechanical Analysis and Retrieval of Text (S.M.A.R.T.)* erprobt. [Sto07]

Die primäre Zielstellung bei der Entwicklung des neuen Modells war die Nachteile, die durch die exakte Übereinstimmung im Retrieval Prozess des booleschen Modells entstanden, auszubessern. Die Grundidee war nicht mehr nach exakten Übereinstimmungen, sondern nach der *Ähnlichkeit* zwischen den Dokumenten und der Anfrage zu suchen. [Lew05]

Die gesamte Dokumentensammlung besteht aus einer Ansammlung von Termen, von den jeder eine Dimension eines mehrdimensionalen Vektorraums darstellt. Damit wird der Grundraum, auf dem die Dokumente und Anfragen operieren, aufgespannt. Die Dokumente und Anfragen werden als Vektoren, die aus gewichteten Termen bestehen, repräsentiert und werden als Punkte in dem Vektorraum aufgefasst. Das besondere bei dieser Repräsentation ist, dass es keine Formale Sprache für die Formulierung von Anfragen an ein solches IRS gibt. Die Suchanfrage können direkt als Freitext in das System eingegeben werden. [Lew05, Got10]

Als Retrieval-Funktion für die Berechnung der Ähnlichkeit von Dokumenten zu der Anfrage können verschiedene Maße aus der Vektorrechnung verwendet werden. Meistens wird der Kosinus des Winkels zwischen dem Dokumenten- und Anfragevektor berechnet. [Fer03, Fuh11]

Das VRM macht in der Theorie keine Angaben darüber, wie die Gewichte der Terme innerhalb der Dokumente und der Suchanfrage bestimmt werden sollen. Ein weitverbreitete Methode für die Gewichtung der Terme in VRM heißt *TF-IDF* und wird im Folgenden nach der Beschreibung in [Got10] vorgestellt.

Bei dieser Methode werden zwei Faktoren zu einem Term t berechnet und ins Verhältnis gesetzt. Der erste Faktor ist die *inverse Dokumentenfrequenz* $idf(t)$, welche die globale Verteilung des Terms wie folgt angibt:

$$idf(t) = \frac{N}{df(t)}$$

Dabei gibt N die Anzahl der Dokumente in der Dokumentensammlung und $df(t)$ die Anzahl der Dokumente an, in den Term t mindestens einmal vorkommt. Weil der Funktionswert der *inversen Dokumentfrequenz* gerade bei kleineren Werten von $df(t)$ starke

Veränderungen zeigt, wird auf die *inverse Dokumentfrequenz* der Logarithmus angewendet. Dieser globale Faktor wird größer, umso weniger der Term global verteilt ist. Das impliziert, dass es sich eher um einen beschreibenden spezielleren Term handelt. Das Gegenteil passiert, wenn der Term oft vorkommt, in diesem Fall findet eine Abwertung statt. Es handelt sich also um Terme, welche nicht auszeichnend für die in dem Dokument befindliche Information sind. Solche Terme bewegen sich in die Richtung der Stoppworte, welche im Abschnitt 2.2.1 beschrieben wurden.

Der zweite Faktor heißt *Termfrequenz* $tf(t)$ und gibt die Anzahl der Vorkommnisse des Terms in dem Dokument an. Je häufiger ein Term in einem Dokument vorkommt, umso wichtiger ist es für das Dokument.

Das Termgewicht w für den Term t nach der *TF-IDF*-Methode im VRM wird als Produkt aus der *Termfrequenz* mit der *inversen Dokumentenfrequenz* nach der folgenden Formel berechnet.

$$w_t = tf(t) * log\left(\frac{N}{df(t)}\right)$$

Das VRM ist benutzerfreundlich, weil es die natürliche Sprache direkt für die Suche verwenden lässt und keine Sprachelemente erfordert. Die Gewichtung der Terme erlaubt das Ranking nach der Relevanz der Ergebnisse und die Formulierung von gewichteten Anfragen. Neue Dokumentensammlungen können direkt nach dem Indizierungsprozess für den Retrieval Prozess eingesetzt werden. [Lew05, Fuh11]

Da keine Operatoren existieren, erfordert eine sinnvolle Anfrage relativ viele Suchbegriffe. Aus demselben Grund ist ein Ausschließen von Suchbegriffen nicht möglich, was in der booleschen Logik mit dem Komplement einfach realisiert ist. [Lew05]

Probabilistisches Modell

Bei dem probabilistischen Retrieval Modell wird die Fragestellung, wie relevant das Dokument für die Anfrage ist, beantwortet. Die Relevanz unter der Bedingung, dass die Anfrage und das Dokument vorliegen, kann mithilfe der Formel von Bayes berechnet werden. Dafür muss die Unabhängigkeit der Dokumente untereinander vorausgesetzt werden [Fer03]. Der Nutzer ordnet die Dokumente als relevant auf der Grundlage bereits angesehener Texte ein. Das probabilistische Modell berechnet stets die Relevanz, die der Algorithmus vorgibt, so dass der subjektive Einfluss durch die Unabhängigkeit nicht einbezogen werden kann [Sto07].

Die Umsetzung dieses Modells erfordert mindestens zwei Suchanfragen für das Generieren des Suchergebnisses. Der Algorithmus erfordert für die Berechnungen die Wahrscheinlichkeit der Relevanz unter der Bedingung, dass das Dokument vorliegt. Diese Wahrscheinlichkeit liegt zunächst nicht vor und kann erst im initialen Durchlauf ermittelt werden. Im zweiten Schritt kann nach dem Feedback des Nutzer oder automatisiert durch Pseudo-Feedback des System der zweite Durchsuchungsschritt gestartet werden. Der Nachteil dabei ist, dass der Nutzer, die für ihn relevanten Dokumente nicht notwendig in den Top-Ergebnissen findet. Bei dem Feedback durch das System geht die subjektive Relevanz verloren, was zu stark verfälschten Ergebnissen aus der Sicht des Nutzers führen kann. [Sto07]

In der Tabelle 2.5 sind die wichtigsten Kriterien der vorgestellten Modelle zusammenfassend dargestellt. Alle hier vorgestellten IRMs, bis auf das boolesche Modell, geben eine geordnete Dokumentenliste aus. Das bedeutet insbesondere, dass das Ergebnis eines solchen

	Boolesch	**Fuzzy**	**Vektor**	**Probabilistisch**
Boolesche Operatoren	ja	ja	nein	nein
Gewichtung	nein	ja	ja	ja
Ranking	nein	ja	ja	ja

Tabelle 2.5: Vergleich von IRM in Anlehnung an [Lew05]

IRS alle eingehenden Dokumente enthält, wobei die letztendlich an den Nutzer vermittelte Anzahl durch die Anzahl der Suchergebnisse bzw. Konfiguration der Relevanzgrenze manipuliert werden. Die mit der höheren Ähnlichkeit, Relevanz oder Wahrscheinlichkeit werden entsprechend weiter vorne eingeordnet. Dieses Konzept wird als Relevance Ranking bezeichnet, das historisch gesehen seinen Anfang in dem probabilistischen Modell findet [Sto07].

2.3 Electronic Commerce

Der elektronische Handel (Electronic Commerce oder E-Commerce) bezeichnet den Einkaufsvorgang, bei dem der Geschäftsprozess zwischen dem Warenanbieter und dem Kunden unter dem Einsatz elektronischer Kommunikationstechniken abgewickelt wird. Dabei umfasst der Geschäftsprozess jede Art von Transaktion, die im Rahmen von Leistungsanbahnung, -vereinbarung oder -erbringung ausgeführt wird. [Hau10]

Nach der Art der teilnehmenden Geschäftspartner kann E-Commerce genauer kategorisiert werden. Im Rahmen dieser Arbeit ist Electronic Commerce aus dem Bereich *Business-to-Consumer (B2C)* vorrangig von Interesse. Diese Art des elektronischen Handels umfasst die geschäftlichen Beziehungen zwischen einem Unternehmen und seinen Endkunden. [Peg01]

Die Plattform für den elektronischen Handel wird als *Online*, *Electronic* oder *E-Shop* bezeichnet. Der Kunde kann das Warenangebot des Onlineshops über eine Web-Adresse im Internet einsehen. Bei der Abwicklung des Einkaufsvorganges legt der Kunde die gewünschten Produkte zunächst in einen virtuellen Warenkorb. Im weiteren Verlauf wird der potentielle Käufer in den sogenannten *Checkout*-Prozess geleitet, bei dem die Rechnungs- und Versandadresse dem Shop-Betreiber mitgeteilt werden sowie eine der unterstützten Zahlungsarten des Shops ausgewählt. Nach erfolgreicher Vermittlung der notwendigen Informationen an den Produktanbieter erhält der Kunde die Zahlungsinformationen, womit der Bestellvorgang abgeschlossen wird. [Zoe01]

2.3.1 Enfinity Suite der Intershop AG

Ein bekannter Vertreter aus der Welt von E-Commerce ist Intershop AG. Das Unternehmen wurde im Jahr 1992 von Stephan Schambach, Karsten Schneider und Wilfried Beeck in Jena gegründet. Der Geschäftsbereich von Intershop ist die Entwicklung und der Vertrieb von Standardsoftware für Anwendungen im Bereich des elektronischen Handels. Weiterhin bietet Intershop Beratung- und Serviceleistungen, wie das Outsourcing der kompletten Prozesse. Zur Zeit beschäftigt Intershop ca. 430 Mitarbeiter und erzielte im Jahr 2010 einen Umsatz von 38,25 Millionen Euro. [Int11b]

Das Hauptprodukt von Intershop ist die E-Commerce-Software *Enfinity Suite*, die aktuell in der Versionsnummer 6 vertrieben wird. Enfinity Suite wurde im Jahre 1994 entwickelt und umfasst Komplettlösungen zur Abbildung verschiedener Geschäftsmodelle für den Vertrieb von Produkten über das Internet. [IES11]

Die praktische Ausarbeitung zu der vorliegenden Arbeit wurde mit dem Softwareprodukt Enfinity Suite 6 umgesetzt. Dieses Produkt stellt unterschiedliche Frameworks für E-Commerce-Lösungen bereit, von den im praktischen Teil der vorliegenden Arbeit die wichtigsten unmittelbar eingesetzt wurden. Aus diesem Grund werden gemäß [E6410] und [P6410] die wichtigsten Komponenten sowie die Programmierebenen, die für die Entwicklung unmittelbar genutzt wurden, vorgestellt.

Enfinity Suite 6 - Komponenten
Die Struktur von Enfinity Suite 6 ist durch Web-, Server- und Datenebene dreigeteilt. In den Ebenen sind Web-Server mit -Adapter, Applikation-Server, File-System und Oracle-Datenbankinstanz als Komponenten enthalten. Für die Skalierung der Architektur ist die Verwendung von mehreren Web- und Applikation-Servern möglich. In diesem Fall spricht man von einem Enfinity Cluster. Die Abbildung 2.3 veranschaulicht den Aufbau einzelner Ebenen, welche im Folgenden erläutert werden.

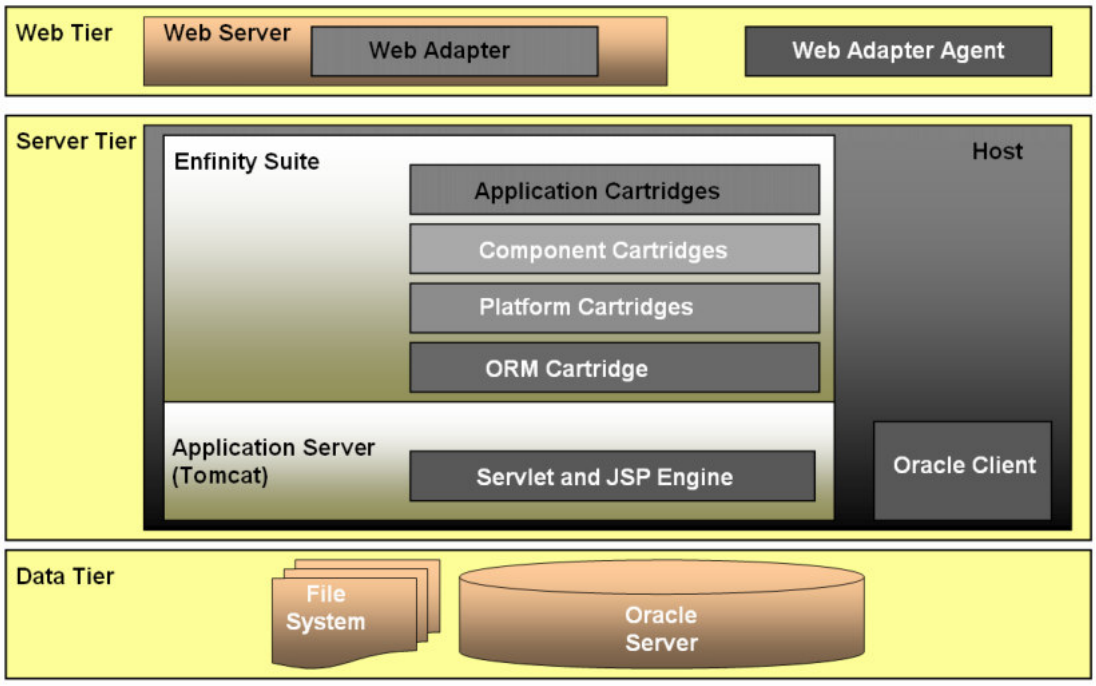

Abbildung 2.3: Komponenten von Enfinity Suite 6 aus [Int11a]

Die Bestandteile der *Web-Ebene (Web Tier)* sind Web-Server, Web-Adapter und der Web-Adapter-Agent. Web-Server ist ein Hypertext Transfer Protocol (HTTP)-Server von Apache, in den der Web-Adapter als ein Zusatzmodul von Enfinity integriert ist. Web-Adapter sorgt für die Verteilung der Client-Anfragen über die Applikation-Server, Generierung der Session-ID (je nach Konfiguration in der URL oder als Cookie parametrisiert) und Pufferung der Daten und Überprüfung bereits vorhandener Informationen (z. B. Formularinhalte, statische Seiteninhalte) der Applikation-Server

Der Web-Adapter-Agent ist ein Hintergrundprozess (*deamon process*), welcher für die Räumung des Puffers (invalide Seiten) und Versenden von Anfragen und Log-Dateien (mitunter die Statistiken der Verbindungsdaten) an den Applikation-Server, verantwortlich ist.

Die *Server-Ebene (Server Tier)* enthält einen Application Server (Apache Tomcat), einen Oracle Client und die Enfinty Applikation.

Apache Tomcat stellt die Ausführungsumgebung für die Enfinity Suite und eine Web-Applikation für die Konfiguration der Serverprozesse in einem Enfinity Cluster bereit. Enfinity Suite ist aus Cartridges aufgebaut. Eine Cartridge ist ein Software-Modul, das einen bestimmten Bereich an Funktionalität der Enfinity Suite abdeckt. ORM-Cartridge (Object-Relational-Mapping) umfasst die Funktionalität zur Verwaltung der persistenten Objekte. Dazu gehören die Abbildung zwischen Datenbankobjekten und Java-Klassen, Transaktionsverwaltung, ORM-Caching, Logging und Verwaltung der Datenbank-Java-Schnittstelle (JDBC). Plattform-Cartridges umfassen die technischen Grundfunktionen wie Cache-Verwaltung, Import-/Export-Funktionalität und Datenreplikation. Des Weiteren sind hier die praktischen Funktionen wie das Nutzer-Rechte-System oder Währungs- und Länderverwaltung implementiert. Component- und Application-Cartridges bitten anwendungsnähere Funktionalität, die auf den Plattform-Cartridges basieren. Der Umfang der Cartridges hängt von den Anforderungen und Zielen der Organisation ab, welche die Enfinity Suite zur Abbildung der Geschäftsstruktur und -prozesse einsetzt. Oracle Client bietet die grundlegende Funktionalität für die Import- und Export-Prozesse und Datenreplikation.

Daten-Ebene (Data Tier) umfasst den Oracle Server und das File-System. Die Oracle Datenbank speichert die Daten, welche den transaktionsbasierten Zugriff erfordern oder durchsuchbar sein müssen. In der Regel sind das alle Anwendungs- und ein Teil der Konfigurationsdaten.

Das File-System speichert Systemdaten wie Cartridges und Konfigurationsdateien sowie Seiten und deren statische Inhalte, die auf dem Applikation-Server laufen.

Enfinity Suite 6 - Programmierebenen
In der Abbildung 2.4 sind die Programmierebenen von Enfinity Suite dargestellt. Dieses Architekturmuster entspricht dem Model-View-Controller-Ansatz ([P6410], kurz: MVC), der in der Software-Entwicklung zunehmend eingesetzt wird.

Java-Schicht (Java Layer) besteht aus Business-Objekten, Manager-Klassen und den Pipelets. Business-Objekte sind Java-Klassen, welche persistente Daten aus der Datenbank abbilden. Manager-Klassen definieren den Zugriff auf die Business-Objekte. Aus der Sicht der Programmierung sind das Java-Interfaces, welche die eigentliche Implementierung für den Entwickler transparent gestalten und somit den direkten Zugriff auf die persistenten Objekte vermeiden. Pipelets sind atomaren Programmbausteinen. Das sind Java-Klassen, welche die grundlegenden Funktionen, wie z.B. Ermittlung des Produktpreises oder Auflösen der Nutzerdaten anhand des Logins und Passwortes, implementieren. Zu jedem Pipelet gehören weiterhin zwei Extensible Markup Language (XML)-Dokumente. Diese erfassen die ein- und ausgehenden Parameter für die Kommunikation mit den anderen Pipelets.

Pipeline-Schicht (Business Logic Layer) fast die Pipelets zu einem Programmfluss als eine Abfolge von atomaren Operationen zusammen. Ein Programmfluss kann z. B. einen Geschäftsprozess oder einen Hintergrundprozess des Verwaltungssystems abbilden. Die

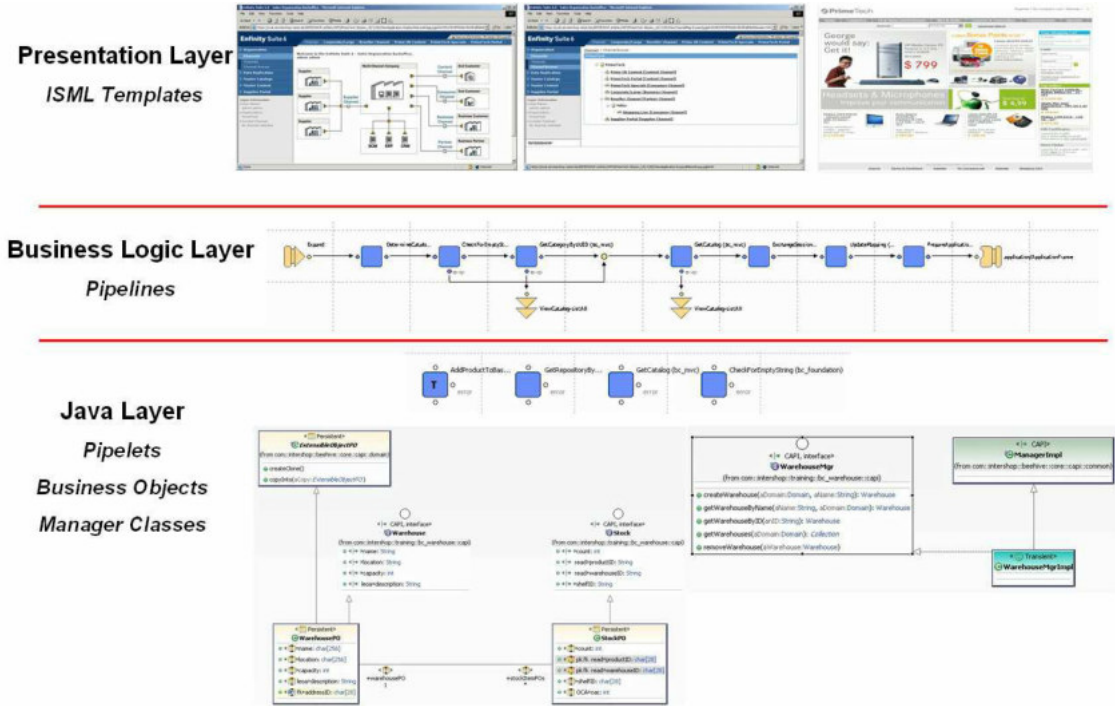

Abbildung 2.4: Programmierebenen von Enfinity Suite aus [Int11a]

Elemente zur Programmsteuerung, wie Abzweigung oder Schleifen, werden neben den Pipelets für den Aufbau des Programmsteuerungsflusses in den Pipelines eingesetzt. Bei jeder Anfrage an den Server wird intern eine Pipeline, die in der URL kodiert ist, getriggert und damit ein Prozess zur Behandlung der Anfrage gestartet. Je nach Operation, die mit der Pipeline abgearbeitet wird, unterscheidet man zwischen Prozess-, Job- und Präsentation-Pipelines. Die Prozess- und Job-Pipelines führen gekapselte Prozesse aus, wobei die Job-Pipelines einen Hintergrundprozess spezifizieren. Die Präsentation-Pipelines bereiten dagegen Business-Objekte zur Präsentation auf der Nutzeroberfläche auf.

Präsentationsschicht (Presentation Layer) konvertiert die Ergebnisse der Pipeline-Schicht für die Darstellung auf der Nutzeroberfläche und sendet das Resultat in Form fertigen HyperText Markup Language (HTML)-Seiten an den Client. Das Erstellen der Templates wird mithilfe von *Intershop Markup Language (ISML)*, eine Erweiterung von HTML, realisiert.

2.3.2 SCOOBOX der Firma dotSource GmbH

Im Januar 2006 wurde dotSource GmbH von C. Grötsch und C. Malik gegründet und war im Geschäftsbereich der Entwicklung von IT-Mittelstandlösungen sowie E-Commerce- und Java-Beratung in transaktionsbasierten Web-Anwendungen vertreten. Nach drei Jahren wurde der Geschäftsbereich um das Hosting und den Betrieb von auf Magento[2] basierten E-Commerce-Lösungen erweitert. Im Jahr 2010 beschäftigte dotSource GmbH etwa 60 Mitarbeiter und konnte ihren Umsatz um 53% im Vergleich zum Vorjahr steigern. [dot10b]

[2] `http://www.magentocommerce.com/`

Die dotSource GmbH ist zertifizierter Partner für die Shopsoftware Magento und die E-Commerce-Lösung Intershop Enfinity Suite 6 [dot10c]. Weiterhin bietet dotSource GmbH mit SCOOBOX eine eigens entwickelte *Social-Commerce-Lösung* für Enfinity Suite 6 der Intershop AG an [dot10a].

Derzeit ist SCOOBOX („Social Commerce Out Of the BOX“) in der Version 2.1 für Enfinty Suite 6 verfügbar [SC10]. Social Commerce bedeutet hierbei eine Erweiterung von E-Commerce und wird in [Ric07] wie folgt präzisiert: „*Der Social Commerce stellt die zwischenmenschlichen Beziehungen und Interaktionen (den Austausch von Bewertungen, Produktinformationen und Feedback) in den Vordergrund, die vor, während und nach geschäftlichen Transaktionen eine Rolle spielen und setzt damit dem Electronic Commerce eine zusätzliche kooperations- und kommunikationsorientierte Ebene auf.*“

SCOOBOX bietet dem Nutzer verschiedene Social Commerce-Funktionen an [SC10]. An dieser Stelle sollen zwei Kategorien an Funktionen, welche den Social Commerce Aspekt am besten wiedergeben, kurz vorgestellt werden.

Produkt- und Shop-Funktionen

- Die Produkte des Shops können kommentiert und bewertet werden. Durch diese Funktionalität teilen die Nutzer ihre Meinung über die Produkte oder die Kommentare anderer Nutzer mit oder holen die Meinungen für die eigene Kaufentscheidung ein.
- Top-Listen wie z. B. meist gekaufter Produkt, oder der aktivste Nutzer, im Sinne von Kommentare schreiben können angelegt werden.

Web-2.0-Funktionen

- Es gibt einen vollständig in Facebook integrierten Demo-Shop, welcher die vollständige Shopfunktionalität im Facebook nutzen kann.
- Mithilfe der Twitter- und Facebook-APIs lassen sich die Funktionen der Social Networks auch direkt aus dem Shop nutzen.
- Der Blog ist die redaktionelle Komponente von SCOOBOX, welcher die Warenkorbfunktionalität und Anbindung zu Twitter und Facebook hat.

Die praktische Integration von FACT-Finder und Solr werden in die SCOOBOX der Firma dotSource GmbH integriert. Speziell soll der Blog mit der Suchfunktion der genannten Suchplattformen ausgestattet werden und das Ergebnis anschließend für einen Performance Vergleich genutzt werden. Aus diesem Grund wird im Ausblick auf den praktischen Teil im Folgenden ein Ausschnitt aus dem vereinfachten *Entity-Relationship-Modell* zum Blog vorgestellt, welches in der Abbildung 2.5 dargestellt ist.

ScBlog enthält alle Blogs, welche in der Web-Applikation vorhanden sind. Es können mehrere Blogs gleichzeitig für einen Shop angelegt werden.

ScBlogCategory erfasst die Kategorien zu den die Blog-Beiträge zugeordnet werden können. Ein Blog-Beitrag kann zu mehreren Kategorien angehören. eine Kategorie kann mehrere Blog-Beiträge erhalten. Diese Beziehung ist mithilfe der Tabelle *ScBlogPostCategoryAssignment* umgesetzt.

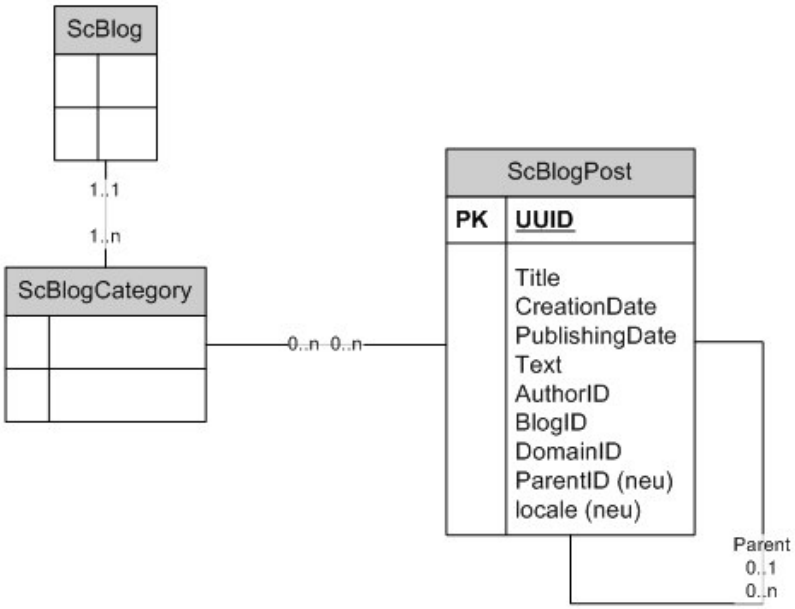

Abbildung 2.5: Ein Ausschnitt aus dem Entity-Relationship-Modell zum Blog

ScBlogPost enthält die Beiträge des Blogs.

Die Daten aus den Tabellen *ScBlogpost* und *ScBlogCategory* werden für die Volltextsuche im Blog verwendet.

Kapitel 3

Anforderungen an Suchplattformen

FACT-Finder und Solr sind in der E-Commerce-Branche sehr erfolgreich und bei dem Einsatz in den Web-Shops weitverbreitet. Aus diesem Grund stellt sich die Frage, welche der beiden Lösungen die bessere Wahl für den Onlineshop-Betreiber ist. Um das zu beantworten, werden in diesem Kapitel die Vergleichskriterien für die Gegenüberstellung der Suchplattformen im E-Commerce-Bereich aufgestellt.

Für einen fairen Vergleich müssen im ersten Schritt objektive Vergleichskriterien in Form von Anforderungen aufgestellt werden. Für die Gegenüberstellung wird im zweiten Schritt der direkte Bezug auf diese Eigenschaften genommen. Die Vorstellung und Erläuterung von den Anforderungen, welche wirtschaftliche und technische Aspekte abdecken, erfolgt in diesem Kapitel. Der direkte Vergleich konkreter Suchplattformen, welche anhand der aufgestellten Anforderungen gegenübergestellt werden, ist im nächsten Kapitel erfasst.

Die Aufstellung von Vergleichskriterien erfordert aufgrund unterschiedlicher Erwartungen und Zielstellungen die Differenzierung zwischen Anforderungen des Shop-Betreibers (vgl. Abschnitt 3.1) und des Shop-Nutzers (vgl. Abschnitt 3.2). Für den Shop-Betreiber sind z. B. folgende Kriterien, die in Zusammenhang mit einer Suchplattform stehen, interessant: Integrationsaufwand in das laufende System, angebotene Features, Einstellungsvielfalt bei der Administration und die mit der Suchplattform entstehenden Kosten. Für den Nutzer ist dagegen eine intuitive und benutzerfreundliche Suche wichtig, welche auf eine Suchanfrage korrekte Ergebnisse liefert und die Suche performant ausführt. Zum Abschluss des Kapitels werden die aufgestellten Vergleichskriterien in dem Abschnitt 3.3 zusammengefasst.

3.1 Shop-Betreiber

Für den Einsatz einer externen Suchplattform in einem Web-Shop muss der Betreiber die Integration in das laufende System durchführen und langfristig gesehen für die Wartung und Aktualisierung der Suchfunktion Sorge tragen, was in Folge einen gewissen finanziellen und technischen Aufwand mit sich bringt. Als Gegenleistung erwartet der Betreiber ein breites Spektrum an zusätzlicher Funktionalität, welche über die Kundenzufriedenheit zu Mehreinnahmen führen soll. Gemäß den genannten Aufgaben und Erwartungen des

Shop-Betreibers werden die Anforderungen in wirtschaftliche (Abschnitt 3.1.1), technische (Abschnitt 3.1.2) und funktionale (Abschnitt 3.1.3) Kriterien kategorisiert.

3.1.1 Wirtschaftliche Anforderungen

Kosten
Wie bereits in der Einleitung des Abschnittes 3.1 erwähnt, ist der Einsatz einer externen Suchplattform mit einem finanziellen Aufwand, welcher aufgrund der anfänglichen Investition, langfristiger Instandhaltung sowie möglicher Erweiterung und Aktualisierung entsteht, verbunden. So können je nach Shop-Betreiber die *Kosten* ein wichtiges Kriterium bei der Auswahl der Suchplattform darstellen.

Preismodell
Bei dem Einsatz von kommerziellen Suchplattformen muss weiterhin der Preis, welcher von dem jeweiligen Anbieter für seine Leistungen verlangt wird, mit in die Kosten eingerechnet werden. In diesem Zusammenhang will der Shop-Betreiber ein angepasstes *Preismodell* nutzen, welches seinen Bedürfnissen genau entspricht. Die Anforderung besteht in der flexiblen Preisgestaltung, welche eine Abrechnung in Abhängigkeit des Ressourcenverbrauchs ermöglicht. Zum Beispiel kann der Preis nach der Anzahl der Serveranfragen oder in Abhängigkeit der genutzten Komponenten gestaltet werden.

Integrationsaufwand
Der *Integrationsaufwand* ist ein wichtiges Kriterium, das bei der Entscheidung über den Einsatz der Suchplattform eine große Rolle spielt. Die Integration sollte in kurzer Zeit und geringem Aufwand für einen bestehenden Onlineshop umgesetzt werden können.

3.1.2 Technische Anforderungen

Die technischen Anforderungen umfassen eine Vielzahl an Kriterien, welche im Folgenden vorgestellt werden.

Konfigurierbarkeit
Bei der Integration der Suche in ein System gibt es eine Fülle an Parametern, welche das Ergebnis der Suche entscheidend beeinflussen können. Die Cache-Einstellung kann zum Beispiel bei richtiger Konfiguration die Suche beschleunigen, jedoch bei falscher Einstellung das System sehr ineffizient machen. Weiterhin ist die Auswahl der Algorithmen ein sehr wichtiger Aspekt. Diese Einstellung ist für die Suche entscheidend und kann je nach Anwendungsfall und Erfordernis für Abstimmung zwischen der genauen und schnellen Suche eingesetzt werden. Langfristig gesehen muss die Suchplattform aufgrund der Wartung des Systems angepasst werden. Das hat den Grund, dass die Inhalte von Onlineshops oft aktualisiert und erweitert werden. Beispiele für solche Inhalte sind Produktneuerungen, deren Beschreibungen sowie von den Nutzern generierte Inhalte wie Kommentare und Artikelbewertungen. Aus diesem Grund ist es wichtig die Indizes zu pflegen, um dem Nutzer stets aktuelle Informationen über die Suchfunktion bereitzustellen.

Die *Konfigurierbarkeit* ist ein wichtiges Kriterium, welches für die Qualität der Suchergebnisse über die Zeit eine entscheidende Rolle spielt. Man muss jedoch beachten, dass die Vielfalt von Konfigurationseinstellungen die Komplexität der Integration erhöht und damit dem geringen Integrationsaufwand entgegen wirkt. Denn je breiter das Spektrum an Konfigurationsmöglichkeiten in einer Suchplattform gegeben ist, umso komplizierter und

zeitaufwendiger ist es ohne ein festes Basiswissen eine optimale Einstellung vorzunehmen. Der Provider will eine schnelle Integration mit Standardeinstellungen und anschließender Möglichkeit zur Feinkonfiguration der Suchplattform.

Skalierbarkeit
Eine Suchplattform muss die *Skalierbarkeit* des Systems bei enormen Datenmengen und hoher Speicherlast, welche im Folgenden verdeutlicht werden, gewährleisten.

1. ***Enorme Datenmengen***
   Der Umfang der Dokumentensammlung hat einen direkten Einfluss auf die Größe der Indizes, welche aufgrund zusätzlicher Informationen wie z. B. Stoppwortlisten und Thesauren (vgl. Abschnitt 2.2.1) den Speicherverbrauch weiterhin erhöhen. Für die effiziente Ausführung der Suche wird der Index i. d. R. in den Arbeitsspeicher geladen. Wenn der Index die Größe des Arbeitsspeichers überschreitet, so muss die Suchplattform Mechanismen zur Skalierung z. B. durch Partition des Indexes und Verteilung auf mehrere Applikationsserver bereitstellen.

2. ***Hohe Speicherlast***
   In der Regel nutzen mehrere Nutzer gleichzeitig dieselbe Suchplattform. Die Suchanfragen werden parallel verarbeitet, wobei die Ausführung einer Suchanfrage ein gewisses Maß an Rechenleistung des Servers erfordert. Damit kann ab einer gewissen Anzahl von parallelen Nutzern die Rechenleistung des Servers für die effiziente Verarbeitung von Suchanfragen nicht mehr ausreichen. Aus diesem Grund muss die Skalierbarkeit der Suchplattform z. B. durch die Verteilung von Suchanfragen auf mehrere Server gewährleistet werden.

Ausfallsicherheit
Die Architektur einer Suchplattform muss für die *Ausfallsicherheit* des Systems sorgen. Ein mögliches Vorgehen für die Umsetzung dieser Anforderung ist die Replikation des Servers auf dem die Suchapplikation ausgeführt wird.

Schnittstellen
Für die Integration der Suche müssen für den Austausch von Daten zwischen dem Web-Shop und der Suchapplikation *Schnittstellen* bereitgestellt werden. Um die Kompatibilität der Suchplattform zu erhöhen und damit ein möglichst breites Spektrum an Web-Shops anzusprechen sollten die Schnittstellen unterschiedliche Programmiersprachen und Kommunikationstechnologien unterstützen.

Unterstützung der Hardware
Die Plattform sollte die Vorteile aktueller Technologien im Bereich der Hardware ausnutzen und beispielsweise 64-Bit-Architekturen und Mehrkernprozessoren unterstützen. Letzteren bieten z. B. die Möglichkeit Verarbeitung von mehreren Suchanfragen auf einzelnen Kernen gleichzeitig auszuführen bzw. bei komplexen Anfragen die Verarbeitungsschritte auf unterschiedlichen Kernen zu parallelisieren um die Gesamtlaufzeit der Berechnung von Suchergebnissen zu verbessern.

Erweiterbarkeit
Die *Erweiterbarkeit* spielt bei Erscheinung neuer Versionen eine wichtige Rolle. So muss die Suchplattform ohne erheblichen Aufwand um neue Funktionalitäten erweitert werden können.

Sprachunabhängigkeit
Ein wichtiges Kriterium für Web-Shops, welche international vertreten sind, ist die *Sprachunabhängigkeit* der Suchplattform. Diese Anforderung kann z. B. durch sprachübergreifende Indizes oder mithilfe automatischer Erkennung der verwendeten Sprache und anschließender Verwendung des Indexes mit zutreffender Lokalisierung gelöst werden. Hierzu muss die Suchplattform unterschiedliche Implementierungen für sprachabhängige Algorithmen bereitstellen wie z. B. phonetische Analyse oder Stammwortreduzierung (vgl. Abschnitt 2.2.1).

3.1.3 Funktionale Anforderungen

Features
Die Suchplattformen bieten neben der Volltextsuche zahlreiche *Features* zur Unterstützung des Nutzers wie z. B. Kategorisierung von Suchergebnissen und Autovervollständigung (vgl. Abschnitte 3.2.3 und 3.2.1). Weiterhin sind Werkzeuge für die Manipulation von Suchergebnissen durch den Shop-Betreiber die Regel. Dadurch kann der Anbieter gezielt der Anordnung von Suchergebnissen beeinflussen um z. B. neue oder meist gekaufte Artikel im Suchergebnis aufzuwerten. Weiterhin sollte die Erstellung von Kampagnen von der Suchplattform unterstützt werden, welche definierte Aktionen bei der Suche nach bestimmten Begriffen ausführt. Beispielsweise kann eine Weiterleitung des Nutzers auf die Seite mit den Allgemeinen Geschäftsbedienungen definiert werden, falls dieser nach dem Begriff „AGB“ sucht.

Analysewerkzeuge
Durch den Einsatz von *Analysewerkzeugen* kann die Performance der Suchapplikation entscheidend verbessert werden. Die Analyse einzelner Vorverarbeitungsschritte und der eigentlichen Suche (vgl. Abschnitt 2.2) bieten die Möglichkeit zur Feinkonfiguration sowie Speicher- und Laufzeitoptimierung der Suchplattform. Weiterhin sind Tools für die Auswertung des Kaufverhaltens der Nutzer insbesondere im Bereich des E-Commerce sehr hilfreich. Auf diese Weise kann der Shop-Betreiber genau nachvollziehen nach welchen Produkten die Kunden suchen und ob diese gefunden werden.

3.2 Shop-Nutzer

Die potenziellen Kunden besuchen in den meisten Fällen einen Onlineshop mit einer klar definierten Vorstellung über die Produkte, die sie kaufen wollen [Zoe01]. Die schnellstmögliche Lösung dabei ist die direkte Eingabe in das Suchfeld mit der Zielstellung die relevanten Ergebnisse zu erhalten. Die Suchfunktion sollte also effizient ablaufen und den Nutzer in dem Suchprozess unterstützen. Im Folgenden werden die Anforderungen in die Kategorien zur Unterstützung der Sucheingabe, Finden und Präsentieren von Suchergebnissen eingeteilt.

3.2.1 Unterstützung der Sucheingabe

Autovervollständigung
Die *Autovervollständigung* (engl. Auto-Suggest) unterstützt den Kunden bei der Suche nach gewünschten Artikeln. Diese Funktion macht während der Eingabe Vorschläge für

mögliche Suchbegriffe. Wählt der Nutzer eines der vorgeschlagenen Suchbegriffe aus, so erhält er mit hundertprozentiger Wahrscheinlichkeit eine nicht leeres Suchergebnis zurück. Der Grund dafür ist, dass die vorgeschlagenen Begriffe als Terme im Index vorliegen. Der Suchbegriff wird zur Eingabezeit mit den Termen aus dem Index auf Ähnlichkeit überprüft und als Vorschlag dem Nutzer angezeigt. [Soc11]

Benutzbarkeit
Die *Benutzbarkeit* stellt eine Anforderung an die Schnittstelle zu dem Shop-Nutzer dar. Die Suchfunktion muss einfach, verständlich und selbsterklärend sein, sodass die Shop-Nutzer die Suche ohne weitere Kenntnisse richtig bedienen können.

3.2.2 Finden von Suchergebnissen

Fehlertoleranz
Mithilfe der *Fehlertoleranz* ist die Suchapplikation in der Lage Suchergebnisse zu liefern auch wenn der Ausdruck von den indizierten Termen etwas abweicht. Fehlertolerante Suche ist ein wichtiges Konzept, weil die Eingabefehler wie ein Buchstabendreher oder das Auslassen einzelner Buchstaben eine sehr häufige Quelle für ein leeres Suchergebnis sind. Die Fehlertoleranz sollte einstellbar sein, weil diese u. U. von verschiedenen Anbietern im unterschiedlichen Maße erwünscht ist.

Performance
Die *Performance* der Suche ist eines der zentralen Aspekte bei der Verarbeitung von Suchanfragen. Es wichtig, dass korrekte Ergebnisse in kurzen Zeiten generiert werden und das Ergebnis möglichst vollständig ist.

3.2.3 Präsentation der Suchergebnisse

Kategorisierung der Suchergebnisse
Die *Kategorisierung der Suchergebnisse* ist unter dem Begriff *After-Search-Navigation (ASN)* bekannt und ist mittlerweile von der Suchfunktion in einem Onlineshop nicht mehr wegzudenken. ASN fasst die Ergebnisse der Suche dynamisch in Kategorien zusammen. Der Nutzer kann im zweiten Schritt die Ergebnismenge über die Kategorien der After-Search-Navigation weiter einschränken. Als ASN-Kategorien können beispielsweise Produktkategorien des Shop-Betreibers, Preis, Hersteller, Farbe usw. verwendet werden. In Abhängigkeit von den Produkten können spezifische Attribute wie die Diagonale bei Fernsehern oder Auflösung bei Digitalkameras für das Einschränken der Suche dienen. Es ist üblich das zu einer ASN-Kategorie die Anzahl der entsprechenden Ergebnisse angezeigt wird. [BYRN11]

Soziale Suche
Die *Soziale Suche* umfasst solche Kriterien wie das Kaufverhalten anderer Nutzer u.ä., welche die Suchergebnisse und ihre Anordnung beeinflussen. In [Soc11] werden mögliche Ausprägungen der sozialen Suche aus dem praktischen Umfeld aufgezählt.

- Die Nutzer beurteilen die Relevanz der Ergebnisse, sodass diese Information bei den ähnlichen Suchanfragen in Zukunft zur Auswertung miteinbezogen wird.
- Die Suche wird mit sozialen Netzwerken verknüpft und mit persönlichen Akzenten oder mit Suchkriterien der Freunde erweitert.

Somit muss die Suchapplikation die Nutzerbewertungen bei der Platzierung von Suchergebnissen einbeziehen und Schnittstellen zu sozialen Netzwerken bereitstellen.

3.3 Zusammenfassung

In der Abbildung 3.1 werden die gestellten Anforderungen grafisch zusammen. Diese werden im nächsten Kapitel für den Vergleich der Suchplattformen FACT-Finder und Solr verwendet. Die Kriterien *Performance* und *Integrationsaufwand* die Ausnahme und werden erst nach der konkreten Implementierung, welche in dem Kapitel 5 stattfindet, ausgewertet.

Abbildung 3.1: Anforderungen an Suchplattformen in E-Commerce

Kapitel 4

FACT-Finder und Solr

In diesem Kapitel werden die Suchplattformen *FACT-Finder* und *Solr* vorgestellt. In den ersten zwei Abschnitten wird ein Einblick zu historischen Hintergründen, bereitgestellten Funktionen sowie technischen Spezifika der jeweiligen Suchlösungen gegeben. Anschließend werden die Suchplattformen anhand der Anforderungen, welche im Kapitel 3 aufgestellt wurden, gegenübergestellt.

4.1 FACT-Finder

FACT-Finder ist eine von Omikron Data Quality GmbH bereitgestellte Such- und Navigationsplattform, welche für die Suche in Online-Shops zum Einsatz kommt. Die Suchplattform wird weltweit in 24 Sprachen und 26 Ländern angeboten [FACT11]. FACT-Finder ist modular aufgebaut und kann unabhängig von der verwendeten Plattform eingesetzt werden. Die folgenden Abschnitte stellen die geschichtliche Entwicklung und die technischen Eigenschaften dieser Suchplattform vor.

4.1.1 Hersteller und Historie

Die Firma Omikron Data Quality GmbH wurde von dem heutigen Geschäftsführer Carsten Klaus im Jahr 1981 gegründet. Omikron entwickelt in der Anfangszeit Programme für Computer von Commodore und Atari. In der Gegenwart bietet Omikron Services im Bereich Datenqualität sowie Softwareprodukte für Suche und Navigation im Bereich E-Commerce an. [Omi11]

Im Jahr 1993 entstand das erste Produkt und wurde auf den Namen *Data Quality Software DBRS* getauft. Diese Software wurde für die Erkennung und Beseitigung von Dubletten, welche u. a. durch Schreibfehler entstehen und doppelte Datensätze z. B. in Adress- oder Produktdaten in einer Datenbank darstellen, eingesetzt. Der Kernalgorithmus stellt eine fragmentarischen Ähnlichkeitsvergleichstechnik und wird als *FACT*, was aus der Kurzschreibweise der englischen Übersetzung *Fragmentary Alikeness Comparing Technique* folgt, bezeichnet [Sch04]. FACT findet heute die Anwendung in der Suchplattform FACT-Finder, welche i. A. für die Produktsuche in E-Commerce-Systemen verwendet wird. Die erste Version von FACT-Finder entstand im Jahr 2001 und ist heute bei der Version 6.6 angelangt. [Omi11]

Neben dem Hauptprodukt FACT-Finder gibt es die Speziallösungen *FACT-Finder eXpress* und *FACT-Finder Travel*. FACT-Finder eXpress zielt mit der vereinfachten Variante von FACT-Finder auf kleine, mittelständische und saisonabhängige Web-Shops ab. FACT-Finder Travel ist eine Speziallösung für Online-Reiseportale, welche für die Übersetzung von individuellen Reisewünschen eingesetzt wird. [Sol11]

Im Folgenden wird der verallgemeinerter Einsatz von FACT-Finder betrachtet, welcher für die Suche nach abstrakten Inhalten neben der Produktsuche durchaus praktikabel ist. Ein praktischer Anwendungsfall wird im Kapitel 5 für das Blog-System von SCOOBOX vorgestellt, welches im Rahmen dieser Arbeit implementiert wurde.

4.1.2 Funktionen

Neben der traditionellen Suche stellt FACT-Finder eine umfassende Funktionalität für zusätzliche Interaktion mit dem jeweiligen Shop und Werkzeuge für die Analyse und Konfiguration bereit. In der Abbildung 4.1 sind die Kategorien mit den jeweils enthaltenen Funktionen in einem Kreisdiagramm dargestellt. Im inneren Kreis der Abbildung sind die

Abbildung 4.1: Funktionen von FACT-Finder aus [FFe10]

Funktionspakete, welche aus mehreren Basisfunktionen des äußeren Kreises gebildet werden, dargestellt. Die einzelnen Funktionen werden im Folgenden gemäß [Int11] genauer betrachtet.

Search, Conversion & Analytics:
Die Kernfunktionen dieses Pakets und der Suchplattform allgemein ist die *Search Engine*, welche u. a. den Kernalgorithmus FACT (vgl. Abschnitt 4.1.3) enthält, und *Data Center*, das die Analyse von Anfragen und Suchergebnissen als Log-Dateien erfasst.

Automatic Search Result Optimisation hat die Aufgabe Suchergebnisse anhand des Verhaltens der Online-Kunden aus Statistiken über die Klick-, Kaufdaten und Navigationsverlauf automatisch zu optimieren. Klicken z. B. die Online-Kunden bei der Suche nach den Begriffen „Nokia N-Serie“oft das Produkt „Nokia N95g“an, so wird diese Produkt in der Ergebnisliste von „Nokia N-Serie“schrittweise nach vorne gesetzt.

Das *Business User Cockpit* ist ein Konfigurationsportal des FACT-Finders, welches in Form von HTML-Templates zur Integration in Backend-Bereich der Anwendung Zugriff auf diverse Einstellungen des Servers bereitstellt. Die Konfigurationseinstellungen bieten einen wichtigen Anhaltspunkt für die Optimierung des Suchservers und umfassen u. a. die Verwaltung von Thesauren, Ergebnisplatzierungen, Filterung von Suchergebnissen und Kampagnen. Neben der Konfiguration können die Log-Dateien mit den bei der Suche und Navigation gesammelten Statistiken über die gesuchten Begriffe und das Nutzerverhalten abgerufen werden.

User Experience & Frontend Design:
Diese Kategorie umfasst drei weitere Funktionen, von den *Suggest* der Autovervollständigung aus dem Abschnitt 3.2.1 entspricht. Die Vorschläge der Suggest-Funktion von FACT-Finder werden auf Basis eines separaten Indexes generiert, welcher im Gegensatz zu dem allgemeinen Index stark in den Konfigurationsmöglichkeiten eingeschränkt ist.

Weiterhin werden so genannte *Tag-Clouds* unterstützt. Diese visualisieren zusätzlich zu dem Suchergebnis Wortwolken aus Verlinkungen zu bestimmten Artikeln des Shops. Die Wortwolken sind abhängig von der Suchanfrage und bestehen aus ähnlichen Artikeln sowie Synonymen von Sucheingaben und Suchergebnissen.

Weiterhin bietet FACT-Finder einen nach den indizierten Produkten aufbereiteten *Produktvergleich* als eigenständige Funktion an.

Dynamic Filter & Faceted Navigation:
Diese Kategorie bietet die Funktionalität im Sinne der Kundenanforderung ASN aus dem Abschnitt 3.2.3. Die indizierten Attribute, die im Shop als *Facetten* dargestellt werden sollen, kann der Shop-Betreiber selbst in der Konfiguration des Business-User-Cockpit festlegen. Klickt der Nutzer auf eine Kategorie der ASN, so werden die Facetten bezüglich des gefilterten Ergebnisses neu berechnet.

Weiterhin kann die gesamte *Shop Navigation* über FACT-Finder geregelt werden. Das bedeutet, dass der komplette Struktur der Shop-Kategorien mit dem FACT-Finder indiziert wird, sodass bei der Navigation innerhalb des Shops die Subkategorien und enthaltene Produkte durch den FACT-Finder geliefert werden.

Merchandising & Recommendation:
Mithilfe der *Recomendation Engine* analysiert FACT-Finder die Einkäufe im Onlineshop und ermittelt auf Basis von Suchverhalten und Verkaufsdaten, welche von dem Shop-Betreiber separat gestellt werden müssen, Verwandtschaften unter den Produkten und

Kategorien. Bei Verwandtschaften kann es sich um ähnliche und zugehörige Artikel wie z. B. eine Schutzhülle zu einem Mobiltelefon handeln. Die Verkaufsdaten können entweder über eine Schnittstelle oder in einer Datei an den Server übermittelt werden. Die Recomendation Engine kommt dann als eine Empfehlung zum Einsatz, wenn Kunden nach gleichen oder ähnlichen Artikel suchen bzw. einkaufen.

Die Funktion *SEO Enhancer* generiert bei der Navigation innerhalb des Web-Shops URLs, welche für die Suchmaschinen optimiert werden. FACT-Finder generiert dabei dynamisch die geeigneten Begriffe in der URL wie z. B. den Titel eines Artikel anstatt seiner Identifikationsnummer. Funktion setzt allerdings den Einsatz von FACT-Finder in der Shop-Navigation voraus.

Mit dem *Campaign Manager* ist es möglich für bestimmte Suchbegriffe definierte Aktionen auszuführen (vgl. Abschnitt 3.1.3). Bei der Erstellung einer Kampagne kann der Shop-Betreiber zwischen einer Feedback- und einer Weiterleitungsaktion auswählen. Bei dem Feedback-Ansatz können vorab ausgewählte Produkte hervorgehoben werden und mit Hilfe von Text und Bannern zum Beispiel auf Rabatt-Aktionen aufmerksam machen. Bei der Weiterleitung wird der Nutzer auf eine zuvor festgelegt Seite verlinkt sobald sie nach bestimmten Begriffen suchen. Eine Kampagne wird immer für einen bestimmten Zeitabschnitt definiert.

Mobile Commerce:
FACT-Finder unterstützt mithilfe der *FACT-Finder App* die Suche in der mobilen Variante des Web-Shops, welche speziell für die Produkte iPhone und iPad der Firma Apple angepasst ist.

Die Module, welche in der Einleitung des Abschnittes 4.1 angesprochen wurden, entsprechen den vorgestellten Funktionspaketen, welche von dem Betreiber komponentenweise in den Web-Shop integriert werden können. Bei der Integration von FACT-Finder sind immer die Funktionen des Moduls *Search, Conversion & Analytics* als Basis enthalten. Andere Funktionspakete oder einzelne Komponenten dieser können gegen Aufpreis zu den Basisfunktionen angefordert werden.

FACT-Finder kann entweder als eine SaaS (Software-as-a-Service) Applikation gegen einen zu zahlenden monatlichen Betrag abonniert oder als eine komplette Lösung gegen eine einmalige Lizenzgebühr gekauft werden. Im ersten Fall tritt Omikron als ein ASP (Application Service Provider) auf, welcher die Wartung und die Pflege einer auf den Shop abgestimmten Hardware-Infrastruktur komplett übernimmt. Im Fall des kompletten Kaufs liegt der gesamte Wartungs- und Pflegeaufwand bei dem Shop-Betreiber selbst. [Int11]

4.1.3 Such-Engine

FACT-Finder basiert auf Algorithmen, welche darauf konzipiert sind Schreibfehler, Wort- und Wortteilumstellungen Fehler aufzuspüren. [Bli01]

Der Kernalgorithmus der Suchmaschine FACT-Finder ist der patentierte Algorithmus FACT, welcher in der Programmiersprache C geschrieben ist. FACT ist ein Verfahren, das zwei Zeichenketten (Strings) vergleicht und ein Ähnlichkeitsmaß zwischen 0 und 100 Prozent ermittelt [FACT11]. Dieser Algorithmus basiert auf mathematisch-linguistischen Verfahren, welchem das Fuzzy-Set-Modell (vgl. Abschnitt 2.2.3) zugrunde liegt und eine

unscharfe Suche auf dem Datenbestand ausführen. Wenn ein nicht exakt mit dem Suchbegriff übereinstimmendes Wort gefunden wird, so wird unter der Berücksichtigung des ermittelten Ähnlichkeitsmaßes eine Abwertung des Dokuments in der Trefferliste vorgenommen. [Sch04]

Zusätzlich zu FACT wird eine Komponente für den phonetischen Vergleich von Zeichenketten verwendet [FACT11]. Die sogenannte *Omikron Phonetik* ermittelt nach dem Vergleich von zwei Zeichenketten, ob diese phonetisch ähnlich oder nicht ähnlich sind. Im Gegensatz zu FACT ist die Omikron Phonetik an sprachliche Spezifika gebunden (vgl. Abschnitt 2.2.1). [Sch04]

Die Suchsyntax von FACT unterstützt die Suche nach Termen oder Phrasen inklusive Wildcard-Suche sowie Boolesche Operatoren wie NOT, AND und OR. [Int11]

4.1.4 Architektur

Die Architektur des FACT-Finder Servers besteht im Wesentlichen aus den Komponenten Search Engine, Data Center und die Konfigurationsoberfläche zur Verwaltung und Administration. Die Grundbausteine entsprechen somit den Komponenten des Funktionspaktes *Search, Conversion & Analytics* (vgl. 4.1.2). In der Abbildung 4.2 ist das Zusammenspiel dieser Komponenten mit einer Web-Applikation schematisch dargestellt, welches im Folgenden erläutert wird.

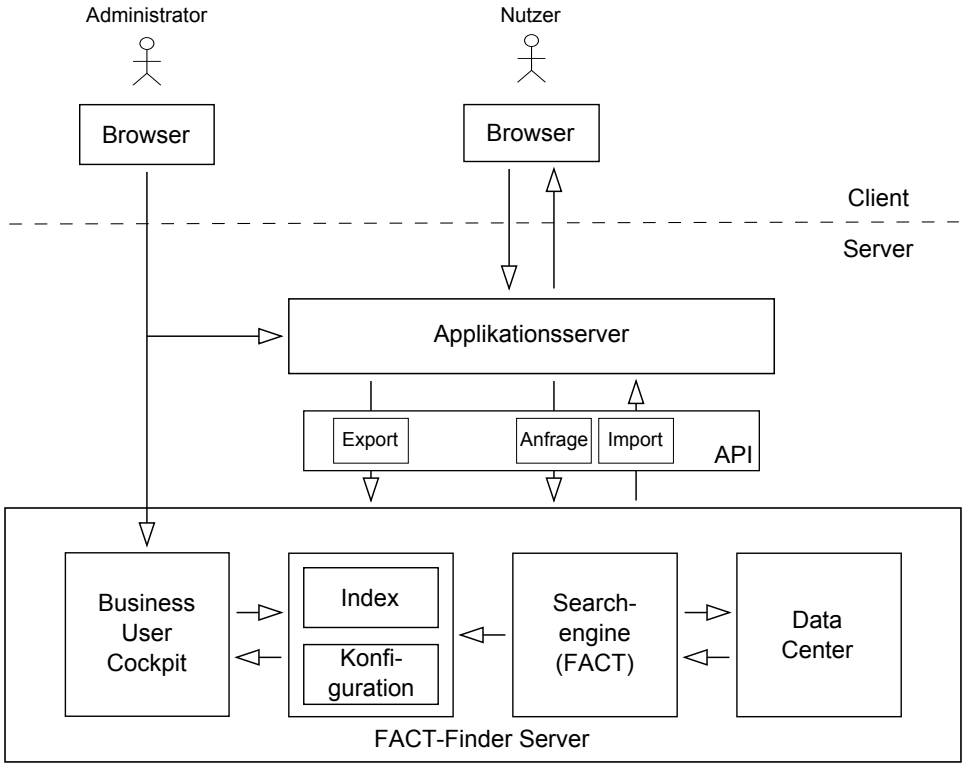

Abbildung 4.2: Interaktion zwischen Web-Applikation und FACT-Finder Server

Der Zugriff von der Client-Seite auf jeweilige Server erfolgt über einen Browser. Der Nutzer stellt eine Suchanfrage an den Applikationsserver, auf dem eine Anwendung wie z. B.

ein Web-Shop geladen ist, und erhält als Antwort eine Seite, auf der die Suchergebnisse dargestellt werden. Für die Ausführung der Suche findet zwischen dem Applikationsserver und dem FACT-Finder Server der Datenaustausch über eine der vier Schnittstellen, welche im Abschnitt 4.1.4.2 vorgestellt werden, statt. Die Schnittstelle sorgt für den *Export* und *Import* von Daten sowie die Aufbereitung von *Suchanfragen*. Bei dem Export werden die Daten von der Applikation zum FACT-Finder Server transportiert. Dieser Prozess wird von dem Administrator aus dem Backend der Anwendung angestoßen und sorgt für den Aufbau des Indexes, welcher im Abschnitt 4.1.4.1 vorgestellt wird. Die Suchanfragen werden von der Search-Engine verarbeitet und die Ergebnisse über die Import-Schnittstelle an die Applikation zurückgeliefert. Für die Generierung von Suchergebnissen nutzt die Search-Engine den Index und die Log-Dateien des Data Centers. Die Konfiguration der Indizes und des FACT-Finder Servers wird von dem Administrator über das *Business User Cockpit* vorgenommen.

4.1.4.1 Index

Der FACT-Finder Index besteht aus drei Bestandteilen:

1. eine Comma-Separated-Values (CSV)-Datei, welche für den Export von Daten der Applikation zum FACT-Finder Server verwendet wird,
2. zwei Suchdatenbankdateien, von den eine für den regulären Index und eine für die Suggest-Funktion (vgl. Abschnitt 4.1.2) verwendet wird, und
3. in XML-Format vorliegenden Konfigurationsdateien, welche für die Suchdatenbankdateien und für Kampagnen angelegt werden.

Nach der Aussage eines Mitarbeiters der Firma Omikron Data Quality GmbH kann der FACT-Finder Index jede Art von textuellen Daten speichern. Grundsätzlich wird zwischen Zeichenketten und Zahlen unterschieden, sodass z. B. eine Facette der ASN (vgl. Abschnitt 4.1.2) je nach Datentyp alphabetisch oder numerisch sortiert wird.

Bei der Indexerstellung wird die Standard-Konfiguration aus der Voreinstellung des Suchservers initial angelegt. Diese Konfiguration kann zu jedem weiteren Zeitpunkt manuell oder über das Business User Cockpit (vgl. Abschnitt 4.1.4) angepasst werden. Im nächsten Schritt muss für den Suchprozess die Suchdatenbankdatei erstellt werden. In dieser Phase wird von der Applikation eine CSV-Datei generiert und an den FACT-Finder Server gesendet. In der CSV-Datei sind die zu indizierenden Attribute mit Namen und Werten erfasst. Diese Datei liefert die Datengrundlage für den FACT-Finder Index. [Doc11]

Die parallele Ausführung beim Anlegen mehrerer Indizes konnte bei der praktischen Anwendung von FACT-Finder im Rahmen der vorliegenden Arbeit erfolgreich durchgeführt werden.

Weiterhin ist der Prozess der Aktualisierung des Indexes ein wiederkehrender Prozess. Wird die Konfiguration verändert oder kommen neue Daten hinzu, so muss der Index aktualisiert werden. Im ersten Fall ist die Vermittlung der Konfigurationsdateien an den FACT-Finder Server ausreichend. Bei Erweiterung des zu indizierenden Datenbestandes ist dagegen eine Indexerstellung von Nöten. Neben dem kompletten Indexupdate, ist es möglich bei geringfügigen Veränderungen eine Teilaktualisierung durchzuführen. [Doc11]

Laut eines Mitarbeiters von Omikron Data Quality GmbH wird keine Indexkomprimierung vorgenommen, um die Performance bei der Verarbeitung von Suchanfragen nicht zu beeinträchtigen.

4.1.4.2 Schnittstellen

Bei der Integration von FACT-Finder besteht die Hauptaufgabe des Entwicklers in der Implementierung der Schnittstelle für den Datenaustausch zwischen FACT-Finder und der vorliegenden Applikation. Bei dieser Aufgabenstellung sind die Komponenten für den Export und Import von Daten sowie Übertragung der Suchanfragen zu implementieren.

Export
Wie in dem Abschnitt 4.1.4.1 bereits beschrieben wurde, erfolgt der Export der Daten an den FACT-Finder Server mittels der CSV-Datei, welche die zu indizierenden Datensätze erfasst.

Suchanfrage
Die Suchanfragen der Nutzer werden auf zwei Arten an den FACT-Finder Server übermittelt. Bei der Webservice-Methode wird ein Programm-Objekt mit den Eigenschaften der Suchanfrage erstellt und versendet. Für die HTML-, XML- und JSON-Methoden wird die Suchanfrage als ein Parameter in die URL eingebaut und an den Server ausgeliefert. [Int11]

Import
Für den Import (vgl. Abschnitt 4.1.4) stellt FACT-Finder vier Schnittstellen bereit, sodass die Ergebnisse entweder als HTML, XML, Webservice oder JavaScript Object Notation (JSON) empfangen werden. Dieses Konzept erlaubt dem Anwendungsentwickler einen gewissen Freiraum in der Wahl der zur Implementierung eingesetzten Programmentwurfskonstrukte. Im Folgenden werden die vier Möglichkeiten für den Datenimport auf Basis von [Int11] beschrieben.

- *HTML*
  Bei dieser Variante des Datenimports generiert der FACT-Finder Server eine fertige HTML-Seite, welche die Suchergebnisse bereits vollständig integriert hat. Diese Seite muss nur noch im Shop eingebunden werden. Der HTML-Import ist in der Abbildung 4.3 vereinfacht abgebildet.

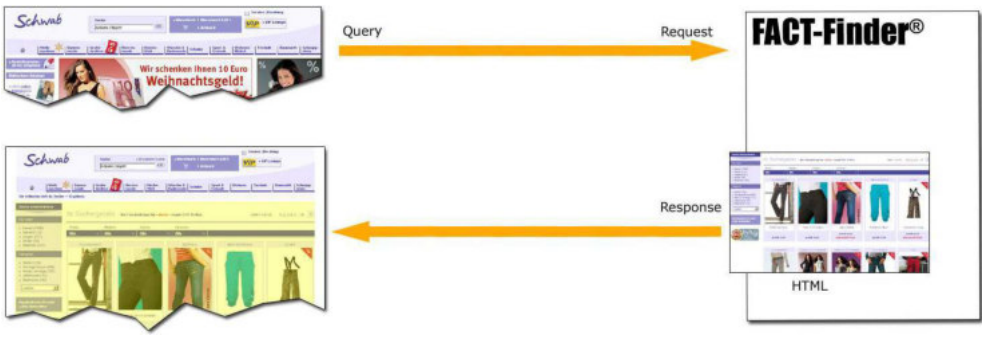

Abbildung 4.3: HTML-Schnittstelle von FACT-Finder aus [Int11]

Der Suchprozess wird mit der Übermittlung des Suchbegriffs an den FACT-Finder Server angestoßen. Nach der Verarbeitung wird das fertige HTML-Konstrukt an die Applikation zurück versendet. Das HTML-Template mit dem integrierten Suchergebnis wird in der Anwendung z. B. als ein Inlineframe (iFrame) eingebunden. Die Gestaltung und der Funktionsumfang der Templates wird während der Spezifikationsphase an die Entwickler von Omikron vermittelt.

- *XML*
  In der Abbildung 4.4 ist die vereinfachte Datenübermittlung mittels der XML-Schnittstelle dargestellt.

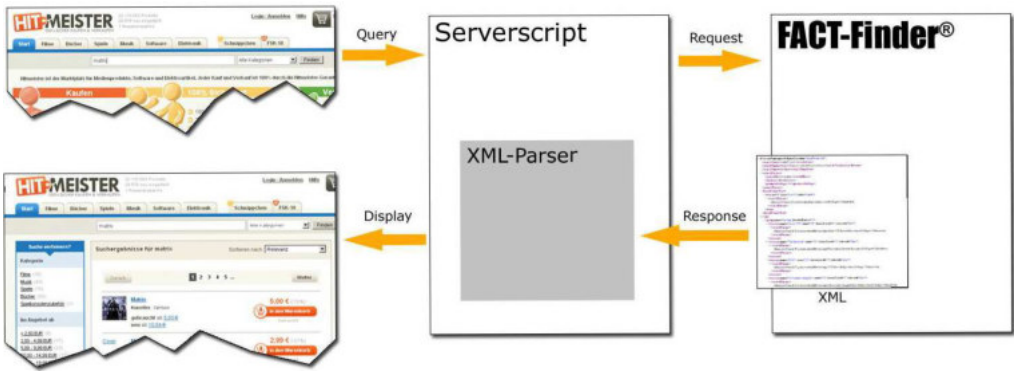

Abbildung 4.4: XML-Schnittstelle von FACT-Finder aus [Int11]

Die Suchbegriffe werden als URL-Parameter an den FACT-Finder Server übermittelt. Das Suchergebnis wird als ein XML-Konstrukt an die Anwendung zurück versendet. Für den Datenaustausch zwischen der Applikation und dem FACT-Finder Server muss ein Serverscript von dem Betreiber der Anwendung geschrieben werden. Seine Aufgabe besteht darin die Suchergebnisse aus der XML-Datei zu extrahieren und auf die Objekte, welche in der Applikation verwendet werden, abzubilden.

- *Webservice*
  Bei dieser angebotenen Schnittstelle wird die Suchanfrage über Web Services Description Language (WSDL)-Objekte an den FACT-Finder Server übermittelt. Das Empfangen und die Verarbeitung der Suchergebnisse wird automatisiert über den WSDL-Code erledigt. Danach stehen die Suchergebnisse als Objekte der Applikation, für die anschließende Integration in der Oberfläche der Anwendung, bereit.

- *JSON*
  Eine weitere Schnittstelle nutzt die JavaScript Object Notation (JSON). Die Suchanfrage wird wie bei der XML-Variante parametrisiert in der URL versendet. Die Antwort enthält die Suchergebnisse in Form von JSON-Objekten, welche dann in die Darstellung der Anwendung integriert werden.

Zur Realisierung des Datentransfers mit dem FACT-Finder Server kann sich der Shop-Betreiber für eine der vier Integrationsmethoden entscheiden. Zusammenfassend werden die vier Methoden des Datenimport in der Tabelle 4.1 bewertet.

Der *Aufwand* bewertet den Umfang und den Schwierigkeitsgrad der Implementierungen, die auf der Seite des Shop-Betreibers umgesetzt werden müssen. Bei der HTML-Integration

	HTML	*XML*	*Webservice*	*JSON*
Aufwand	gering	mittel	mittel	mittel
Programmierkenntnisse	gering	hoch	hoch	hoch
Flexibilität	gering	hoch	hoch	hoch
Kosten	hoch	gering	gering	gering

Tabelle 4.1: Vergleich der Integrationsmethoden aus [Int11]

muss lediglich eine fertige HTML-Seite in den Shop eingebunden werden. Bei den restlichen Varianten müssen dagegen die Ergebnisse verarbeitet und für die Darstellung aufbereitet werden.

Für die Implementierung der Schnittstellen sind *Programmierkenntnisse* erforderlich. Die XML- und Webservice-Schnittstellen haben ähnlich hohen Implementierungsaufwand und erfordern umfangreiche Kenntnisse in der XML- und WSDL-Verarbeitung, was die vorausgesetzten hohen Kenntnisse in der Programmierung erklärt.

Falls FACT-Finder als eine Application Service Provider Lösung verwendet wird, ist die *Flexibilität* auf der Seite des Shop-Betreibers bei der Änderung der Templates im Fall von HTML-Integration nicht gegeben. In so einem Fall müssen die Änderungen von den Entwicklern der Omikron Data Quality GmbH durchgeführt werden.

Die *Kosten* gegenüber Omikron sind entsprechend im Fall der HTML-Integration höher, weil hier der komplette Implementierungsaufwand auf Seite der Omikron Data Quality GmbH fällt.

4.1.4.3 Cluster-Unterstützung

Bei Anwendungen, wo von Anfang bekannt ist, dass die Ressourcenanforderungen die Performance-Grenzen der Suchmaschine übersteigen können, gibt es die Möglichkeit einen FACT-Finder Cluster aufzubauen, welches in der Abbildung 4.5 schematisch dargestellt ist. Auf diese Weise wird die Last über mehrere Suchserver verteilt. Die Bestandteile und deren Funktion werden im Folgenden gemäß [Doc11] erläutert.

Das Cluster besteht aus einem Master und mehreren Slaves, wobei die Konfiguration und die Suchdatenbank zentral auf dem Master-Server verwaltet werden und die Slave-Server für die Verarbeitung der Suchanfragen zuständig sind. Durch die Replikation besitzen alle Slaves dieselbe Konfiguration und Suchdatenbank des Masters. Auf diese Weise wird gewährleistet, dass die Suchergebnisse von Nutzeranfragen auf allen Suchservern übereinstimmen.

Bei der Änderung der Suchdaten (Konfiguration oder Datenbank) wird die Aktualisierung je nach Konfiguration des Clusters auf allen Slaves entweder automatisiert oder über einen HTTP-Request angestoßen. Um den Konflikt bei dem Zugriff auf die Log-Dateien, welche u. a. für Funktionen wie Autovervollständigung, Analysen und Suchoptimierung verwendet werden, zu vermeiden, werden diese in die Verzeichnisse der jeweiligen Suchserver geschrieben. Für die Analyse und Auswertung werden die Log-Dateien zusammengeführt.

Die Suchanfragen der Nutzer werden zentral über den Master-Server entgegengenommen und mithilfe der Serverlastverteilung für die Verarbeitung an die Slave-Server weitergereicht. Die Serverlastverteilung kann z. B. auf einfache Art umgesetzt werden, wobei die

Anfragen nacheinander einzelnen Slaves zugewiesen werden. Für die Optimierung der Serverlastverteilung kann beispielsweise die Berücksichtigung der Verfügbarkeit einzelner Slaves beachtet werden, sodass z. B. mit komplexen Importen belastete Server keine Anfragen bekommen.

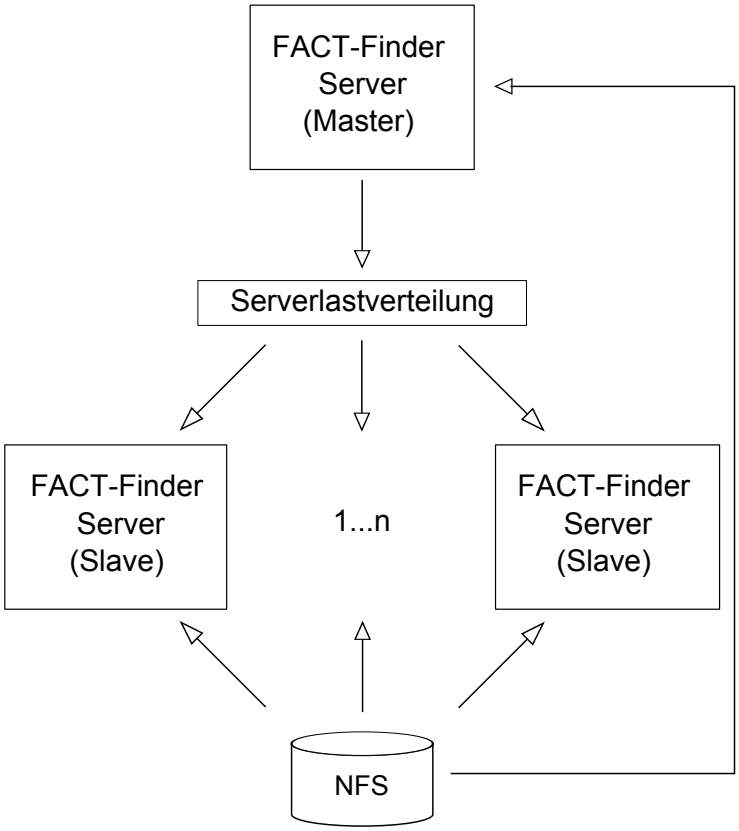

Abbildung 4.5: Architektur eines FACT-Finder Clusters

Für die Optimierung und Vereinfachung der Suchdatensynchronisation empfiehlt Omikron die Verwendung eines Netzwerklaufwerks (NFS), welches die Notwendigkeit der Replikation aufhebt. NFS kann entweder direkt im Master-Server integriert sein oder wie das in der Abbildung des FACT-Finder Clusters veranschaulicht ist, als ein eigenständiger Server vorliegen. Bei dieser Vorgehensweise wird die komplette Konfiguration und die Suchdatenbank auf NFS verlagert, sodass die Server direkt darauf zugreifen können. Weiterhin besteht bei diesem Aufbau der Vorteil, dass ein gemeinsames Cache-Verzeichnis genutzt werden kann. Im Gegensatz zu der Variante ohne NFS, wo auf jedem Server das Caching lokal umgesetzt wird, ist bei diesem Aufbau die Wahrscheinlichkeit eines Treffers höher, was in Folge die Performance des Gesamtsystems steigern kann.

4.2 Solr

Solr ist eine weitverbreitete Suchplattform von Apache Software Foundation[1], welche eine Volltextsuche mit zahlreichen Features bereitstellt. Solr basiert auf dem Suchkern Lucene

[1] http://www.apache.org/

(vgl. Abschnitt 4.2.3) und erweitert diesen mit zusätzlichen Funktionen und Webkomponenten zu einem Suchserver [Dro10]. In den folgenden Abschnitten werden nach einer historischen Einleitung die Funktionen und anschließend die technischen Eigenschaften dieser Suchplattform vorgestellt.

4.2.1 Hersteller und Historie

Die Suchplattform Solr wurde von Yonik Seeley entwickelt und startete 2004 als ein internes Projekt des Medienunternehmens CNET Networks[2][Min09]. Der Code von Solr wurde nach zwei Jahren der Apache Software Foundation gespendet und startete zunächst als ein Apache Incubator Project. Im Januar 2007 wächst Solr zu einem Unterprojekt von Apache Lucene und wird heute in der Community ständig weiterentwickelt. Neben der Community existieren auch kommerzielle Firmen wie Lucid Imagination[3], welche auf Solr aufbauen und die Entwicklung von Solr ebenfalls voran treiben. Derzeit ist Solr in der Version 3.5 auf der offiziellen Seite des Projekts veröffentlicht und kann im Rahmen der Apache-Lizenz[4] frei verwendet werden [Sol11].

4.2.2 Funktionen

In [Int11] werden die Besonderheiten der Suchplattform Solr in sechs Hauptkategorien unterteilt, wobei im Einzelnen technische und funktionale Eigenschaften vermischt werden. Aus diesem Grund ist die Struktur dieser Kategorisierung im Hinblick auf den anschließenden Vergleich von FACT-Finder und Solr ungeeignet. Folglich werden die Funktionen von Solr die Kategorisierung von FACT-Finder aus dem Abschnitt 4.1.2 als Richtlinie verwenden.

Search, Conversion & Analytics:
Die Volltextsuche der Suchplattform Solr verwendet die *Lucene*-Bibliothek, welche im Abschnitt 4.2.3 vorgestellt wird.

Das *Schema* ist eine im XML-Format vorliegende Definition der Struktur eines Dokuments aus der Dokumentensammlung, welche für jeden Index in der Datei *schema.xml* angelegt wird und u. a. die zu indizierenden Felder der Dokumente und Definitionen der Felddatentypen enthält [Ing07a]. Bei der Definition der indizierten Felder können vordefinierte Datentypen verwendet werden, welche von der Lucene-Bibliothek bereitgestellt werden. Falls komplexere Datentypen erforderlich sind, können diese definiert und eingebunden werden. Für die Analyse und Filterung der Felder im Sinne des Abschnittes 2.2.1 stehen konkrete Algorithmen der Lucene-Bibliothek bereit, wobei auch hier der Einsatz eigener Implementierungen nicht ausgeschlossen ist. [Sol09]

Die zentrale Konfiguration eines Suchindizes wird in der Datei *solrconfig.xml* erfasst, welche die Einstellungen zur Konfiguration des Indizierungs- und Retrieval Prozesses (vgl. Abschnitt 2.2) enthält. [Ing07b]

Das *Admin Interface* ist eine Oberfläche für die administrativen Aufgaben, die über einen Browser die wichtigsten Systemeigenschaften, nützliche Statistiken über das System sowie Werkzeuge zum Testen und Debuggen der Suchplattform bereitstellt. Die Statistiken

[2] http://www.cnet.com/
[3] http://www.lucidimagination.com/
[4] http://www.apache.org/licenses/

über die Cache-Auslastung, Index-Aktualisierung und Suchanfragen sind sehr hilfreiche Informationen für die Optimierung der Index- und Systemeinstellungen. Das Debuggen der Analyse- und Filtermethoden veranschaulicht die Zwischenergebnisse zum besseren Verständnis der gesamten Vorverarbeitung der zu indizierenden Index-Felder. Das Testen von Suchanfragen bietet eine detaillierte Aufschlüsselung darüber, wie die Bewertung der einzelnen Treffer im Suchergebnis zustande kommen. [Dro10]

User Experience & Frontend Design:
Gemäß der ersten Anforderung des Abschnittes 3.2.1 wird eine Suggest-Funktion unterstützt, welche in der Datei *solrconfig.xml* konfiguriert werden kann. Die Eingabeunterstützung kann entweder auf dem Index für die Volltextsuche oder auf einem separaten Index ausgeführt werden. [Sol09]

Dynamic Filter & Faceted Navigation:
Die Suchplattform Solr unterstützt gemäß der Anforderung aus dem Abschnitt 3.2.3 die Kategorisierung von Suchergebnissen mithilfe von *Facetten*. Die Kategorisierung der Suchergebnisse braucht in Solr nicht explizit konfiguriert werden [Ing07b]. Um die Facetten bei einer Suchanfrage zu erhalten, müssen der Suchanfrage zusätzliche Parameter übergeben werden. Die Parameter geben die Kategorie und den Facettentyp an. Facettentyp gibt explizit an, ob es sich um ein Feld wie Hersteller oder um einen Bereich wie z. B. der Preis zwischen 10 und 20 Euro handelt. [See09]

Merchandising & Recommendation:
Mithilfe der *MoreLikeThis*-Komponente werden in Solr die Dokumente, welche zu der gestellten Suchanfrage ähnlich sind, in das Suchergebnis mitaufgenommen. Nach dem eine Suchanfrage ausgeführt ist, werden im ersten Schritt die wichtigsten Terme durch die Analyse der zugehörigen Gewichte ermittelt. Falls die Gewichte nicht vorliegen, werden diese von der MoreLikeThis-Komponente berechnet. Danach wird die zuvor gestellte Suchanfrage mit den ermittelten Termen erweitert und erneut ausgeführt. Diese Funktionalität kann für jedes Feld in der Datei *schema.xml* ein- bzw. abgeschaltet werden. [Ing08]

Durch die *QueryElevation*-Komponente kann der Shop-Betreiber die Suchergebnisplatzierungen gezielt beeinflussen oder ausgewählte Dokumente aus dem Suchergebnis komplett ausschließen. Dazu muss in der Datei *elevate.xml* der Suchbegriff, für den die Manipulation der Rangordnung angewandt werden soll, und die Dokumente in der gewünschten Reihenfolge eingetragen werden. Falls ein Dokument aus dem Suchergebnis ausgeschlossen werden muss, so wird das mit der zusätzlichen Eigenschaft gesetzt. Auf diese Weise werden gewünschte Dokumnete für ausgewählte Suchbegriffe vor den Dokumenten, welche von Solr regulär angeordnet wurden, angezeigt. Diese Funktion kann in der Konfigurationsdatei *solrconfig.xml* ein bzw. ausgeschaltet werden. [Ing08]

4.2.3 Such-Engine

Die Suchmaschine Solr basiert auf der Suchengine Lucene. Diese hält Mittel für die Indizierung von Texten bereit und ist in der Programmiersprache Java geschrieben. Das Suchmaschinen-Framework wurde 1997 von Doug Cutting entwickelt. Im Jahr 2001 wurde Lucene zu einem Projekt von Apache Foundation und gilt seit 2005 als ein Top-Level-Projekt. [Rod10]

Lucene ist ein Suchmaschinen-Framework, welches auf der Kombination aus dem Booleschen und dem Vektorraummodell (vgl. Abschnitt 2.2.3) aufbaut. Das Boolesche Modell

ist erweitert und wird für die Unterstützung der booleschen Operatoren und Ausführung von unscharfer Suche verwendet. Das Vektorraummodell bildet die Grundlage für die Gewichtung der Terme und Berechnung der Dokumentenplatzierungen im Suchergebnis. Die Gewichtung der Terme erfolgt nach einer modifizierten Variante der *TF-IDF*-Methode, welche im Abschnitt 2.2.3 für das Vektorraummodell erläutert wurde. Die Dokumentenplatzierung im Suchergebnis erfolgt nach Lucenes Scoring-Formel, welche in der Java-Dokumentation[5] der Klasse *org.apache.lucene.search.Similarity* detailliert erläutert wird. [Sco11]

Lucene unterstützt die Suche nach Termen oder Phrasen inklusive Wildcard-Suche, kann mit Booleschen Operatoren umgehen, führt Bereichsanfragen aus und ist in der Lage unscharf mittels implementierter Fuzzy-Algorithmen zu suchen. Weiterhin ist es möglich die Suchterme phonetisch mit den Index-Einträgen zu vergleichen. [Sol09]

4.2.4 Architektur

Die Architektur der Suchplattform Solr besteht im Wesentlichen aus dem Lucene-Kern, den Schnittstellen für die Indexierung und Anfrageverarbeitung, der Administrationsoberfläche und dem Solr-Kern [Ste09]. Die Komponenten des Solr Servers und die Interaktion mit der Web-Applikation ist in der Abbildung 4.6 vereinfacht dargestellt und wird im Folgenden erläutert.

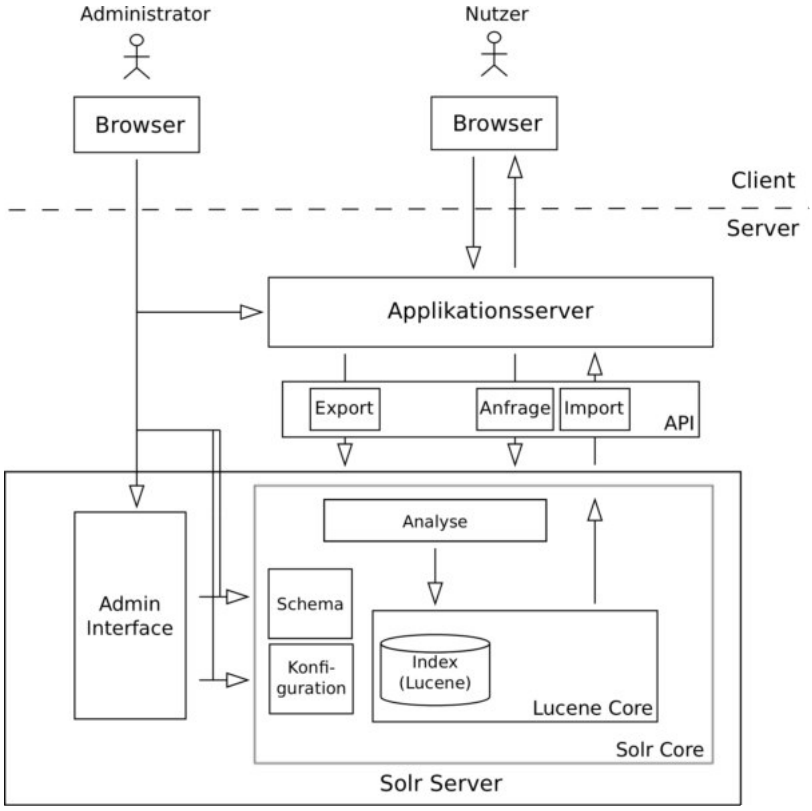

Abbildung 4.6: Interaktion zwischen Web-Applikation und Solr Server

[5] Javadoc (Lucene 3.5): `http://lucene.apache.org/java/3_5_0/api/all/index.html`

Die Interaktion zwischen Applikations- und Suchserver wird genau wie im Abschnitt 4.1.4 durch die Schnittstellen für den Export und Import von Daten sowie Übertragung der Suchanfragen definiert. Die Schnittstellen des Solr Servers werden im Abschnitt 4.2.4.2 erläutert. Die Dokumente werden als Lucene-Indexe, welche im Abschnitt 4.2.4.1 vorgestellt werden, gespeichert. Für jeden Index wird ein abgeschlossener Solr-Kern mit individuellen Definitionen der Konfiguration und des Schemas (vgl. Abschnitt 4.2.2) angelegt, wobei über das Admin Interface gleichzeitiger Zugriff auf alle Solr-Kerne möglich ist. Die Konfiguration von den Indizes erfolgt auf Dateiebene. Für den Export der Applikationsdaten wird intern je nach verwendeter Methode ein *UpdateHandler* verwendet. Für die Ausführung der Suche werden die Suchanfragen der Nutzer durch den Parser aufbereitet. Danach werden die Information analysiert und zum Index hinzugefügt bzw. mit dem Index verglichen. Das Suchergebnis wird durch den *ResponseWriter* formatiert und über den Import dem Applikationsserver bereitgestellt.

4.2.4.1 Index

Wie bereits in der Abbildung 4.6 verdeutlicht, werden in Solr die Dokumente als Lucene-Indexe verwaltet. In der Abbildung 4.7 ist der strukturelle Aufbau eines Lucene-Indexes schematisch dargestellt und wird im Folgenden gemäß [Ind11] erläutert.

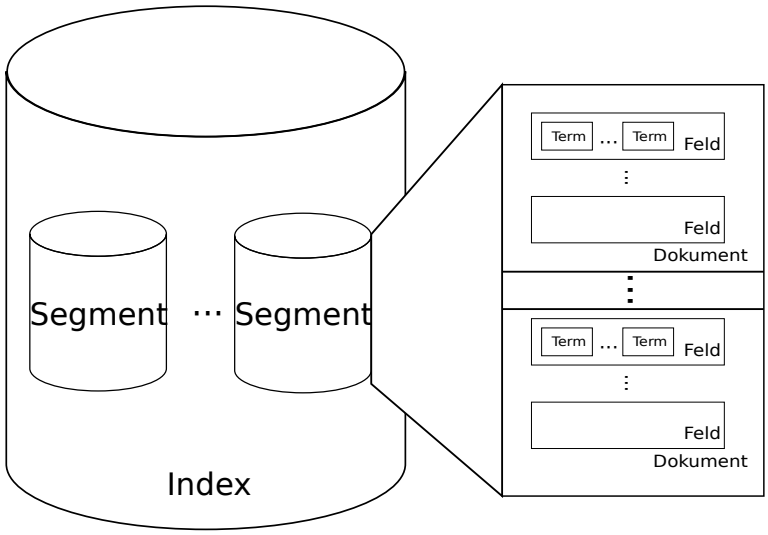

Abbildung 4.7: Der strukturelle Aufbau eines Lucene-Indexes

Index
Der *Index* stellt die oberste Struktureinheit dar und wird als eine invertierte Liste (vgl. Abschnitt 2.2.2) gespeichert.

Segment
Ein Index ist i. d. R. in Subindizes, die sogenannten *Segmente*, unterteilt, die eigenständig durchsucht werden können. Diese Unterteilung in Segmente hat den Hintergrund die Änderungsoperationen auf dem Index als Teilaktualisierungen umzusetzen. Gemäß [Ing07a] werden von Lucene folgende Änderungsoperationen auf dem Index unterstützt, welche über die Export-Schnittstelle an den Solr Server kommuniziert werden.

- ***add/update*** – Ein Dokument wird aktualisiert bzw. in den Index aufgenommen. Intern wird entweder ein Segment neu erstellt oder ein vorhandenes erweitert.
- ***commit*** – Alle Änderungen werden für den Durchsuchungsprozess freigegeben.
- ***optimize*** – Der Index wird optimiert, indem einzelne Segmente zu einem Index zusammengeführt werden.
- ***delete*** – Ausgewählte Dokumente werden aus dem Index entfernt. Diese Operation kann entweder gezielt über die DokumentenID oder über eine Suchanfrage veranlasst werde.

Dokument
Ein Index (bzw. ein Segment) ist aus einer Sequenz von *Dokumenten* aufgebaut, welche nach dem Indizierungsprozess als eine Menge von *Termen* im Sinne des Abschnittes 2.2.1 repräsentiert sind.

Feld
Die Terme des Dokumentes werden durch die *Felder* kategorisiert, welche im Schema (vgl. Abschnitt 4.2.2) erfasst sind. Die einzelnen Felder besitzen einen Datentyp und werden über die Eigenschaften *stored* und *indexed* bezüglich der Indizierung charakterisiert. [Ing07a]

- *stored* – Das Feld wird in der Form wie es dem Dokument vorkommt in den Index aufgenommen. Solche Felder werden z. B. für die Hervorhebung der Suchanfragen in den Suchergebnissen verwendet.
- *indexed* – Das Feld kann durchsucht, sortiert und durch Analysefunktionen verändert werden. Die Analysefunktionen können die Felder in Ausdrücke zerlegen und darauf Methoden als Aufbereitungsschritte vor der eigentlichen Indizierung ausführen. Zum Beispiel wird der Inhalt eines Feldes nach Leerzeichen in Ausdrücke zerlegt, welche im zweiten Schritt in Kleinbuchstaben umgewandelt und anschließend in den Index aufgenommen werden.

4.2.4.2 Schnittstellen

Für die Integration von Solr in eine bestehende Applikation müssen die Schnittstellen für Den Export, Import und die Übermittlung der Suchanfrage implementiert werden. Die Suchplattform bietet mehrere Schnittstellen an, um den Datenaustausch zwischen der vorliegenden Anwendung und dem Solr Server zu ermöglichen. Die Vorstellung der verfügbaren Schnittstellen erfolgt im Folgenden auf Basis von [Sol09].

Export
Bei dem Export, welcher bei der Erstellung bzw. Aktualisierung der Indizes erfolgt, werden die Daten aus der Applikation an den Solr Server versendet. Solr akzeptiert verschiedene Datenformate, welche als Grundlage für die Indizierung verwendet werden können. Der Datenexport erfolgt i. d. R. mithilfe der folgenden drei Möglichkeiten. Mit dem *Solr Cell*-Framework von Apache Tika kann der Export von Inhalten aus strukturierten Dateien mit den Formaten wie z. B. Portable Document Format (PDF), CSV oder Microsoft Word bewerkstelligt werden. Weiterhin ist es möglich Die Daten über HTTP-Requests an den Solr

Server zu übermitteln. Diese Methode ist die typische Herangehensweise für den Transport von Dokumenten, welche im XML-Format vorliegen. Schließlich wird die Übertragung der zu indizierenden Daten über die Java-Schnittstelle von Solr unterstützt.

Suchanfrage
Die Suchanfragen der Nutzer werden als Parameter in der URL an den Solr Server kommuniziert. Intern wird der Suchbegriff durch den *Parser* in ein *Query*-Objekt überführt, welches anschließend für die Suche im Lucene-Index verwendet wird. Für die Übersetzung der im Abschnitt 4.2.4 beschriebenen arten von Anfragen werden in Solr zwei Parser bereitgestellt. Daneben gibt es eine Schnittstelle für die Integration eigener Parser, sodass die Definition eigener Anfragesyntax möglich ist.

Import
Für den Import von Suchergebnissen bietet Solr unterschiedliche Methoden an, die intern durch den entsprechenden *ResponseWriter* realisiert sind. Welcher *ResponseWriter* für den Import von Suchergebnissen verwendet wird, kann parametrisiert in der URL übergeben oder in der Konfigurationsdatei des Solr-Kerns (vgl. Abschnitt 4.2.4) definiert werden.

- *XML*
  Standardmäßig wird in Solr XML als das Format für den Import von Suchergebnissen verwendet. Der Einsatz von XML erfordert die Verwendung von dem *XMLResponseWriter*. Über den *XSLTResponseWriter* kann Extensible Stylesheet Language Transformation (XSLT) verwendet werden, so dass die Ergebnisse formatiert beispielsweise als Really Simple Syndication (RSS)-Feed oder in anderen textbasierten Formaten wie z. B. HTML ausgegeben werden können.

- *JSON*
  Um die Suchergebnisse des Solr Servers als JSON-Objekte zu importieren, kann der *JsonResponseWriter* eingesetzt werden. Dieser bietet ebenfalls die Grundlage für die Ausgabeformatierung der Programmiersprachen *Python* und *Ruby*, welche in den Klassen *PythonResponseWriter* und *RubyResponseWriter* implementiert sind. Dazu werden bestimmte Notationen für die Python- und Ruby-Interpreter vorgenommen. Beispielsweise wird *null* aus der Notation von JSON zu *None* in Python- bzw. *nil* in Ruby-Notation umgewandelt.

- *PHP*
  Die Notation der Skriptsprache PHP wird von Solr ebenfalls als eine Möglichkeit für den Import des Suchergebnisses unterstützt, welche in der Klasse *PHPResponseWriter* von Lucene implementiert ist. Die Ausgabe wird als ein PHP-Array für den Import in die Applikation bereitgestellt.

- *Java*
  Die Möglichkeit die Suchergebnisse in der Notation der Programmiersprache Java zu erhalten, wird durch den *BinaryResponseWriter* umgesetzt.

4.2.4.3 Cluster-Unterstützung

In der Abbildung 4.6 wurde aufgrund der vereinfachten Darstellung die Komponente für die Replikation des Solr Servers nicht dargestellt, welche denn Aufbau eines Clusters aus mehreren Solr Servern ermöglicht. Die Replikationskomponente bietet Mechanismen für

den Aufbau von replizierten und verteilten Server-Umgebungen, welche die Skalierbarkeit der Suchplattform Solr gewährleisten. Wie die Replikation und die Verteilung umgesetzt werden, wird im Folgenden auf Basis von [Mil03] und [Ing07b] beschrieben.

Replikation
Die Methode der Replikation findet Anwendung, wenn z. B. die Verfügbarkeit des Suchservers gesteigert werden soll oder aufgrund des erhöhten Anfragevolumens die physischen Ressourcen einer Maschine für die parallele Verarbeitung der Suchanfragen nicht den gestellten Performance-Ansprüchen ausreichen.

Bei der Replikation wird ein Server zum Master erklärt, welcher mit mindestens einem i. d. R. jedoch mit mehreren Slaves kommuniziert. Der Master verwaltet die Indizes und stellt ihre Kopien als Momentaufnahmen (*Snapshots*) den Slaves bereit. Die Slaves führen die vom Master kommenden Suchanfragen aus und liefern die Suchergebnisse zurück.

Bei jeder Änderungen der Indizes wie z. B. die *commit*- und *optimize*-Operationen (vgl. Abschnitt 4.2.4.1) wird von dem Master-Server ein Snapshot erzeugt. Die Slaves prüfen in gewissen Zeitabständen, ob eine neuere Version auf dem Master verfügbar ist und synchronisieren sich gegebenenfalls.

Für die Serverlastverteilung der Nutzeranfragen kann genauso wie im Abschnitt 4.1.4.3 bereits beschrieben, vorgegangen werden.

Verteilung
Der Anwendungsfall für dieses Verfahren ist ein Index, welcher für einen einzelnen Suchserver zu groß ist. Das heißt Solr kann die Suchprozesse auf so einem Index nicht mehr im Einklang mit den gestellten Anforderungen an die Performance ausführen oder die physische Speichermengenkapazität des Servers überschritten wird.

Bei der Verteilung wird ein Index über mehrere Server, auch als *Shards* bezeichnet, in Teile zerlegt und verschiedenen Shards zugewiesen.

Eine Suchanfrage wird immer an einen beliebigen Shard gestellt, von dem die Anfrage an restliche Shards für die parallele Abarbeitung weitergeleitet wird. Die Suchergebnisse einzelner Server werden an den Shard, der die Suchanfrage verteilt hat, zurückgeliefert. Anschließend führt dieser Shard die einzelnen Teile zu einem Gesamtergebnis zusammen.

Die mögliche anfängliche Verteilung der Dokumente kann zufällig oder über ein Hash-Funktion, welche anhand der DokumentenID einen Server auswählt, erfolgen. Einen konkreten Mechanismus zur gleichmäßigen Verteilung der Dokumente über alle Server stellt Solr nicht bereit.

Bei diesem Verfahren bietet sich ebenfalls eine Serverlastverteilung von Suchanfragen über die einzelnen Shards an.

Kombination aus Verteilung und Replikation
Bei Anwendungsszenarien, wo viele Anfragen auf großen Indizes ausgeführt werden und die Ausfallsicherheit des Solr Servers erhöht werden soll, kann die Kombination aus Replikation und Verteilung Abhilfe schaffen. Der schematische Aufbau einer derartigen Kombination ist in der Abbildung 4.8 veranschaulicht.

Im Vergleich zu der Verteilung wird bei dieser Methode jeder Shard auf mehrere Slaves repliziert, sodass die Shardmaster die Verarbeitung von Suchanfragen auf die Slaves verlagern und ausschließlich für die Aktualisierung von Indexpartitionen verantwortlich sind. Die einzelnen Slaves kommunizieren miteinander und bilden mit den anderen Slaves eines

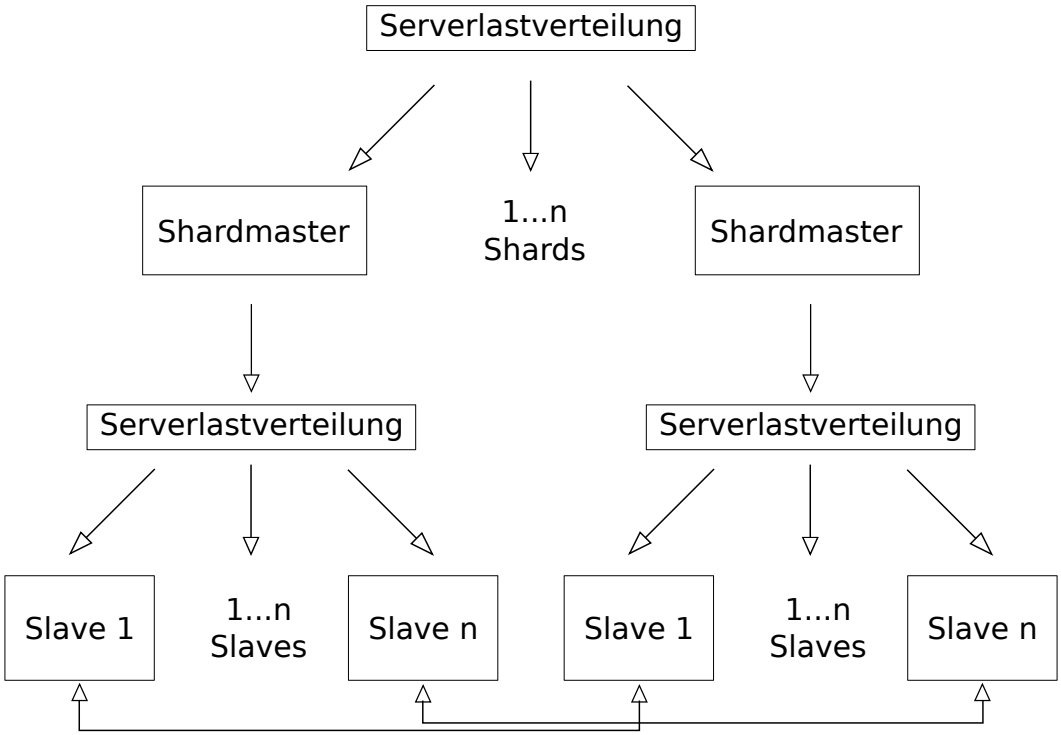

Abbildung 4.8: Kombination aus Verteilung und Replikation aus [Ing08]

Shards eine Slave-Farm. Das heißt, dass die Anzahl an verfügbaren Slave-Farms gleich der Anzahl der Slaves eines Shards ist.

Die Suchanfragen werden an einen beliebigen Mastershard gestellt, welcher diese wiederum an irgendeinen Slave weiterleitet. Danach stellt der Slave-Server die Suchanfrage an die anderen Server der zugehörigen Slave-Farm, führt nach der Ausführung der Suche die Teile zusammen und liefert das Gesamtergebnis an den Mastershard zurück.

Bei dieser Methode bietet sich der Einsatz der Serverlastverteilung bei der Auswahl des Shardmasters und des zugehörigen Slaves an.

Durch die Kombination von Replikation und Verteilung wird ein ausfallsicheres System aufgebaut. Fällt ein Slave-Server aus, kann der zugehörige Shardmaster die Anfrage auf einem anderen Slave ausführen. Bei Ausfall eines Shardmasters kann die Suche ebenfalls ausgeführt werden, jedoch gehen die Suchanfragen des ausgefallenen Mastershards verloren.

4.3 Gegenüberstellung

In diesem Abschnitt werden die Suchplattformen FACT-Finder und Solr anhand der Anforderungen, welche im Kapitel 3 aufgestellt wurden, gegenübergestellt.

4.3.1 Shop-Betreiber

Kosten
Alle Faktoren, welche Kosten verursachen und im Abschnitt 3.1.1 aufgezählt wurden, können im Rahmen diese Arbeit nicht betrachtet werden. Dazu zählen z. B. die langfristige Instandhaltung sowie der Aufwand für die Erweiterung und Aktualisierung der Suchplattform. Das bedeutet, dass an dieser Stelle primär die anfänglichen Investitionen betrachtet werden. Da der Integrationsaufwand ausschließlich aus technischer Sichtweise betrachtet wird (vgl. Abschnitt 3.1.2), ist die nicht kommerzielle Suchplattform Solr FACT-Finder vorzuziehen.

Preismodell
Das Preismodell ist für den Betreiber ein wichtiges Kriterium. Insbesondere für kleine und mittlere Unternehmen ist eine flexible Bezahlung wichtig. Diese Anforderung kann ausschließlich für FACT-Finder betrachtet werden, weil Solr in der Standardversion frei verfügbar ist. FACT-Finder bietet auf diesem Gebiet eine Komplettlösung und eine SaaS-Variante, wobei Omikron als ASP auftritt. Auf diese Weise deckt Omikron die Wünsche von Großunternehmen ab, welche eine im Bezug auf Datensicherheit unkritische Lösung wollen, und ermöglicht mit dem SaaS-Modell kleineren Unternehmen, den Service ohne große Vorinvestitionen in vollem Ausmaß einzusetzen. Zusätzlich zu diesen technischen Preisbildungen können die Nutzer zwischen den einzelnen Funktionspaketen (vgl. Abbildung 4.1) auswählen und somit den Preis präziser auf die Shop-Anforderungen anpassen. Somit wird den Kunden, welche den FACT-Finder einsetzen wollen, ein flexibles und modulares Preismodell angeboten.

Integrationsaufwand
Der Integrationsaufwand impliziert die fachlichen Kenntnisse, den zeitlichen Aspekt sowie den Umfang an erforderlichen Implementierungen, welche für die Integration der Suchplattform in eine Anwendung erforderlich sind. Sowohl bei FACT-Finder als auch bei Solr ist die Integration mit der Implementierung der Schnittstellen für den Export der Daten an die Suchmaschine und den Import der Suchergebnisse in die Anwendung sowie mit der Konfiguration der Suchplattform und der Indizes verbunden. Die Bewertung des technischen Aspekts dieser Anforderung kann erst nach der praktischen Umsetzung, welche für ein konkreten Beispiel im Abschnitt 5.2.2 beschrieben wird, vergeben werden.

Konfiguration
Die Konfiguration der Suchplattform kann in die Bereiche Index- und Servereinstellungen unterteilt werden. Für die Konfiguration der Indizes bieten sowohl FACT-Finder als auch Solr die Möglichkeit Thesauren zu erstellen, Stoppwörter zu definieren. Bei FACT-Finder kann eine Abwertung für ausgewählte Attribute von Dokumenten vorgenommen werden, welche für die Suchergebnisplatzierungen einbezogen wird. Dadurch kann z. B. dem Namen mehr Bedeutung als der Beschreibung eines Artikels zugewiesen werden, sodass diese Einstellung für die gezielte Manipulation der Dokumentenplatzierungen verwendet werden kann. In Solr kann die Dokumentenplatzierung durch Parameter in der Suchanfrage ebenfalls vorgenommen werden, jedoch gibt es dafür keine Konfigurationseinstellung. Bei beiden Produkten hat man die Möglichkeit die Datentypen für die einzelnen Indexattribute auszuwählen. Weiterhin besteht bei Solr die Möglichkeit spezielle für eigene Anwendung zugeschnittene Datentypen durch Kombination ausgewählter Analyse- und Filterfunktionen zu definieren, was für die Feineinstellung der Indizes hohen Stellenwert hat. Die Einstellungen, welche für die Konfiguration des Servers wichtig sind wie z. B.

Caching oder verwendetes Ausgabeformat für die Suchergebnisse (Import-Schnittstelle), lassen sich nur bei Solr direkt einstellen. Solr bietet somit mehr bzw. feingranularere Konfigurationsmöglichkeiten, was die Feineinstellung des Indexes betrifft. Die Mächtigkeit bringt jedoch die Gefahr mit sich falsche Einstellungen zu treffen, welche sich auf die Performance fatal auswirken können.

Skalierbarkeit
Die Skalierbarkeit der Suchplattform ist bei Systemen mit übermäßiger Ressourcenauslastung unumgänglich. Dabei sind Mechanismen für die Fälle erforderlich, bei denen das Anfrage-Volumen und/oder die Größe eines Indexes die Performance eines einzelnen Suchservers dramatisch verschlechtern, erforderlich. Für den ersten Anwendungsfall sind die Vorgehensweisen beider Suchplattformen identisch. FACT-Finder und Solr bieten bis auf kleinere Unterschiede den Aufbau eines Clustersystems an, wobei eine Indexkopie über mehrere Suchserver repliziert wird. Auf diese Weise wird die Last über die Such-Server des Clusters verteilt werden. Der Unterschied liegt in der Umsetzung der Synchronisation zwischen dem Master und den Slaves. Bei FACT-Finder kann durch den Einsatz von NFS die Synchronisation auf Kosten von Rechenleistung vereinfacht werden. Bei Solr kann dagegen bei der Methode mit den Momentaufnahmen des Indexes zu inkonsistenten Zuständen auf einzelnen Slaves kommen, was zur Folge verfälschte Suchergebnisse haben kann. Bei dem Problem eines zu großen Indexes wird bei FACT-Finder keine konkrete Lösung vorgeschlagen. In diesem Zusammenhang wurde nur auf die Erstellung eines kompakteren Indexes angedeutet. Solr bietet dagegen eine konkrete Lösung über die Partitionierung und Verteilung des Indexes über mehrere Server. Des weiteren lassen sich die Vorteile von der Replikation und Verteilung durch die Kombination dieser Methoden vereinigen. Somit stellt Soll im Vergleich zu FACT-Finder ein breiteres Spektrum an Lösungsmöglichkeiten um die Skalierbarkeit der Suchplattform zu erhöhen.

Ausfallsicherheit
Aufgrund der Möglichkeit die Replikation und Verteilung bei Solr kann die Suchplattform bei Ausfällen von Master- und Slave-Servern bei weiterhin die Suchanfragen bearbeiten. Die Cluster-Unterstützung von FACT-Finder kann dagegen mit Ausfällen von Master-Servern nicht umgehen. Somit wird bei Solr mehr Ausfallsicherheit als bei der Suchplattform FACT-Finder geboten.

Schnittstellen
Teilweise bieten die Suchplattformen die selben Schnittstellen wie z. B. CSV für den Export oder JSON und XML für den Import von Daten an. Im Gesamtergebnis ist jedoch das Angebot an unterstützten Methoden und Programmiertechniken bei der Suchplattform Solr breiter gefächert als bei FACT-Finder gefächert. Des weiteren bietet Solr durch Möglichkeiten zur Definition eigener Anfragesprachen, sodass bei dieser Anforderung Solr der Suchplattform FACT-Finder vorzuziehen ist.

Unterstützung der Hardware
Bei FACT-Finder wird der Einsatz von 64-Bit Architekturen und parallele Ausführung von Suchanfragen auf unterschiedlichen Prozessorkernen unterstützt [Doc11]. Für Solr werden diesbezüglich keine Angaben gemacht, sodass keine Aussage gemacht werden kann, ob diese Funktionalität unterstützt wird. Aus diesem Grund ist FACT-Finder für dieses Kriterium vorzuziehen.

Erweiterbarkeit
FACT-Finder ist modular aufgebaut und lässt sich durch das Einspielen der richtigen

Version als eine gepackte Applikationsdatei auf unkomplizierte Art erweitern. Das selbe gilt für die Suchplattform Solr, sodass für die Anforderung die Produkte gleich zu bewerten sind.

Analysewerkzeuge
Solr bietet mit dem Admin Interface sehr nützliche Analysewerkzeuge was die Performance der Suche und der einzelnen Schritte bei der Auswertung der Terme während des Suchvorgangs betrifft. Bei FACT-Finder ist die Analyse des Suchvorgangs nicht in dem Detail möglich, jedoch kann weiterhin das Suchverhaltens der Kunden analysiert werden. Somit erfüllt FACT-Finder die gestellte Anforderung im höherem Maße als die Suchplattform Solr, weil hier insbesondere ein wichtiger Aspekt für den Bereich E-Commerce analysiert wird.

Features
Beide Suchplattformen unterstützen solche Features wie Autovervollständigung und Kategorisierung von Suchergebnissen welche insbesondere Für den Shop-Nutzer sehr wichtig sind. Daneben hat FACT-Finder sehr nützliche Features wie die Erstellung von Kampagnen und Abbildung der gesamten Shop-Navigation. Folglich hat FACT-Finder mehr nützliche Features als Solr und wird aus diesem Grund für die Anforderung an Features priorisiert.

Sprachunabhängigkeit
Bei Solr kann durch die Verwendung von speziellen Filtern die Dokumente aus verschiedenen Sprachräumen analysiert werden, sodass die Suche u. a. für russische, arabische sowie chinesische Sprachen unterstützt wird. FACT-Finder bietet für diese Anforderung ebenfalls verschiedene Sprachen an. Somit wird von beiden Suchplattformen diese Anforderung gleich gut unterstützt.

4.3.2 Shop-Nutzer

Autovervollständigung und Kategorisierung von Suchergebnissen
Bei der Anforderung an die Repräsentation von Ergebnissen und Interaktion mit den Nutzer schneiden beide Suchplattformen gleich gut ab. FACT-Finder und Solr bieten wichtige Funktionen wie die After-Search-Navigation und Auto-Suggest.

Benutzbarkeit
Die Benutzbarkeit aus der Sicht des Shop-Nutzers ist für beide Suchplattformen gleich gut. Der Nutzer wird nicht mit übermäßigen Einstellungen überfordert, sondern bei der Sucheingabe und Auswertung durch Auto-Suggest und ASN unterstützt.

Fehlertoleranz
Beide Suchplattformen beherrschen die fehlertolerante Suche durch die phonetische Analyse der Dokumente und durch den Einsatz von fehlertoleranten Information Retrieval Modellen. Die Fehlertoleranz lässt sich in beiden Produkten durch die Einstellungen, welche in der Anforderung an die Auswahl an Algorithmen beschrieben wurden, konfigurieren. Somit erfüllen beide Suchplattformen die gestellte Anforderung.

Performance
Die Anforderung an die Performance kann erst nach den entsprechenden Performance-Tests bewertet werden. Das wird im Rahmen dieser Arbeit im Kapitel 5 für eine konkrete Testumgebung durchgeführt, sodass an dieser Stelle für die Auswertung und Gegenüber-

stellung der Suchplattformen bezüglich dieser Anforderung auf den Abschnitt 5.3 verwiesen wird.

Soziale Suche
Die Anforderung an die soziale Suche impliziert die Vorschläge und die Sortierung, welche auf den Verkaufsstatistiken und Suchverhalten basieren. Bei FACT-Finder wird dieser Aspekt teilweise durch die *Recomendation Engine* (vgl. Abschnitt 4.1.2) abgedeckt. Damit kann FACT-Finder die Suchergebnisplatzierungen durch den Einbezug des Nutzerverhalten beeinflussen, sodass z. B. oft gekaufte Produkte schrittweise automatisch aufgewertet werden. Bei Solr lässt sich für die Analyse des Suchverhaltens ein Modul von *LucidWorks Enterprice* einbinden, welches das Suchverhalten der Nutzer analysiert und auswertet [Str10]. Bei beiden Suchplattformen handelt es nicht um Standardmodule, welche gegen einen Aufpreis erworben werden können.

4.3.3 Zusammenfassung

Die Tabelle 4.2 fasst die Ergebnisse der Gegenüberstellung nochmal zusammen. Die Gegenüberstellung einzelner Anforderungen werden durch die Vergleichsoperatoren $<$, $>$ und $=$ dargestellt, sodass die Bewertungen ohne einen Referenzwert veranschaulicht werden.

Kategorie	**Anforderung**	**FACT-Finder**	**Solr**
Betreiber wirtschaftlich	Kosten	$<$	
Betreiber technisch	Konfiguration	$<$	
	Skalierbarkeit	$<$	
	Ausfallsicherheit	$<$	
	Schnittstellen	$<$	
	HW-Unterstützung	$>$	
	Erweiterbarkeit	$=$	
	Sprachunabhängigkeit	$=$	
Betreiber funktional	Analysewerkzeuge	$>$	
	Features	$>$	
Nutzer Eingabe	Auto-Suggest	$=$	
	Benutzbarkeit	$=$	
Nutzer Suche	Fehlertoleranz	$=$	
Nutzer Repräsentation	ASN	$=$	
	Soziale Suche	$=$	

Tabelle 4.2: Zusammenfassung der Gegenüberstellung

Die Wertung der hier aufgeführten Betrachtungen stammen aus den Dokumentationen sowie der Recherche, welche im Vorfeld in den Abschnitten 4.1 und 4.2 für ausgewählte Themengebiete erfasst ist. Hierbei bestand die Hauptschwierigkeit darin, dass es zu FACT-Finder kaum Informationen zu internen Abläufen gibt, was im Ergebnis des theoretischen Vergleichs zur folgender Wertung geführt hat.

Solr bietet eine ausgereifte Lösung, jedoch fehlt es an der Umsetzung einer intuitiven Administrationsoberfläche für die Konfiguration der Suchplattform. Insbesondere für den Bereich E-Commerce kann FACT-Finder mit spezifischen Einstellungsmöglichkeiten und

Features gegenüber Solr punkten. Somit empfiehlt sich der Einsatz von der Suchplattform FACT-Finder für professionelle Shops, welche einen schnellen Einstieg sowie eine einfache und unkomplizierte Konfigurationsoberfläche und Services wie SaaS und Support erfordern. Dagegen ist die Suchplattform Solr für den Einsatz in Web-Shop geeignet, welche keine finanziellen Investition machen wollen und selbst die Feinkonfiguration der Indizes und der Suchplattform bis auf kleinste Details bestimmen möchten. Die Gegenüberstellung bezüglich des Integrationsaufwandes, der Performance sowie der Qualität der Suche erfolgt im nächsten Kapitel für einen konkreten Anwendungsfall.

Kapitel 5

Integration in SCOOBOX

Die Firma dotSource GmbH stellt einen Demo-Shop *Outbacker*[1] zur Verfügung, welcher die Social Commerce-Funktionen von SCOOBOX (vgl. Abschnitt 2.3.2) veranschaulicht. Der Shop ist über das Internet und ohne Registrierung erreichbar. Über den Verweis *Outbacker Blog* auf der Startseite von Outbacker kann der Nutzer zum Blog dieses Online-Shops, welcher die Grundlage für die Integration der Suchfunktionalität bildet, weitergeleitet werden.

In diesem Kapitel wird eine exemplarische Integration der Suchplattformen in den Outbacker Blog beschrieben, welche durch die folgenden Ziele motiviert wurde. Die Implementierung bietet eine Grundlage für den technischen Vergleich auf Basis eines Referenzsystems, wobei speziell der Integrationsaufwand und die Performance der Suche für einen konkreten Anwendungsfall geschätzt werden. Des weiteren bestand aus der Sicht der Firma dotSource GmbH ein Interesse an der Machbarkeit dieser Implementierung.

Nach der einleitenden Vorstellung der Ausgangssituation und des Ziels wird im Abschnitt 5.2 die Integration von FACT-Finder und Solr in den Outbacker Blog beschrieben. Hierbei werden die vorhandenen Schnittstellen und für die Integration notwendige Anpassungen und Erweiterungen vorgestellt und der Integrationsaufwand verglichen. Anschließend wird im Abschnitt 5.3 die Performance sowie Qualität der Suche und Indizierung anhand von Performance-Tests verglichen und bewertet.

5.1 Ausgangssituation und Ziel

Derzeit ist auf der Seite des Blogs ein Sucheingabefeld zur Verfügung gestellt, über das der Benutzer eine Suche ausführen kann. Hierfür wird im Endeffekt eine Datenbankanfrage ausgeführt und die erhaltenen Suchergebnisse aufbereitet und dem Benutzer auf der Seite präsentiert. Die Seite mit den Suchergebnissen besteht aus einem Seitenkopf, Seitenrand und den eigentlichen Blog-Einträgen, welche bei der Suche gefunden wurden. Falls die Suche ein leeres Ergebnis zurückliefert, wird dem Benutzer eine entsprechende Meldung angezeigt.

Nach der Integration der Suchplattformen soll der Ablauf nach der Eingabe des Suchbegriffs wie folgt ablaufen. Der Applikationsserver, auf dem der Onlineshop betrieben

[1] `http://facebook-scoobox-demostore.dotsource.de`

wird, kommuniziert mit dem jeweiligen Suchserver über das Hypertext-Übertragungsprotokoll um entsprechende Inhalte finden zu können. Der Suchserver ermittelt anhand der vorliegenden Informationen eine relevante Suchergebnismenge und sendet diese an den Applikationsserver zurück. Nach Aufbereitung der Informationen werden diese auf der Ergebnisseite präsentiert. Hierbei kann die Ergebnisseite bis auf kleinere Anpassungen wiederverwendet werden.

5.2 Implementierung

Wie bereits im Abschnitt 2.3.2 erläutert wurde, ist *SCOOBOX* eine mit Enfinity Suite 6 entwickelte WEB-Applikation, welche u. a. für das Erstellen von Blogs eingesetzt werden kann. Der Demo-Shop *Outbacker* ist eine Instanz dieser Web-Anwendung, welche die Funktionen von SCOOBOX, darunter den *Outbacker Blog*, demonstriert. Bei der im Folgenden beschriebenen Implementierung werden die Suchplattformen FACT-Finder und Solr in den Outbacker Blog integriert, um dessen Inhalte mit fehlertoleranter Volltextsuche zu erweitern.

5.2.1 Vorhandene Funktionalität

Die zur Entwicklung eingesetzte Version 6.4 der Intershop Enfinity Suite (vgl. Abschnitt 2.3.1) realisiert im Standardpaket die Suchfunktion für die Produktsuche mithilfe externer Komponenten. Die Integration der Suchplattformen basiert auf den generischen Klassen des Funktionsmoduls (Cartridge, vgl. Abschnitt 2.3.1) ***bc_foundation***, welche die Grundfunktionalität für den *Index*, die *Indexkonfiguration* sowie den *Export* und *Import* von Applikationsdaten implementiert. Zur Zeit werden die Suchmaschinen FACT-Finder und Apache Solr für die Produkt- und Contentsuche unterstützt, welche über die Cartridges ***ac_search_factfinder*** und ***ac_search_solr*** eingebunden werden können [E64F10]. Diese zwei Module implementieren die Schnittstellen für die Kommunikation zwischen dem Applikationsserver der Enfinity Suite und FACT-Finder- bzw. Solr-Suchserver. Somit kann die vorhandene Funktionalität dieser Cartridges für die Integration der Suchplattformen in den Outbacker Blog wieder verwendet werden, sodass sich der praktische Teil dieser Arbeit auf die Erstellung eines Blog-spezifischen Indextypen und die Implementierung der Verarbeitungslogik (Pipeline, vgl. Abschnitt 2.3.1) für die Darstellung der Suchergebnisse beschränkt. Da die vorhandenen Cartridges die Grundlage für die Integration darstellen, sind diese in der Tabelle 5.1 zusammengefasst und werden im Folgenden gemäß [ESSe10] vorgestellt.

Index
Das Index-Objekt repräsentiert die Datei bzw. die Dateien, welche persistent auf dem Speichermedium vorliegen. Dieses Objekt stellt Methoden für die Verarbeitung von Suchanfragen sowie Routinen für die Verwaltung der Indizes zur Verfügung. Wie z. B. Beispiel das Laden der Indexdaten bzw. Freigabe von dem verwendeten Arbeitsspeicher, wenn der Index nicht mehr gebraucht wird. Für jeden Index wird eine eigene Instanz des Index-Objekts erzeugt, welche durch die spezielle Index-Manager-Klasse im Arbeitsspeicher des Applikationsservers gehalten wird. Die FACT-Finder- und Solr-Index-Implementierungen erben die Grundfunktionalität aus *SearchIndexBaseImpl* und erweitern diese Klasse um die jeweiligen Server-spezifischen Methoden.

Cartridge	Klasse	Funktion
bc_foundation	SearchIndexBaseImpl	Index
	SearchIndexConfigurationImpl	Konfiguration
	AbstractSearchIndexImportHandler	Import
ac_search_factfinder	FactfinderIndex	Index
	FactfinderIndexConfiguration	Konfiguration
	CSVIndexHandler	Import
	FactfinderResult	Export
ac_search_solr	SolrIndex	Index
	SolrIndexConfiguration	Konfiguration
	SolrIndexHandler	Import
	SolrSearchResultImpl	Export

Tabelle 5.1: Vorhandene Java-Klassen in Enfinity Suite 6 aus [ESSe10]

Konfiguration
Zu jedem Index-Objekt wird eine Konfigurationsdatei angelegt, welche die Einstellungen wie Informationen über die indizierten Attribute, den verwendeten Indextyp oder die Lokalisation der Indexdaten speichert. Bei der Erstellung eines neuen Indexes wird die globale Konfigurationsdatei initial für das Index-Objekt verwendet, von der eine Kopie in dem Index-Ordner angelegt wird. Werden die Einstellungen des Indexes verändert, so wird diese Datei entsprechend angepasst und bei der Aktualisierung als Referenz verwendet. Während bei FACT-Finder die Konfiguration in einer Datei erfasst wird, legt die Klasse *SolrIndexConfiguration* die Konfiguration in einem Unterordner ab, wobei zusätzliche Informationen wie z. B. die Liste der Synonyme und Stoppwörter als separate Dateien abgespeichert werden.

Für den Export- und Import-Vorgang ist die Namensgebung der Klassen aus der Sicht des Suchservers entstanden, was in den Abschnitten 4.1.4 und 4.2.4 von der Web-Applikation aus betrachtet wurde. Im Folgenden soll die Namensgebung der Klassen beibehalten werden und die Prozesse aus der Perspektive des Suchservers betrachtet werden.

Import
Für den Import von den zu indizierenden Daten gibt es eine für den jeweiligen Server spezifische Klasse, von der eine Instanz bei jeder Indexerstellung und -aktualisierung erstellt wird.

Die Klasse *AbstractSearchIndexImportHandler* instantiiert und nutzt die sogenannten *DataProvider*-Klassen, welche die zu indizierende Information aufbereiten. Die Klassen *CSVIndexHandler* und *SolrIndexHandler* implementieren die abstrakten Methoden und programmieren die Suchserver-spezifischen Schnittstellen für den Datenimport aus.

Suchergebnis
Diese Klassen sind für die Aufbereitung der Suchergebnisse zuständig. Damit wird der Export der Suchergebnisse von dem Suchserver an den Applikationsserver von Enfinity Suite ermöglicht.

5.2.2 Suchfunktion im Blog

Wie Im Abschnitt 5.2.1 bereits erwähnt, erfordert die Integration der Suchplattformen neue Indextypen sowie Verarbeitungslogik für die Darstellung der Suchergebnisse. Die Beschreibung der hierfür notwendigen Schritte wird im Folgenden vorgestellt.

Indextypen
Für die Implementierung neuer Indextypen wurde für jeden Suchserver ein eigenes Funktionsmodul mit den Namen ***ac_sc_search_factfinder*** und ***ac_sc_search_solr*** angelegt. Um die vorhandene Funktionalität wiederzuverwenden, müssen die notwendigen Cartridges in den Abhängigkeitsbaum des neuen Moduls aufgenommen werden. In der Abbildung 5.1 sind die direkten Abhängigkeiten von ***ac_sc_search_factfinder*** beispielhaft dargestellt.

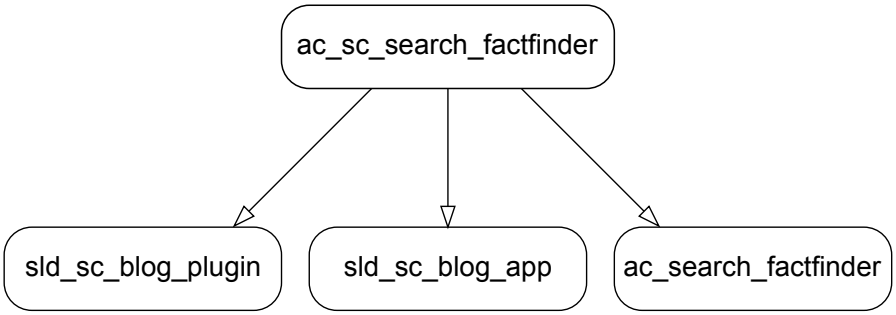

Abbildung 5.1: Abhängigkeitsbaum von ac_sc_search_factfinder

Um einen Indextyp bei dem Applikationsserver von Enfinity Suite zu registrieren, muss unter dem Pfad *<cartridge>/javasource/resources/<cartridge>/searchindex* die Datei *searchindexfeatures.properties* angelegt werden. Diese Datei enthält den Namen und die Klasse des neuen Indextyps und wurde sowohl für den FACT-Finder- als auch für den Solr-Index erstellt. Die Cartridge ***bc_foundation*** enthält den Index-Manager, welcher für die Registrierung der Indextypen verantwortlich. Aus diesem Grund müssen die Cartridges ***ac_sc_search_factfinder*** und ***ac_sc_search_solr*** bei dem Serverstar nach der ***bc_foundation*** geladen werden.

Weiterhin muss ebenfalls die Definition des Indextyps vorliegen. Der Name dieser Datei gleicht dem Muster *FeatureID[.EngineID].xml*. FeatureID entspricht dem Namen des Indextyps und die EngineID kennzeichnet die Erweiterung, welche im Fall von mehreren Implementierungen für den gleichen Indextyp angegeben wird. Die Namen der neuen Indextypen können komplett übernehmen werden, sodass die Dateien *SFBlogContentSearch.solr.xml* und *SFBlogContentSearch.factfinder-ws.xml* erstellt wurden. In diesen Dateien werden die Eigenschaften und die statischen Attribute, die für die Indizierung zur Auswahl stehen werden, festgehalten.

In Listing 5.1 ist ein Ausschnitt aus der Konfigurationsdatei veranschaulicht. Diese Datei enthält alle zur Definition eines Indextypen notwendigen Informationen, die der Applikationsserver der Enfinity Suite erfordert. Aufgrund der fundamentalen Bedeutung dieser Konfigurationen werden im Folgenden die einzelnen Merkmale genauer betrachtet. Der Ausschnitt ist der Datei *SFBlogContentSearch.factfinder-ws.xml* entnommen und wird die Grundlage für exemplarische Erläuterung neuer Indextypen genommen.

```
1  <searchIndexFeature>
2    <description>Storefront Blog Content Search Index</description>
3    <searchEngineName>Omikron FACT-Finder Server</searchEngineName>
4
5  <!-- a class name definition that defines a handler to write index data -->
6    <importHandlerClass>
7     com.intershop.adapter.search_factfinder.internal.webservice.
          CSVIndexHandler
8    </importHandlerClass>
9
10 <!-- a class name definition that is used
11   to convert the results from the search index result type
12   back to enfinity objects - Pipelet ResolveObjectsFromSearchIndex -->
13   <objectsResultClass>
14    com.dotsource.adapter.sc.internal.ResolveBlogPost
15   </objectsResultClass>
16
17 <!-- query names that are used to retrieve the objects to index -->
18   <objectsToIndexQueryName>
19    blogpost/GetPostsToIndex
20   </objectsToIndexQueryName>
21
22   <dataProviders>
23   <dataProvider name="BlogPost"
24    className =
25    "com.dotsource.adapter.sc.internal.searchindex.BlogPostDataProvider" />
26   </dataProviders>
27
28   <attribute>
29    <description>The blogpost title</description>
30    <displayName>tilte</displayName>
31    <filterAttribute>false</filterAttribute>
32    <name>Title</name>
33    <searchable>true</searchable>
34    <standard>false</standard>
35    <dataProviderName>BlogPost</dataProviderName>
36   </attribute>
37   ...
38
39 </searchIndexFeature>
```

Listing 5.1: Ausschnitt aus SFBlogContentSearch.factfinder-ws.xml

Die Merkmale *description* und *SearchEngineName* (2. und 3. Zeile im Listing 5.1) haben einen beschreibenden Charakter und werden gemäß dieser Definition bei der Auswahl der Indextypen wie in der Abbildung 5.2 angezeigt.

Die restlichen Merkmale werden folgendermaßen verwendet. Bei dem Import der Daten in den Suchserver wird mit der Datenbankabfrage des Merkmals *objectsToIndexQueryName* die Information für die Indizierung geholt. Diese Daten werden von der Klasse, welche innerhalb des Merkmals *dataProvider* definiert ist, für den Import aufbereitet. Anschließend sorgt die Klasse des Merkmals *importHandlerClass* für den Transport von Daten zm Suchserver. Im letzten Schritt bildet die Klasse des Merkmals *objectsResultClass* die Suchergebnisse auf die Objekte der Web-Applikation ab. Das Merkmal *attribute* definiert Standardattribute, welche für die Indizierung im Backend der Anwendung angezeigt wer-

Search Index Types

Shows the available search index types. Enable search index types for this channel and set preferences for indexes.

Index Type ID	Engine	Description
SFBlogContentSearch	Omikron FACT-Finder Server	Storefront Blog Content Search Index
SFContentSearch	Apache Solr Server	Storefront Content Search Index
SFContentSearch	Omikron FACT-Finder Server	Storefront Content Search Index
SFProductSearch	Omikron FACT-Finder	Storefront Product Search Index
SFBlogContentSearch	Apache Solr Server	Storefront Blog Content Search Index

Abbildung 5.2: Indextypen in SCOOBOX

den. Auf die Klassen und die Datenbankabfrage wurden zum Teil neu erstellt und werden im Folgenden genauer betrachtet.

importHandlerClass
Das Merkmal *importHandlerClass* definiert die Klasse, mit welcher der Datenimport auf den Suchserver erfolgt. Die Module ***ac_search_factfinder*** und ***ac_search_solr*** können diese Klassen unverändert übernehmen.

objectsResultClass
Mithilfe des *objectsResultClass*-Merkmals wird die Klasse für die Abbildung der Suchergebnisse auf die Enfinity-Objekte definiert. Das geschieht durch einen Manager, welcher bei der Instanziierung der Klasse geladen wird und die Suchergebnisse anhand eindeutiger Schlüssel auf die Enfinity-Objekte abbildet. Diese Klasse ist nicht an einen Suchserver gebunden und kann aus diesem Grund für beide Indextypen ohne Anpassungen eingesetzt werden.

objectsToIndexQueryName
Es gibt drei unterschiedliche Fälle in den in Abhängigkeit von den Blog-Einstellungen unterschiedliche Einträge in den Index aufgenommen werden können. Für die Fallunterscheidung werden die Objekte *searchIndex* und *Blog* als Eingangsparameter in die Datenbankanfrage definiert (vgl. Abschnitt 2.3.2). Die Datenbankanfragen wurden mithilfe des Query-Frameworks von Enfinity Suite geschrieben, welches XML zur Generierung von SQL-Anfragen einsetzt.

1. Fall (Listing A.1: Zeilen 1-8): Die Sprache der Blog-Einträge gleicht der Standardsprache des Systems, welche in Outbacker Blog auf englisch eingestellt ist. Nach der Definition des Datenschemas (Abbildung 2.5) besitzen solche Blog-Einträge kein Vaterelement.

2. Fall (Listing A.1: Zeilen 9-19): Die Sprache der Blog-Einträge ist ungleich der Sprache des Systems, wobei die Blog-Beiträge mit der Systemsprache nicht angezeigt werden.

3. Fall (Listing A.1: Zeilen 20-44): Die Sprache der Blog-Einträge ist ungleich der Sprache des Systems. Falls kein Blog-Eintrag in der gewünschten Sprache existiert, so wird der entsprechende Eintrag in der Systemsprache angezeigt. Es werden zunächst alle Blog-Einträge zu der gewünschten Sprache sowie der Systemsprache geholt. Im zweiten Schritt werden die Beiträge in der Systemsprache, für die Lokalisierungen existieren, wieder abgezogen.

Die Datenbankabfrage ist nicht für den verwendeten Suchserver spezifisch und kann somit für beide Indextypen unverändert eingesetzt werden. Zur Veranschaulichung der Drei Fälle ist in der Abbildung 5.3 ein Minimalbeispiel dargestellt.

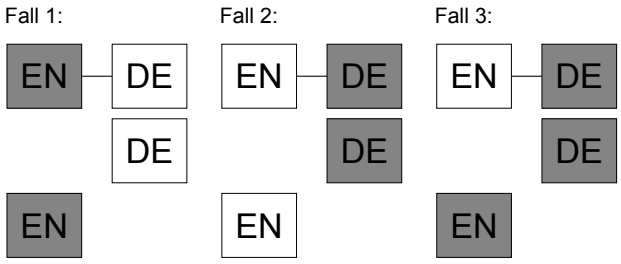

Abbildung 5.3: Fallunterscheidung in *GetBlogPostsToIndex.query*

Insgesamt sind vier Beiträge im Blog von den zwei in englischer (EN) und zwei in deutscher (DE) Sprache verfasst sind, wobei die Systemsprache auf englisch eingestellt ist. Die Verbindungen zwischen einzelnen Beiträgen bedeutet, dass es derselbe Blog-Beitrag in einer anderen Sprache ist. Die in grau dargestellten Beiträge werden von der Datenbankabfrage geholt.

dataProvider
Innerhalb des *dataProviders*-Merkmals werden die *dataProvider* definiert. Dabei muss der Name und die verwendete Java-Klasse als Attribute angegeben werden. Die Klasse *com.dotsource.adapter.sc.internal.searchindex.BlogPostDataProvider* ist mit geringfügigen Anpassungen in beide Indextypen integriert worden.

Die Aufgabe der Java-Klasse besteht in der Aufbereitung der Ergebnisse, die aus der Datenbankabfrage resultieren. In diesem Schritt werden die Werte der zu indizierenden Attribute aus den einzelnen Zeilen extrahiert und in Objekte, entweder für weitere Verarbeitung oder für die Ausführung parametrisierter Datenbankanfragen, umgewandelt. Über den Namen des *dataProvider* greifen die statischen Attribute, welche ebenfalls in der Konfigurationsdatei des Indextypen definiert werden, auf die Klasse zu.

attribute
In der Abbildung 5.4 sind alle statischen Attribute aus dem Bereich der Indexkonfiguration dargestellt, welche in den Konfigurationen der Indextypen *SFBlogContentSearch.solr* und *SFBlogContentSearch.factfinder-ws* definiert sind, dargestellt. Das statische Attribut *Title* ist neben dem Attribut *Text* in der zweiten Tabelle sichtbar, in der bereits zur Indizierung ausgewählte Attribute aufgelistet sind. In der ersten Tabelle sind dagegen die Attribute dargestellt, welche nicht indiziert sind.

Erweiterung der Konfiguration
In der Standard-Implementierung der Cartridge ***bc_foundation*** wird bei der Erzeugung eines neuen Indizes seine Eindeutigkeit mithilfe der Eigenschaften Name, Kontext (Domain), in dem sich der Index befindet, und Indextyp gewährleistet. Der Produkt- bzw. Content-Index wird damit genau zu dem Shop zugewiesen, für den der Index erzeugt wurde.

Da ein Shop mehr als einen Blog haben kann, gilt diese eindeutige Zuordnung nicht mehr für den Blog-Index. Eine mögliche Lösung ist, die Indextypen *SFBlogContentSearch.solr* und *SFBlogContentSearch.factfinder-ws* um ein weiteres Attribut zu erweitern. Hierzu

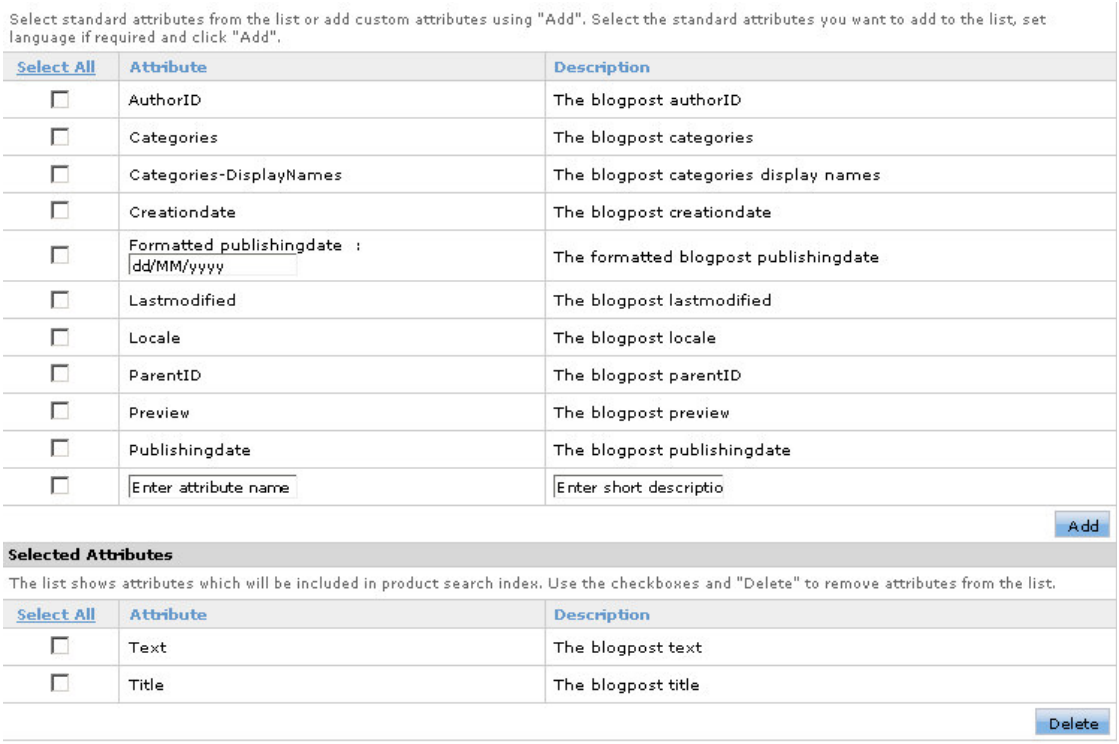

Abbildung 5.4: Auswahl der Attribute für den Index

bietet sich der Identifikationsschlüssel des Blogs (BlogID) an, welcher als vierte Eigenschaft die Eindeutigkeit der Blog-Indizes garantieren wird.

Für diese Erweiterung wird die Datei *Blog-Config.xml* erzeugt, welche die *BlogID* enthält und in Kombination mit der Konfigurationsdatei (*config.xml*) die Eindeutigkeit bei der Zuordnung von Index zu Blog gewährleistet. Für die Eindeutigkeitsprüfung wurden die Klassen *BlogIndexConfigurationBase*, *BlogIndexConfiguration* und *BlogIndexConfigurationImpl*, deren Implementierungen sich in Listings A.2, A.3 und A.4 befinden, geschrieben. Diese Klassen erzeugen bei dem Indexerstellungsprozess mithilfe der Java Architecture for XML Binding (JAXB)-Programmierschnittstelle die Datei *Blog-Config.xml*, welche in dem Indexordner gespeichert wird. *BlogIndexConfigurationBase* enthält Definitionen und die Methoden zum Auslesen und Beschreiben einzelner Merkmale der Konfigurationsdatei. Das Interface *BlogIndexConfiguration* repräsentiert die Konfigurationsdatei als ein Java-Objekt, dessen Methoden in der Klasse *BlogIndexConfigurationImpl* implementiert werden. .

Entsprechend dem Konzept von Enfinity Suite (vgl. Abschnitt 2.3.1) wurden neben den erwähnten Klassen Pipelets für die Verarbeitung der Konfiguration innerhalb der Pipelines geschrieben. Mit diesen werden die Funktionen zum Erstellen, Auslesen sowie zum Laden von Blog-Indizien der Konfigurationen aus den Dateien *config.xml* und *Blog-Config.xml* abgedeckt.

Weiterhin wurde eine Pipeline für die Verarbeitung von Suchanfragen erstellt, welche über das Sucheingabefeld der Blog-Seite in das System gestellt werden. Nachdem sichergestellt wurde, dass der entsprechende Indextyp aktiv und ein zugehöriger Index vorhanden ist,

wird ein Anfrage-Objekt erzeugt. Im letzten Schritt wird je nach Anwendungsfall die Suchanfrage auf dem entsprechenden Suchserver ausgeführt.

Für die Anzeige der Suchergebnisse konnte das bereits vorhandene Template zur Darstellung der Beiträge im Blog verwendet werden. Wenn im Suchergebnis eine große Anzahl an Blog-Einträgen vorhanden ist, erfolgt der Zugriff über einen Seitenauswahlmechanismus, wobei im Standardfall zehn Beiträge pro Seite angezeigt werden. Die Standardimplementierung kann für die Suchergebnisse aufgrund der spezifischen Suchergebnisplatzierung der jeweiligen Suchplattform nicht verwendet werden. Nach der Ausführung der Suchanfrage werden von der Suchmaschine (FACT-Finder oder Solr) immer nur eine bestimmte Anzahl an Suchergebnissen übertragen. Die Umsetzung dieser Funktion wurde im Template *BlogPostsIndexPaging* realisiert, welches bei der Darstellung von Suchergebnissen aufgerufen wird und im Anhang als Listing A.5 beigefügt ist.

5.2.3 Zusammenfassung

Auf Basis von Intershop-Modulen, welche die grundlegende Funktionalität und Schnittstellen für die Kommunikation mit FACT-Finder- bzw. Solr-Suchserver bereitstellen, wurden die Cartridges ***ac_sc_search_factfinder*** und ***ac_sc_search_solr*** entwickelt. Die neuen Module integrieren die Suchfunktionalität in den Blog von SCOOBOX der dotSource GmbH, welche mittels der Suchmaschinen FACT-Finder bzw. Apache Solr bereitgestellt wird.

In den Cartridges wurde jeweils ein neuer Indextyp für die Indizierung von Inhalten der Blog-Beiträge implementiert. Die Indextypen erweitert die Standardkonfiguration von Indizes um die Eigenschaft *BlogID*, sodass die eindeutige Zuweisung von Index zum Blog gewährleisten wird. Für den Transport der Daten zum Suchserver und die Aufbereitung der Suchresultate wurde vorhandene Implementierung der Intershop-Logik überschrieben und für den Blog-Index erweitert. Anschließend wurde der Seitenauswahlmechanismus sowie die Darstellung der Suchergebnisse angepasst und überschrieben.

Der Integrationsaufwand ist für beide Suchplattformen durch die gleiche Vorgehensweise bezüglich dem zeitlichen Umfang und der Komplexität gleich zu gewichten. An vielen Stellen konnte die vorhandene Lösung einer Suchplattform mit geringem Aufwand für die andere Suchplattform angepasst werden. Bemerkenswerte Unterschiede wurden bei der Konfiguration der beiden Suchplattformen festgestellt. Während FACT-Finder ohne weitere Anpassungen mit der Standardkonfiguration bereits gute Suchergebnisse auf den Testdaten geliefert hat, gab es bei der Einstellung des Solr Servers Schwierigkeit die richtige Konfiguration vorzunehmen. Solr erfordert mehr spezifisches Wissen über die Arbeitsweise der Suchplattform und deren Datentypen, erlaubt jedoch mehr Transparenz des Verarbeitungsablaufs als FACT-Finder. So konnte die Konfiguration aufgrund des zeitlichen Rahmens nicht bis im Detail an die Umgebung und die Daten angepasst werden, was u. U. die Ergebnisse des Abschnittes 5.3 verfälscht hat.

5.3 Gegenüberstellung von Messergebnissen

Für den Performance-Vergleich von FACT-Finder und Solr in dem Blog von SCOOBOX wurden ausgewählte Leistungsmerkmale gemessen und qualitativ gegenübergestellt.

5.3.1 Testumgebung

Vor dem eigentlichen Vergleich wurden zunächst reproduzierbare Voraussetzungen geschaffen. Damit die Messwerte unabhängig von den äußeren Faktoren erfasst werden konnten, wurden die Tests auf dem gleichen System ausgeführt. Folgende Testumgebung wurde für die Generierung der Messergebnisse eingesetzt.

Enfinity Suite: Für die Entwicklung und die Tests wurde Enfinity Suite 6 in der Version 6.4.0.3.7 eingesetzt, welche als Applikationsserver Apache Jakarta Tomcat 6 verwendet. Der Applikationsserver führt Java-Anwendungen in der Laufzeitumgebung (Java Server Edition Runtime Environment-JRE) der Version 1.6.0_15 aus.

Datenbanksystem: In Enfinity Suite wird standardmäßig Oracle eingesetzt, das in der Testumgebung in der Version *10g Enterprise Edition Release 10.2.0.1.0 - Production* verwendet wurde. Als Bedienungswerkzeug für Datenbankanfragen wurde *SQL Developer 2.1.1.64* eingesetzt.

Browser: Für Interaktion mit der Web-Applikation und Darstellung der Ergebnisse wurde der Browser *Mozilla Firefox 8.0* herangezogen, weil er für die Entwicklung eine Vielzahl an hilfreichen Werkzeugen bereitstellt.

FACT-Finder: Der Suchserver*FACT-Finder* wurde in der *Version 6.3.13.30* eingesetzt, welche als Web-Server Apache Tomcat 6.0.29 verwendet.

Apache Solr: Der Suchserver *Solr* wurde in der *Version 1.4* eingesetzt, welche als Web-Server Apache Tomcat 6.0.33 verwendet.

System: Als Plattform für die beiden Suchserver und den Enfinity Suite Applikationsserver wurde der Arbeitsplatzrechner mit folgenden Basisinformationen eingesetzt. Ein Betriebssystem mit 64-Bit-Version von Windows 7 Professional - Service Pack 1 mit dem Prozessor Intel Core i5 CPU 760 @ 2.80GHz und 16 Gigabyte Arbeitsspeicher.

5.3.2 Outbacker Blog

Der Outbacker Blog enthält nach der Installation des Systems initial elf Blog-Beiträge. Da diese Menge für eine aussagekräftigen Leistungsvergleich unzureichend ist, wurden die Beiträge von *Handelskraft*[2] und *Socialcommerce*[3] am 23.09.2011 abgerufen und in den Blog von SCOOBOX importiert. Die thematische Ausrichtung dieser Blogs fällt in die Bereiche E- und Social-Commerce, wobei die Beiträge in deutsche Sprache verfasst sind. Nach dem Import waren insgesamt 1876 Beiträge in dem Blog vorhanden, sodass ein etwas repräsentativerer Suchraum zur für den Test der Suchplattformen geschaffen wurde. Für die Suche wurden ausschließlich die Attribute *Title* und *Text* verwendet. Das hat den Hintergrund, dass bei dem Import der Beiträge keine weiteren Attribute importiert wurden, weil diese überwiegend numerischen Datentyp hatten und somit für die Volltextsuche ungeeignet wären.

[2] http://www.handelskraft.de/
[3] http://www.socialcommerce.de/

5.3.3 Indizierung

Die Indizierung erfolgt bei dem Indexerstellungsprozess und ist grundlegend für jede Suchplattform. Bei jeder Änderung des Datenbestandes ist die erneute Indizierung erforderlich, damit die Suchergebnisse stets aktuell bleiben. Hierbei stellt sich die Frage wie aufwendig dieser Prozess ist und, ob die Aktualisierung automatisiert oder manuell ausgeführt werden sollte. Bei der automatischen Aktualisierung besteht die Gefahr, dass es im laufenden Betrieb zu Leistungseinbussen aufgrund des hohen Ressourcenverbrauchs kommen kann. Diesem Nachteil kann mit manueller Aktualisierung durch Akzeptanz von kurzen Inkonsistenzen der Suche aus dem Weg gegangen werden. Somit kann beispielsweise die automatisierte Aktualisierung in den Zeiträumen, wenn der Online-Shop kaum besucht wird, einen Kompromiss zwischen der manuellen und automatischen Aktualisierung bedeuten.

Weiterhin ist das zeitliche Aktualisierungsverhalten der Suchplattformen mit leicht modifizierten Daten interessant. Ab welchen Änderungsanteil ziehen die Suchplattformen den kompletter Update dem Teilupdate des Indexes vor. Da in der Standardimplementierung immer eine kompletter Update durchgeführt wird, kann so ein Verhalten im Rahmen dieser Arbeit nicht getestet werden.

Aus den genannten Überlegungen wird das Interesse geweckt die Laufzeiten, welche die Suchserver für diesen Prozess benötigen, zu untersuchen. Um das Verhalten der Suchplattformen bei wachsendem Datenbestand zu untersuchen, wurden die Beiträge vervielfacht und erneut indiziert. Das hat zur Folge, dass die Anzahl der indizierten Wörter hierbei identisch bleibt und die Länge der Posting Liste (vgl. Abschnitt 2.2.2) einzelner Terme ansteigt. Somit ist dieses Beispielszenario nicht hundertprozentig repräsentativ, es zeigt dennoch die Unterschiede in der Laufzeit beider Plattformen auf.

In der Abbildung 5.5 ist die durchschnittlich benötigte Zeit für die Indizierungsprozesse in Millisekunden für unterschiedliche Datenmengen dargestellt. Die Messwerte wurden in der *System Management Console (SMC)*, welche in Enfinity Suite u. a. die zeitliche Überwachung der Performance erlaubt, abgelesen. Speziell wurde die Performance des Pipelets *AddObjectsToSearchIndex*, welches den Datenimport ausführt, betrachtet. Für jede Datenmenge wurden pro Suchplattform drei Messwerte abgelesen, der Durchschnittswert sowie die Standardabweichung und in den Tabellen B.1 und B.2 (Anhang B) erfasst.

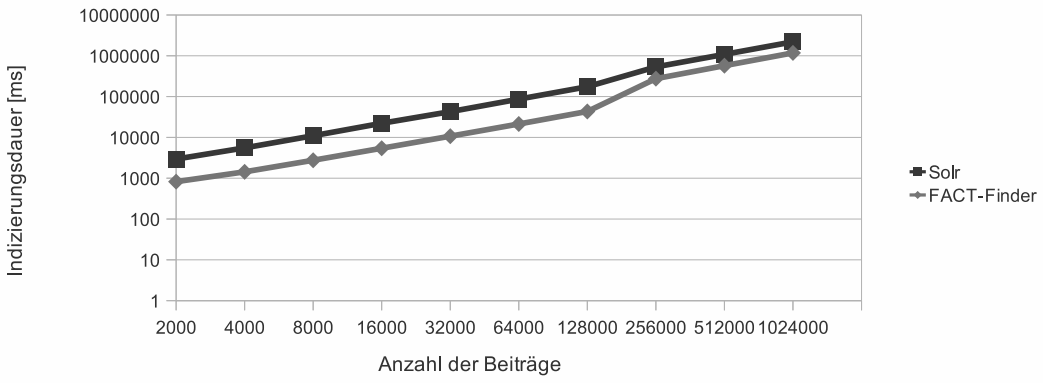

Abbildung 5.5: Vergleich der Indizierungsdauer in Abhängigkeit von Beitragsanzahl

Das Ergebnis dieser Messung hat gezeigt, dass die Indizierung des Solr-Indexes bei jeder Stichprobe mehr Zeit erfordert hat, als die des FACT-Finder-Indexes. Bis 100.000 Einträge ist die Laufzeit von Solr über das Vierfache des FACT-Finders angestiegen. Die Indizierungsdauer für mehr als 256.000 Beiträge haben einen Faktor von über 200%. Eine mögliche Erklärung bietet der strukturelle Unterschied der beiden Indizes. Wie bereits im vorigen Kapitel erwähnt wurde, wird bei FACT-Finder nur eine Datenbankdatei, die Konfigurationsdateien ausgenommen, angelegt, während bei Solr der Index aus mehreren Dateien besteht. Das hat zur Folge, dass bei Solr mehrere Dateien bei der Analyse und Aufnahme eines Terms in den Index angefasst und gegebenenfalls verändert werden müssen. Jede Dateioperation erfordert Ressourcen und kann sich auf diese Weise in der Verarbeitungszeit auswirken.

Es ist auch denkbar, dass die verschiedenen Herangehensweisen bei der algorithmischen Umsetzung der Prozesse zur Analyse, Aktualisierung und Optimierung der Indizes den maßgeblichen Anteil an dem Laufzeitunterschied bewirken.

Neben der Laufzeit spielt die Index-Größe ebenfalls eine wichtige Rolle für die Performance der Suche. Wie verhält sich der Overhead, welcher bei der Indizierung der Daten immer vorhanden ist, in Abhängigkeit von dem Zuwachs an Daten.

Eine nach oben skalierende Datenmenge impliziert einen mit wachsenden Index. Die Frage über die Skalierbarkeit des Overheads, welcher durch die Indizierung der Daten immer vorhanden ist, gibt ebenfalls eine Aussage über die Leistungsfähigkeit der verwendeten Algorithmen. Insbesondere hat die Indexgröße den direkten Einfluss auf den Speicherverbrauch, weil für der performanten Zugriff der Index in den Arbeitsspeicher geladen wird. In der Abbildung 5.6 sind die Index-Größen der jeweiligen Suchplattformen für verschiedene Datenmengen in einem Balkendiagramm veranschaulicht.

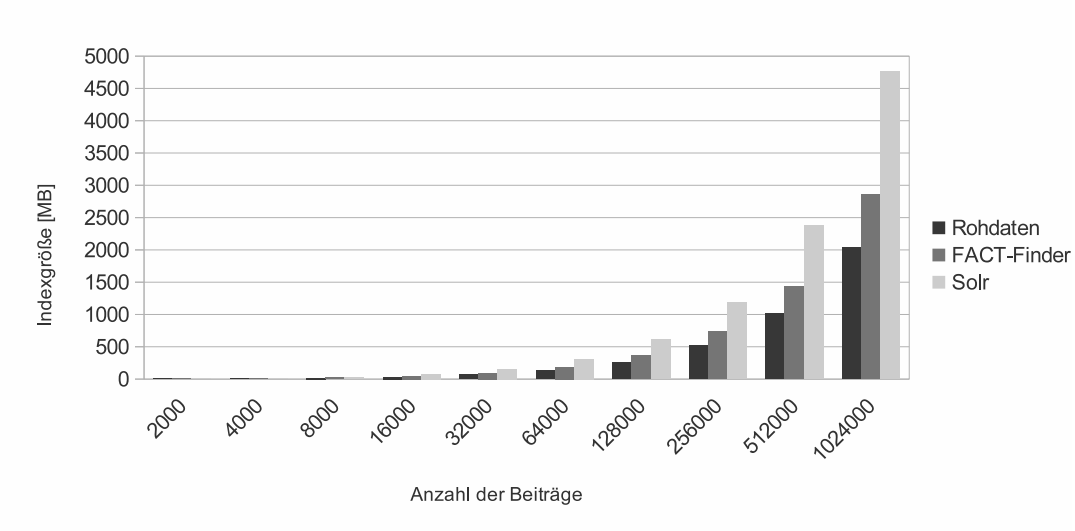

Abbildung 5.6: Vergleich der Indexgrößen in Abhängigkeit von Beitragsanzahl

Die Größen aus dem Diagramm sind im Anhang B in der Tabelle B.3 erfasst. Die Messwerte wurden direkt aus dem Dateisystem, wo die Indexdaten gespeichert werden, abgelesen. Die Originalgrößen wurden zum Vergleich ebenfalls dargestellt. Diese stammen aus der CSV-Datei, welche für den Importvorgang in dem FACT-Finder-Indexordner erzeugt wird.

Man sieht deutlich, dass die Größe der Solr-Indizes, ausgenommen die Werte bei 2.000 Beiträgen, mehr Speicherplatz als die FACT-Finder-Indizes erfordern. Dabei pendelt sich das Verhältnis der Index- zu der Originalgröße ab 128.000 Beiträgen bei beiden Suchplattformen um einen festen Wert ein. Bei FACT-Finder ist der Overhead in etwa 40% und bei Solr 133% von der Originalgröße der indizierten Daten.

Der deutliche Größenunterschied kann zum Einen wieder auf die Anzahl der für den Index angelegten Dateien und der Struktur dieser Dateien gesehen werden. Vermutlich liegt jedoch der Hauptunterschied in dem strukturellem Aufbau der Indizes in folge verschiedener Algorithmen, welche für die Suche eingesetzt werden. Des Weiteren muss auch in diesem Test die Tatsache, dass die Beiträge dupliziert worden sind, berücksichtigt werden.

5.3.4 Qualität der Suche

In diesem Abschnitt soll die Qualität der Suche durch die Analyse der von den Suchplattformen generierten Suchergebnisse analysiert und bewertet werden, wobei wieder der Ausgangszustand von 1876 Beiträgen erzeugt wurde. Die Qualität der Suche impliziert in diesem Zusammenhang die schnelle Ausführung der Suche und sowie eine relevante Ergebnismenge von Dokumenten. Somit sind diese Kriterien fundamental für Information Retrieval (vgl. Definition von IR-Abschnitt 2.1). Da es kein automatisches Auswertungssystem für die Schätzung der Relevanz von Suchergebnissen vorhanden war, wurden die Suchergebnisse „von Hand“ analysiert als relevant oder nicht relevant zu dem gestellten Suchbegriff angesehen. Aufgrund der manuellen Auswertung wurden die Suchbegriffe mit Absicht gewählt, um kleine Ergebnismengen zu erhalten. Weiterhin wurde in der Datenbank nach den Begriffen gesucht, um einen Referenzwert über die Menge von Beiträgen mit exakter Übereinstimmung zu erhalten. Um das Ergebnis der Datenbank zu erhalten wurde die SQL-Anfrage aus Listing 5.2 verwendet.

```
1 SELECT title, text
2 FROM scblogpost
3 WHERE title like '%Suchbegriff%' OR text like '%Suchbegriff%'
```

Listing 5.2: SQL-Anfrage für den Referenzwert

Das Ergebnis der Oracle-Anfrage dient somit als Referenzwert für das tatsächliche Aufkommen der Suchbegriffe ohne der Texttransformationen wie z. B. Wortstammreduzierung und phonetische Analyse (vgl. Abschnitt 2.2.1). Für die im Anschluss aufgeführten Performance-Zahlen ist natürlich die Frage nach Indizes in Oracle interessant, um auch die dortigen Möglichkeiten FACT-Finder und Solr gegenüberzustellen. Im zeitlichen Rahmen dieser Arbeit war dies jedoch einfach nicht machbar.

Die Untersuchung basiert auf zwei Szenarien, bei den die Suche mit und ohne phonetische Analyse des Suchraums erfolgte. Hierfür wurde bei Solr der phonetische Filter aus der Konfiguration entfernt und die Stoppwortliste, welche für die deutsche und englische Sprache ausgelegt ist, mit den Standardeinträgen von dem FACT-Finder Server gefüllt. Weitere Filter für die Stammwortzerlegung und Thesauri wurde bei Solr ebenfalls nicht aktiviert. Bei FACT-Finder wurde der Regler für die Kompromisseinstellung zwischen der Schnelligkeit und Präzision der Suche auf die maximale Stufe 9 gesetzt, so dass gemäß [Doc11] die Phonetik ausgeschaltet wurde und die Suche nach 2.000 durchsuchten Einträgen automatisch abbricht. Genauso wie bei Solr wurden auch bei FACT-Finder keine Konfiguration im Bezug auf Thesauri und Stammwortreduzierung explizit vorgenommen.

Die Zeitangaben wurden bei diesen Tests ebenfalls in dem CMS-Konsole aus dem Pipelet *QuerySearchIndex*, welches die Suchanfragen auf dem Index der Suchplattform ausführt, abgelesen.

Im Folgenden werden die Ergebnisse der Suche an sieben Beispielen erläutert und gegenübergestellt. Die Auswahl der Begriffe sollte möglichst breites Spektrum an unterschiedlichen Szenarien mit geringer Anzahl an Beispielen abdecken, wobei folgende Begriffe verwendet wurden: *Schneekugel*, *Shnekugel*, *Machtkampf*, *Volltextsuche*, *GEZ-Gebühr*, *Blog-Verzeichnis* und *Intershop*.

Schneekugel: Die Datenbank generiert zwei Suchergebnisse im *Text*, welche das Suchwort komplett in der Form enthalten. Diese Ergebnisse sind ebenfalls bei Solr und FACT-Finder mit phonetischen Suche erhalten. Wenn die Suche ohne Phonetik erfolgt, so liefert Solr erwartungsgemäß zwei exakte Ergebnisse zurück. Das Suchergebnis von FACT-Finder enthält ebenfalls zwei Beiträge. Mit Phonetik enthält die Ergebnismenge von Solr ein weiteres Ergebnis. Der von Solr verwendete phonetische Filter berechnet für *Schneekugel* und *Schwungkraft* den selben phonetischen Code, so dass der Beitrag (in dem das Stichwort vorkommt) fälschlicherweise in das Suchergebnis aufgenommen wird.

Solr und FACT-Finder ordnen die zwei richtigen Ergebnisse unterschiedlich in ihren Ergebnissen an. Solr ordnet den Beitrag, wo der Begriff vier mal vorkommt, ganz oben ein. FACT-Finder dagegen stuft den Beitrag, wo der Begriff einmal vorkommt, jedoch weiter vorne im Text steht als relevanter ein, was entweder auf die Berücksichtigung der Position oder Ignorieren der weiteren Vorkommnisse hindeutet.

Schnekugel: In der Datenbankanfrage wird dieser Begriff nicht gefunden. Ohne Phonetik findet Solr keine Ergebnisse, FACT-Finder kann dagegen einen Beitrag mit *Schneekugel* immer noch finden. Was auf die weiterhin vorhandene Fehlertoleranz bei der Suche vermuten lässt. Mit Phonetik stimmen die Ergebnisse mit der Suche nach *Schneekugel* erwartungsgemäß überein.

Machtkampf: Die SQL-Abfrage liefert drei Suchergebnisse, wo der Begriff in exakter Form vorhanden ist und in jedem Post genau einmal im Attribut *Text* vorkommt. Bei der Suche ohne Phonetik enthält das Suchergebnis von Solr und FACT-Finder die exakten Ergebnisse. Das Ergebnis mit Phonetik stimmt bei FACT-Finder mit dem der Datenbank überein. Solr hat neun Suchergebnisse gefunden, von den fünf durch die gleiche Phonetik des Begriffs *mächtig* und ein Beitrag über den Begriff *Machtwechsel* als relevant identifiziert wurden. Nach visueller Inspektion ist der Post mit dem Begriff *Machtwechsel* aus thematischer Sicht durchaus als relevant für den Suchbegriff anzusehen, was vermutlich aufgrund des Präfixes *Macht* und nicht aufgrund der Semantik den Treffer erkannt hat.

Bei FACT-Finder wurden die Ergebnisse nach genauerem Betrachten vermutlich nach der Position des Vorkommens im Text angeordnet werden. Die Anordnung von Solr ist in diesem Fall anders als bei FACT-Finder. Die Posts, wo auf den ersten Blick der Textumfang geringer ist, werden weiter oben angeordnet.

Volltextsuche: Laut der Datenbanksuche Oracle enthält kein Post diesen Suchbegriff. Ohne Phonetik liefert Solr keine Ergebnisse. FACT-Finder findet ein Ergebnis mit

dem Begriff *Volltext*. Mit Phonetik findet FACT-Finder vier Posts, welche die Begriffe *Volltext*, *Postleitzahlensuche*, *Suche* und *Preissuche* in dieser Reihenfolge enthalten. Der letzte Post enthält neben *Preissuche* achtmal *Suche* und einmal *suchen*. Solr findet nur zwei Posts von den das erste Ergebnis mit dem von FACT-Finder übereinstimmt. Der zweite Post enthält keine relevanten Ergebnisse. Ohne Phonetik liefert Solr keine Ergebnisse. FACT-Finder findet immer noch das erste Ergebnis.

GEZ-Gebühr: Mit der SQL-Abfrage wird exakt ein Ergebnis mit dem Suchbegriff im Attribut *Text* gefunden. Mit Phonetik kann FACT-Finder das eine Ergebnis finden, bei Solr kommt allerdings ein leeres Ergebnis zurück. Nach der Überprüfung in dem Analysetool im Admin Iterface wurde das Ergebnis zurückgeliefert. Somit ist das Problem vermutlich bei der Übertragung von Umlauten aufgetreten. Ohne Phonetik wird das Ergebnis für diesen Suchbegriff nicht verändert.

Blog-Verzeichnis: In der Datenbank wird genau ein Blog-Eintrag, welcher den Suchbegriff exakt im Attribut *Text* enthält, gefunden. Ohne Phonetik finden Solr und FACT-Finder erwartungsgemäß den eine Blog-Beitrag. Dasselbe Suchergebnis wird von Solr mit und ohne Phonetik gefunden. FACT-Finder findet ohne Phonetik ebenfalls den einen Post. Mit Phonetik zeigt Solr keine Veränderung. Im Gegensatz dazu findet FACT-Finder weitere Beiträge, welche die Begriffe *Blogverzeichnis* und *Videoblog-Verzeichnis* enthalten.

Intershop: Die Datenbank findet 86 Suchergebnisse, von den 32 Beiträge in bei den Attributen, einmal nur im *Titel* und 53 Mal nur im *Text* vorkommen. Ohne Phonetik findet Solr 97 und Solr 9 Ergebnisse. Mit Phonetik sind es bei der Suche mit Solr 1397 und mit FACT-Finder 99 Suchergebnisse.

In der Tabelle 5.2 ist die Anzahl mit durchschnittlicher Zeit in Millisekunden in eckigen Klammern für die Berechnung der Ergebnisse jeweils für Indizes mit (m.P.) und ohne Phonetik (o.P.) erfasst.

Suchbegriff	**Solr**		**FACT-Finder**		**Oracle**
	m. P.	o. P.	m. P.	o. P.	
Schneekugel	3 [7,9 ms]	2 [5,3 ms]	2 [69,5 ms]	2 [62,5 ms]	2 [272,6 ms]
Schnekugel	3 [7,9 ms]	0 [4,3 ms]	2 [55 ms]	1 [28,1 ms]	0 [368 ms]
Machtkampf	9 [10,6 ms]	3 [5,7 ms]	3 [69,5 ms]	3 [64,8 ms]	3 [244,3 ms]
Volltextsuche	2 [6,2 ms]	0 [4,6 ms]	4 [122 ms]	1 [112,7 ms]	0 [237,6 ms]
GEZ-Gebühr	0 [12,7 ms]	0 [5,5 ms]	1 [51,3 ms]	1 [51 ms]	1 [283,3 ms]
Blog-Verzeichnis	1 [7 ms]	1 [15,2 ms]	3 [124,4 ms]	1 [120,9 ms]	1 [235,6 ms]
Intershop	1397 [44,9 ms]	97 [11,1 ms]	99 [59,3 ms]	9 [52,9 ms]	85 [251 ms]

Tabelle 5.2: Anzahl und Berechnungsdauer der Ergebnisse

Die Datenbank kann mit der Ausführungszeiten der Suchplattformen nicht mithalten und erfordert die meiste Zeit bei der Berechnung der Suchergebnisse. Im Vergleich zwischen Solr und FACT-Finder ist Solr bezüglich der Berechnungsdauer klar überlegen. Möglicherweise macht sich an dieser Stelle die vermutlich, aufgrund der Größenunterschiede zu FACT-Finder Index (vgl. Abschnitt 5.3.3), komplexere Struktur des Solr Indexes bei der Berechnungsdauer bemerkbar. Im Fall des DBMS Oracle wurden keine Optimierungen

und Indizes für den Text erstellt, was gravierenden Laufzeitunterschiede im Vergleich zu den Suchplattformen erklärt.

Beide Suchplattformen unterstützen boolesche Operatoren für die Konstruktion von komplexen Suchanfragen. Bei den Tests, welche an dieser Stelle nicht erfasst wurden, hat sich gezeigt, dass die grundlegenden Operatoren wie UND, NOT und ODER unterstützt werden.

5.3.5 Erkenntnisse

Aus der Gegenüberstellung der Suchbegriffe konnten folgende Schlussfolgerungen gezogen werden.

Die exakte Suche funktioniert bei beiden Suchplattformen bis auf die Ausnahme mit dem Begriff *GEZ-Gebühr* bei Solr, welches vermutlich durch ein Umlaut-Problem zu keinem Ergebnis geführt hat, ansonsten in allen Fällen gut funktioniert und richtigen Ergebnisse gefunden.

Mithilfe der Werkzeuge für die Analyse von Suchanfragen im Admin Interface von Solr konnte für die Suchbegriffe *Schneekugel* und *Machtkampf* überprüft werden, dass der eingesetzte Algorithmus für die phonetische Analyse zum Teil Übereinstimmungen festgestellt hat, welche aus subjektiver Betrachtung semantisch nicht korrekt waren. Beispielessweise wurden *Schneekugel* und *Schwungkraft* als phonetisch ähnlich bewertet.

Insgesamt hat FACT-Finder mit eingeschalteter Phonetik für jedes der betrachteten Suchbegriffe ein aus subjektiver Sicht besseres Suchergebnis gefunden. Ohne Phonetik haben die Suchplattformen erwartungsgemäß schneller die Suche ausgeführt und weniger Suchergebnisse gefunden. In Anbetracht der Suchergebnisse hat FACT-Finder nicht aufgrund der Phonetik, sondern durch souveräne Wortzerlegung und Thesaurus-Nutzung ein besseres Suchverhalten gezeigt. Wie vollständig jedoch das Ergebnis war, kann nicht genau eingeschätzt werden, weil die tatsächlich relevante Ergebnismenge nicht bekannt ist.

Die Laufzeit bei der Suche hat die Wichtigkeit eines Indexes für die Volltextsuche bestätigt. Das wurde durch die Laufzeitunterschiede bei der Berechnung von Suchergebnissen von Oracle, wo es keine Volltextindizierung gab, im Vergleich zu den Suchplattformen deutlich. Des Weiteren hat die Suchlaufzeit von Solr die von FACT-Finder für jeden Testsuchlauf deutlich unterschritten. Vermutlich besteht an dieser Stelle der Zusammenhang zu der Größe des Indexes und seiner Berechnungsdauer. Ein vermutlich in der Aufbaustruktur komplexerer Index von Solr verbraucht mehr Speicher und Zeit zur Erstellung, ist aber in der Berechnung der Suchergebnisse deutlich effizienter als der Index von FACT-Finder.

Die Suchplattformen haben ein unterschiedliches Verhalten in der Anordnung der Suchergebnisse gezeigt. Was durch die verschiedenen zugrundeliegenden Information Retrieval Modellen und demzufolge Unterschiedliche Algorithmen für die Suchergebnisplatzierungen erklärt werden kann.

Beide Suchplattformen bieten ausgereifte Suchlösungen, welche eine Vielfalt an Funktionalität insbesondere für den Bereich E-Commerce mitbringen. Durch den Performance Vergleich bei der Indizierung und Suche hat sich ein insgesamt besseres Verhalten von FACT-Finder im Vergleich zu Solr aufgezeigt. Dieses Ergebnis muss aus folgenden Gründen kritisch betrachtet werden.

1. Die Dokumentensammlung, welche für die Indizierung und die Suche verwendet wurde, ist möglicherweise nicht repräsentativ, um einen qualitativ guten Vergleich der Performance durchzuführen.

2. Die Konfiguration spielt eine sehr wichtige Rolle für das Suchverhalten, welche nicht selten einen langen Optimierungsprozess für das erreichen optimaler Ergebnisse erfordert.

Kapitel 6

Zusammenfassung und Ausblick

Dieses Kapitel fasst die wichtigsten Erkenntnisse und Lücken dieser Arbeit zusammen und gibt einen Ausblick auf mögliche weiterführende Themen.

6.1 Zusammenfassung

Die Ergebnisse der Arbeit werden anhand der, in der Einleitung gesetzten, Ziele zusammengefasst.

1. Ziel – *Definition von Anforderungen seitens der Shoppingportale an die externe Suchplattformen:*

Für einen objektiven Vergleich der Suchplattformen wurden Bewertungskriterien als Anforderungen definiert, welche in die Bereiche *Shop-Betreiber* und *Shop-Nutzer* untergliedert wurden.

Die *Shop-Betreiber-Anforderungen* wurden weiter untergliedert in wirtschaftliche wie z. B. mit der Suchplattform entstehenden Kosten, technische wie die Skalierbarkeit des Systems sowie funktionale Kriterien wie z. B. angebotene Features und Einstellungsvielfalt bei der Administration.

Die *Shop-Nutzer-Anforderungen* wurden ebenfalls weiter kategorisiert in die Bereiche Unterstützung bei der Sucheingabe wie z. B. Autovervollständigung, das Finden wie Performance bei der Suche und Repräsentieren von Suchergebnissen wie z. B. die Filterung der Ergebnisse.

2. Ziel – *Vorstellung, Bewertung und Vergleich von FACT-Finder und Solr:*

FACT-Finder von Omikron Data Quality GmbH und Solr von Apache sind externe Suchplattformen, welche in der E-Commerce-Branche sehr erfolgreich und bei dem Einsatz in Web-Shops weitverbreitet sind. Aus der Sicht des Shop-Betreibers ist ein Vergleich insbesondere interessant, um zu entscheiden, ob sich die kommerzielle Lösung FACT-Finder bezahlt macht oder die frei verfügbare Suchplattform Solr für die Deckung der gestellten Anforderungen ausreicht.

Der Kernalgorithmus der Suchplattform FACT-Finder basiert auf dem Ansatz des Fuzzy-Set-Modells, welcher in Kombination mit der Phonetik bei der Suche zum Einsatz kommt.

FACT-Finder ist modular und kann entweder als Software-as-a-Service mit Omikron als ASP oder als eine Komplettlösung in Anspruch genommen werden. Die Suchplattform Solr benutzt die Lucene-Bibliothek von Apache als Kernalgorithmus, welche das Vektorraummodell als den Ansatz für die Umsetzung der Volltextsuche verwendet. In Kombination zu der Standardsuche kann genau bestimmt werden welche Algorithmen genutzt werden sollen. Solr ist frei verfügbar und wird als ein eigenständiger Apache-Projekt intensiv weiterentwickelt.

Insgesamt bieten beide Suchplattformen ein breites Spektrum an Funktionalität, wobei FACT-Finder mehr den Ansprüchen seitens der E-Commerce-Branche z. B. mit einer Shop-Navigation und Kampagnen genügt. Weiterhin ist die Administration von FACT-Finder benutzerfreundlicher und intuitiver organisiert, als es der Fall bei Solr ist, wo die wichtigen Einstellungen auf der Dateiebene vorgenommen werden müssen. Aus technischer Sicht bietet Solr die Möglichkeit die Suchplattform optimal bis in kleinste Details an die eigenen Anforderungen anzupassen. So ist beispielsweise gezielte Konfiguration von Analyseschritten und die Wahl von eingesetzten Algorithmen möglich. In einem Anwendungsfall ist FACT-Finder für Online-Shops, welche auf Services wie SaaS und Support sowie einfache Konfiguration mit zahlreichen Funktionen setzen, geeignet. Solr zielt dagegen auf Shops ab, die ohne finanziellen Investitionen eine vollwertige und fein konfigurierbare Suchplattform nutzen möchten mit dem Kompromiss etwas mehr Zeit in die technischen Belange zu spenden.

3. Ziel – *Die Ergebnisse des theoretischen Vergleichs anhand der Integration und des Performance Vergleichs überprüfen:*

Die Aufwände für Wartung und Anpassung konnten im Rahmen dieser Arbeit nicht eingeschätzt werden, weil es für die Umsetzung dieser Anforderung keine Erfordernis während der Entwicklungszeit ergeben hat. Der Integrationsaufwand war aufgrund gleicher Vorgehensweise nahezu identisch. Der eigentliche Unterschied stellte sich erst bei der Konfiguration der Indizes heraus. Im Gegensatz zur dateibasierten Solr-Konfiguration war die FACT-Finder Web-Oberfläche denkbar unkompliziert. Auf die Integration von Autovervollständigung und Suchergebnisfilterung wurde aus zeitlichen Gründen im Rahmen der vorliegenden Arbeit verzichtet. Weiterhin wurde durch die Erweiterung der Konfiguration der Indizes für den Outbacker Blog der Caching-Mechanismus der Standardimplementierung nicht beachtet, sodass an dieser Stelle noch Bedarf nach Refaktorierung der Implementierung besteht.

Um das Verhalten im praktischen Umfeld zu testen, wurde die Volltextsuche mithilfe der Suchplattformen in den SCOOBOX Blog der dotSource GmbH integriert und die Umsetzung für ausgewählte Kriterien gegenübergestellt. Laut der Tests zur Indizierung benötigen FACT-Finder Indizes insbesondere bei größeren Datenbeständen weniger Speicherplatz und auch weniger Zeit für die Indizierung. Vermutlich liegt der Hauptgrund in der Struktur der Indizes, welche deutliche Unterschiede in dem Aufbau des Indexes zeigen. Während bei FACT-Finder eine große Datei erstellt wird, legt Solr mehrere Dateien mit unterschiedlichen Funktionen an. Bei der Betrachtung von Laufzeiten der Suchanfragen ist Solr deutlich schneller als FACT-Finder. Dieses Ergebnis kann entweder durch die Verwendung unterschiedlicher Algorithmen oder auch hier durch die Ungleichheit in der Index-Struktur erklärt werden. Die Qualität der Suchergebnisse hat für die Standardkonfiguration bei FACT-Finder für unsere Abfragen und nach der subjektiven Wertung zumindest ein Teil der relevanten Treffer generiert, weil über die Vollständigkeit keine

Aussage getroffen werden kann. Im Gegensatz dazu hat Solr teilweise inadäquate Treffer ausgegeben oder relevante Dokumente ausgeschlossen, was durch die falsche bzw. unvollständige Konfiguration des Indexes erklärt werden kann.

Im Gesamtergebnis haben beide Plattformen bezüglich der Anforderungen etwas zu bieten. Wie gut die einzelnen Kriterien umgesetzt werden, kann aus der theoretischen Betrachtung heraus nicht klar gestellt werden. Zudem bestand die Hauptschwierigkeit darin, dass es zu FACT-Finder kaum Informationen zu internen Abläufen gibt. Teilweise hätte die zusätzliche Information durch Tests einer praktischen Lösung validiert werden können. Was nur zum Teil durch die Integration und anschließenden Tests abgedeckt werden konnte. Für die vollständige Validierung von den Ergebnissen des theoretischen Vergleichs durch Tests einer praktischen Lösung war der zeitliche Rahmen dieser Arbeit nicht ausreichend.

6.2 Ausblick

Folgende Themen sind in Zusammenhang mit Suchplattformen sowie FACT-Finder und Solr für weitere Betrachtungen von Interesse.

Automatisierte Bewertung von Suchplattformen: Aus den Ergebnissen des praktischen Vergleichs resultiert die Anforderung nach einem Werkzeug, welches die Suchqualität automatisiert auswerten kann. Ein möglicher Ansatz ist, ein Test-System, welches aus dem Vergleich von Experten- und Suchmaschinenergebnis eine Bewertung für die Güte des Suchmaschinenergebnisses berechnet. Die Herausforderungen bestehen in der Aufstellung von Bewertungskriterien, welche die Suchqualität möglichst nah am menschlichen Beurteilungsverhalten beschreiben, der Beschaffung von Expertenergebnissen, welche qualitativ hochwertige Referenzwerte liefern, und der Implementierung des Prototypen, welcher die automatische Bewertung an konkreten Suchplattformen wie z. B. FACT-Finder und Solr testet. In [Lew05] werden für die Qualitätsmessung bei Suchmaschinen Evaluationsmaße vorgestellt, welche als solche Bewertungskriterien verwendet werden können.

Umfassender Vergleich weiterer Plattformen: Neben dem Vergleich von Solr und FACT-Finder gibt es die Möglichkeit weitere Suchplattformen anhand der im Kapitel 3 gestellten Anforderungen gegenüberzustellen. Hierbei könnte ein umfassender Vergleich gleichzeitig mehrerer Suchplattformen insbesondere aus der Sicht kleinerer Online-Shops interessant sein.

Adaption der Implementierung auf Foren und Wikis: Die im Rahmen dieser Arbeit entstandene Implementierung kann auf weitere Komponenten ausgedehnt werden. Hierfür bieten sich Foren und Wikis an. Die Herausforderung könnte in der Implementierung einer verallgemeinerten Herangehensweise liegen, wobei die Adoption für mehrere Ressourcen gleichzeitig durchgeführt wird.

Volltextsuche in DBMS und IRS : Der Vergleich von DBMS und IRS im Bezug auf Freitextsuche wurde bereits im Abschnitt 5.3.4 erwähnt. Auf diesem Gebiet existieren zahlreiche Lösungen für DBMS-Produkte wie z. B. Oracle oder Microsoft SQL Server. Hierbei ist es interessant die Suche in Freitexten eines IRS mit der des DBMS zu vergleichen und auszuwerten. Die Evaluation für so einen Vergleich kann in einer praktischen Anwendung gefestigt werden.

Anhang A

Implementierung

```
1  <template-if
2   condition="SearchIndex:Configuration:Locale:LocaleID EQ LeadLocale">
3    SELECT post.*, COUNT(post.uuid) OVER() AS ROWCOUNT
4    FROM scblogpost post
5    WHERE
6     post.parentid is NULL
7     ...
8  </template-if>
9  <template-if
10  condition="(SearchIndex:Configuration:Locale:LocaleID NE LeadLocale)
11        AND
12        (Blog:Fallback == 0)">
13   SELECT post.*, COUNT(post.uuid) OVER() AS ROWCOUNT
14   FROM scblogpost post
15   WHERE
16    post.locale =
17     <template-variable value="SearchIndex:Configuration:Locale:LocaleID"/>
18    ...
19 </template-if>
20 <template-if
21  condition="(SearchIndex:Configuration:Locale:LocaleID NE LeadLocale)
22        AND
23        (Blog:Fallback == 1)">
24   SELECT post.*, COUNT(post.uuid) OVER() AS ROWCOUNT
25   FROM scblogpost post
26   WHERE
27    post.uuid IN
28    (
29     (SELECT s.uuid
30     FROM scblogpost s
31     WHERE
32      s.locale =
33       <template-variable value="SearchIndex:Configuration:Locale:LocaleID"/>
34     OR
35      s.parentid is NULL)
36
37     MINUS
38
39     (SELECT t.uuid
40     FROM scblogpost t
41     WHERE t.uuid IN (SELECT parentid FROM scblogpost))
42    )
```

```
43    ...
44 </template-if>
```

Listing A.1: GetBlogPostsToIndex.query

```
1  ...
2  import javax.xml.bind.annotation.XmlElement;
3  import javax.xml.bind.annotation.XmlRootElement;
4  import javax.xml.bind.annotation.XmlType;
5  ...
6  /**
7  * search index configuration bean implementation that is used to read/write
8  * the configuration from the underlying xml representation.
9  */
10 @XmlRootElement(name="blogIndexConfiguration")
11 @XmlType(name="BlogIndexConfigurationBase",
12       propOrder={"featureID", "indexID", "blogID"})
13
14 public class BlogIndexConfigurationBase extends CustomXmlValuesObject
15 {
16     private String featureID;
17     private String indexID;
18     private String blogID;
19
20     public BlogIndexConfigurationBase(){}
21
22     @XmlElement(required = true)
23     public String getFeatureID()
24     {
25         return featureID;
26     }
27
28     public void setFeatureID(String featureID)
29     {
30         this.featureID = featureID;
31     }
32
33     @XmlElement(required = true)
34     public String getIndexID()
35     {
36         return indexID;
37     }
38
39     public void setIndexID(String indexID)
40     {
41         this.indexID = indexID;
42     }
43
44     @XmlElement(required = true)
45     public String getBlogID()
46     {
47         return blogID;
48     }
49
50     public void setBlogID(String blogID)
51     {
52         this.blogID = blogID;
53     }
54 }
```

Listing A.2: BlogIndexConfigurationBase.java

```
1  ...
2  public interface BlogIndexConfiguration
3  {
4      public void setFeatureID(String featureID);
5
6      public String getFeatureID();
7
8      public void setIndexID(String indexID);
9
10     public String getIndexID();
11
12     public void setBlogID(String blogID);
13
14     public String getBlogID();
15
16     public abstract boolean save();
17 }
```

Listing A.3: BlogIndexConfiguration.java

```
1  ...
2  public class BlogIndexConfigurationImpl implements BlogIndexConfiguration
3  {
4
5      private static String CONFIG_FILE_NAME = "Blog-Config.xml";
6      protected String indexID;
7      protected Domain domain;
8      File configurationFile;
9      BlogIndexConfigurationBase configuration;
10     private SearchIndexMgr searchMgr = null;
11
12     public BlogIndexConfigurationImpl(Domain domain, String indexID)
13     {
14         this.indexID = indexID;
15         this.domain = domain;
16
17         searchMgr =
18         NamingMgr.getInstance().lookupManager(SearchIndexMgr.REGISTRY_NAME);
19
20         this.configurationFile =
21         new File(searchMgr.getIndexDirectory(domain, indexID),
               CONFIG_FILE_NAME);
22
23         readConfigurationBase();
24     }
25
26     private void readConfigurationBase()
27     {
28         if(configurationFile.exists())
29         {
30             try
31             {
32                 JAXBContext jc =
33                 JAXBContext.newInstance(BlogIndexConfigurationBase.class);
34                 Unmarshaller unMarshaller = jc.createUnmarshaller();
```

```
35                  configuration = unMarshaller.unmarshal(configurationFile);
36              }
37              catch (JAXBException e)
38              {
39                  Logger.error(this, "Error reading XML configuration file", e
                        );
40              }
41          }
42          else
43          {
44              configuration = new BlogIndexConfigurationBase();
45          }
46      }
47
48      public boolean save()
49      {
50          boolean success = false;
51
52          if (!configurationFile.getParentFile().exists())
53          {
54              configurationFile.getParentFile().mkdirs();
55          }
56
57          try
58          {
59              FileOutputStream newConfigFileOutput = new FileOutputStream(
                    configurationFile);
60
61              JAXBContext jc = JAXBContext.newInstance(
                    BlogIndexConfigurationBase.class);
62              Marshaller marshaller = jc.createMarshaller();
63              marshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, Boolean
                    .TRUE);
64              marshaller.marshal(configuration, newConfigFileOutput);
65
66              newConfigFileOutput.close();
67              success = true;
68          }
69          catch(JAXBException e)
70          {
71              Logger.error(this, "Error writing XML configuration file", e);
72          }
73          catch(FileNotFoundException e)
74          {
75              Logger.error(this, "Error writing XML configuration file", e);
76          }
77          catch(IOException e)
78          {
79              Logger.error(this, "Error writing XML configuration file", e);
80          }
81          return success;
82      }
83
84      @Override
85      public String getBlogID()
86      {
87          return configuration.getBlogID();
88      }
89
```

```
90      @Override
91      public void setBlogID(String blogID)
92      {
93          configuration.setBlogID(blogID);
94      }
95
96      @Override
97      public String getIndexID()
98      {
99          return configuration.getIndexID();
100     }
101
102     @Override
103     public void setIndexID(String indexID)
104     {
105         configuration.setIndexID(indexID);
106     }
107
108     @Override
109     public String getFeatureID()
110     {
111         return configuration.getFeatureID();
112     }
113
114     @Override
115     public void setFeatureID(String featureID)
116     {
117         configuration.setFeatureID(featureID);
118     }
119 }
```

Listing A.4: BlogIndexConfigurationImpl.java

```
1  <iscontent type="text/html" charset="UTF-8" compact="true">
2
3  <!-- SearchIndexPaging.isml -->
4  <%@page import = "com.intershop.component.foundation.capi.searchindex.
       SearchResult"%>
5  <isif condition="#SearchResult:Pageable:ElementCount > 0#">
6    <!-- sorting select -->
7    <isset scope="request" name="SortingAttribute" value="''">
8    <isloop iterator="SearchResult:Query:Sortings" alias="sorting">
9      <isset scope="request" name="SortingAttribute" value="#sorting:
           AsFormParameter#">
10     <isbreak>
11   </isloop>
12   <isloop iterator="SearchIndex:Configuration:Attributes" alias="att">
13     <isif condition="#att:Sortable#">
14       <isset scope="request" name="isReSortable" value="true" />
15     </isif>
16   </isloop>
17   <isif condition="#isDefined(isReSortable)#">
18   <form method="post" action="#URL(Action('ViewScBlogFFSearch-
         ShowSearchIndexResult'),
19     Parameter('SearchParameter', SearchResult:Query:RemoveAllSortings))#">
20   <div class="paging area" style="margin:0px;padding:0px;">
21   <fieldset class="inline" style="padding: 4px 0px 2px 4px;" >
22     <label for="SortingAttribute">Sort By:</label>
```

```
23      <select style="color:#666;" id="SortingAttribute" name="SortingAttribute
            " >
24        <option value="" <isif condition="#SortingAttribute EQ ''#">
25        selected="selected"</isif>>Relevance</option>
26        <isloop iterator="SearchIndex:Configuration:Attributes" alias="att">
27          <isif condition="#att:Sortable#">
28            <option value="#att:Name#-asc" <isif condition="#SortingAttribute
                  EQ (att:Name.'-asc')#">selected="selected"
29            </isif>>#att:DisplayName# ascending</option>
30            <option value="#att:Name#-desc"
31            <isif condition="#SortingAttribute EQ (att:Name.'-desc')#">s
32            elected="selected"</isif>>#att:DisplayName# descending</option>
33          </isif>
34        </isloop>
35      </select>
36      <input type="submit"
37        class="inputSubmitGo"
38        name="changeSorting"
39        title="Sort" value="" />
40    </fieldset>
41    </div>
42    </form>
43    </isif>
44  </isif>
45  <div class="paging area">
46  <isif condition="#SearchResult:Pageable:ElementCount > 0#">
47  <span class="pagingProducts">
48    <span class="emphasis">
49      <isprint value="#SearchResult:Pageable:PositionFirstElement#">
50    </span> -
51    <span class="emphasis">
52      <isprint value="#SearchResult:Pageable:PositionLastElement#">
53    </span> of
54    <span class="emphasis">
55      <isprint value="#SearchResult:Pageable:ElementCount#">
56    </span>
57  </span>
58  </isif>
59  <ul class="linkList">
60  <%
61  SearchResult sr = (SearchResult)getObject("SearchResult");
62  int pageCount = sr.getPageable().getPageCount();
63  int currentPage = sr.getPageable().getCurrentPage();
64  int maxNumberOfPagesToDisplay = 5;
65
66  // link to previous page
67  if (currentPage > 1)
68  {
69   getPipelineDictionary().put("PrevSearchIndexPage",currentPage-1);
70   %>  
71    <a href="#URL(Action('ViewScBlogFFSearch-ShowSearchIndexResult'),
72         Parameter('SearchParameter',
73         SearchResult:Query:ExtendedPage(PrevSearchIndexPage)))#">
74        Prev
75       </a>
76   <%
77  }
78
79  // link to first page
```

```
80 if (currentPage - maxNumberOfPagesToDisplay > 1)
81 {%>  <a href="#URL(Action('ViewScBlogFFSearch-ShowSearchIndexResult'),
82                     Parameter('SearchParameter',
83                     SearchResult:Query:ExtendedPage("(Integer)1")))#">1</a>&
                          nbsp;...
84  <%
85 }
86
87 for (int i = currentPage-maxNumberOfPagesToDisplay;
88   i <= currentPage+maxNumberOfPagesToDisplay; i++)
89 {
90    if (i < 1 || i > pageCount)
91    {
92      continue;
93    }
94    getPipelineDictionary().put("SearchIndexPage",i);
95
96    // no link for the currentPage
97    if (i == currentPage)
98    {
99       %><span class="count">
100        [ <isprint value="#SearchIndexPage#"> ]
101        </span>
102      <%
103   }
104   // provide a link
105   else if (i <= pageCount)
106   {
107      %> 
108      <a href="#URL(Action('ViewScBlogFFSearch-ShowSearchIndexResult'),
109           Parameter('SearchParameter',
110           SearchResult:Query:ExtendedPage(SearchIndexPage)))#">
111        <isprint value="#SearchIndexPage#">
112        </a>
113      <%
114   }
115 }
116
117 // link to last page
118 if (currentPage + maxNumberOfPagesToDisplay < pageCount)
119 {
120   %>  ... 
121   <a href=
122     "#URL(Action('ViewScBlogFFSearch-ShowSearchIndexResult'),
123     Parameter('SearchParameter',
124      SearchResult:Query:ExtendedPage(SearchResult:Pageable:PageCount)))#">
125     <isprint value="#SearchResult:Pageable:PageCount#">
126     </a>
127   <%
128 }
129
130 // link to next page
131 if (currentPage < pageCount)
132 {
133   getPipelineDictionary().put("NextSearchIndexPage",currentPage+1);
134   %>  <a href="#URL(Action('ViewScBlogFFSearch-ShowSearchIndexResult'),
135                       Parameter('SearchParameter',
136                        SearchResult:Query:ExtendedPage(NextSearchIndexPage)
                             ))#">Next</a>
```

```
137     <%
138 }
139 %>
140 </ul>
141 </div>
```

Listing A.5: BlogPostsIndexPaging.isml

Anhang B

Messwerte

FACT-Finder

Beiträge	***Indizierung***				
	1	**2**	**3**	∅	σ
2000	887,5	820	759,9	822,45	63,8
4000	1411,1	1410,8	1471,4	1431,1	34,9
8000	2878,4	2764,9	2610,1	2751,1	134,7
16000	5404,9	5348,9	5510,9	5421,6	82,3
32000	10629,7	10759,9	10722,1	10703,9	66,9
64000	21246,1	21309,2	21429,2	21328,2	93
128000	43799,9	42884	42929,9	43204,6	516,1
256000	278513,1	276046,6	271580,6	275380,1	3513,9
512000	579442,4	561653,1	576001,8	572365,8	9435,6
1024000	1139613,3	1174231,3	1213422,2	1175755,6	36928,1

Tabelle B.1: Indizierungsdauer in Abhängigkeit von der Beitragsanzahl

Solr

Beiträge	***Indizierung***				
	1	**2**	**3**	∅	σ
2000	3168,4	2858,8	2814,6	2947,3	192,8
4000	5509,8	5575,8	5537,7	5541,1	33,1
8000	11318,9	10843,2	10722	10961,4	315,5
16000	22098,1	21863,5	22451,7	22137,8	296,1
32000	42999	42774,5	42313,4	42695,6	349,5
64000	87091,3	86291,7	86918,9	86767,3	420,8
128000	175452,6	174465,9	173911,4	174609,9	780,6
256000	535553,6	526736,9	539500,9	533930,5	6534,9
512000	1079521,1	1095775,5	1072245,8	1082514,1	12047
1024000	2200640	2197496,7	2196172,6	2198103,1	2294,6

Tabelle B.2: Indizierungsdauer in Abhängigkeit von der Beitragsanzahl

Beiträge	Indexgröße in MB			Overhead gegenüber Rohdaten in %	
	Rohdaten	FACT-Finder	Solr	FACT-Finder	Solr
2000	4,08	8,74	7,74	114	79
4000	8,17	14,4	14,6	76	90
8000	16,3	25,8	28,3	58	74
16000	32,7	48,7	78,2	49	139
32000	65,4	95,1	155	45	137
64000	130	186	310	43	138
128000	261	370	612	42	134
256000	523	736	1225	41	134
512000	1020	1430	2380	40	133
1024000	2040	2860	4760	40	133

Tabelle B.3: Vergleich der Indexgrößen in Abhängigkeit von Beitragsanzahl

Literaturverzeichnis

[Int11a] Intershop Communications AG. *Enfinity Suite 6 – Technical Introduction*, Juni 2007.

[Int11b] Intershop Communications AG. Annual Report 2010. `http://www.intershop.de/investoren-jaehrliches-dokument.html?file=tl_files/media/downloads/de/investors/jaehrliches-dokument/financial-reports/2010-Annual-Report.pdf`, 2010. [Online; abgerufen am 19.09.2011].

[ESSe10] Intershop Communications AG. *Enfinity Suite 6 – Advanced Search Module*, Mai 2010.

[P6410] Intershop Communications AG. *Enfinity Suite 6.4 - Application Programming Guide*, Januar 2010.

[E64F10] Intershop Communications AG. Enfinity Suite 6.4 Feature List. `https://support.intershop.com/sws/index.php/Deliver/Attachment/215P09/EnfinitySuite6.4_FeatureList.pdf`, 2010. [Online; abgerufen am 10.11.2011].

[E6410] Intershop Communications AG. *Introducing Enfinity Suite 6.4*, Januar 2010.

[IES11] Intershop Communications AG. Die E-Commerce-Lösung erfolgreicher Online-Händler. `http://www.intershop.de/ecommerce-software-enfinity-suite.html`, 2011. [Online; abgerufen am 19.09.2011].

[Bli01] Ralf Blittkowsky. Intelligente Suchlösungen fürs Intranet. `http://www.blittkowsky.net/Suchloesungen.php`, 2001. [Online; abgerufen am 03.07.2011].

[BYRN11] Ricardo A. Baeza-Yates und Berthier A.Ribeiro-Neto. *Modern Information Retrieval - the concepts and technology behind search*. Pearson Education Ltd., Harlow, England, 2. Auflage, 2011.

[CP99] Fabio Crestani und Gabriella Pasi. Soft Information Retrieval: applications of fuzzy set theory and neural networks. In *Neuro-fuzzy Techniques for Intelligent Information Systems*, Seiten 287–315, University of Glasgow, 1999. Physica Verlag.

[BV10] Bundesverband des Deutschen Versandhandels. Bedeutende Umsatzsteigerungen in 2010 - Interaktiver Handel weiter auf Erfolgskurs, Boom dauert an, Februar 2011.

[dot10c] dotSource. dotSource - die E-Commerce Agentur. `http://www.dotsource.de/`, 2010. [Online; abgerufen am 19.09.2011].

[dot10b] dotSource. Meilensteine unserer Geschichte. `http://www.dotsource.de/die-agentur/agenturgeschichte/`, 2010. [Online; abgerufen am 19.09.2011].

[SC10] dotSource. SCOOBOX 2.0 für Intershop Enfinity Suite 6. `http://www.dotsource.de/uploads/flyer/SCOOBOX_IntershopES6_Feature-Liste_Januar_2011_DE.pdf`, 2010. [Online; abgerufen am 19.09.2011].

[dot10a] dotSource. Wer ist dotSource? `http://www.dotsource.de/uploads/flyer/110406_dS_fo_image_2011_d_web.pdf`, 2010. [Online; abgerufen am 19.09.2011].

[Dre08] Dipl. Ingr. Helmut Dreßler. Fuzzy Information Retrieval. *Information*, (59):341–352, Juli 2008.

[Dro10] Isabel Drost. Solr für Einsteiger. *Java Magazin*, Seiten 32–36, Juli 2010.

[Fer03] Dr. Reginald Ferber. *Information Retrieval - Suchmodelle und Data-Mining-Verfahren für Textsammlungen und das Web*. Dpunkt, 1. Auflage, März 2003.

[Ind11] The Apache Software Foundation. Apache Lucene - Index File Formats. `http://lucene.apache.org/java/3_5_0/fileformats.html`, November 2011. [Online; abgerufen am 01.12.2011].

[Sco11] The Apache Software Foundation. Apache Lucene - Scoring . `http://lucene.apache.org/java/3_5_0/scoring.html`, November 2011. [Online; abgerufen am 01.12.2011].

[Fuh11] Prof. Dr.-Ing. Norbert Fuhr. Einführung in Information Retrieval. Skriptum zur Vorlesung, Universität Duisburg-Essen, April 2011.

[FFe10] Omikron Data Quality GmbH. Die leistungsstarke Lösung für Suche und Navigation in Online-Shops, 2010.

[Sol11] Omikron Data Quality GmbH. Conversion-Technologien für mehr Umsatz im eCommerce. `http://www.fact-finder.de/Loesungen.html`, 2011. [Online; abgerufen am 25.06.2011].

[FACT11] Omikron Data Quality GmbH. Die Algorithmen. `http://www.fact-finder.de/Die-Algorithmen.html`, 2011. [Online; abgerufen am 25.06.2011].

[Int11] Omikron Data Quality GmbH. FACT-Finder–Integration, Januar 2011.

[Omi11] Omikron Data Quality GmbH. Historie. `http://www.omikron.net/Historie.html`, 2011. [Online; abgerufen am 25.06.2011].

[Got10] Dr. Thomas Gottron. Information Retrieval. Vorlesungsskript, Johannes Gutenberg-Universität Mainz, 2010.

[Hau10] Matthias Haupt. Erfolgsfaktoren des E-Commerce im Hinblick auf die Erzeugung viraler Schleifen in Liveshopping-Portalen am Beispiel der Umsetzungsmöglichkeiten für Preisbock. Diplomarbeit, Institut für Informatik, Friedrich-Schiller-Universität Jena, Juni 2010.

[HCL03] Yih-Jen Horng, Shyi-Ming Chen und Chia-Hoang Lee. A New Fuzzy Information Retrieval Method Based on Document Terms Reweighting Techniques. *Information and Management Sciences*, 14(4):63–82, 2003.

[Hof10] Peter Hofman. Information Retrieval Seminar: Phonetische Suche. Seminar, Johannes Gutenberg–Universität Mainz, Juli 2010.

[Sol09] Lucid Imagination. *LucidWorks for Solr Certified Distribution Reference Guide*, 2009. Version: 1.4.

[Str10] Lucid Imagination. E-commerce Search Strategies. Technischer Bericht, Juli 2010.

[Soc11] Lucid Imagination. Applying Social Strategies for Search with LucidWorks Enterprise. Technischer Bericht, Februar 2011.

[Ing07a] Grant Ingersoll. Search smarter with Apache Solr, Part 1: Essential features and the Solr schema - Indexing, searching, and faceted browsing with Solr. `http://www.ibm.com/developerworks/java/library/j-solr1/index.html`, Mai 2007. [Online; abgerufen am 09.10.2011].

[Ing07b] Grant Ingersoll. Search smarter with Apache Solr, Part 2: Solr for the enterprise - Administration, configuration, and performance. `http://www.ibm.com/developerworks/java/library/j-solr2/index.html`, Juni 2007. [Online; abgerufen am 09.10.2011].

[Ing08] Grant Ingersoll. What's new with Apache Solr. `http://www.ibm.com/developerworks/java/library/j-solr-update/#mlt`, November 2008. [Online; abgerufen am 09.10.2011].

[Lew05] Dirk Lewandowski. *Web Information Retrieval : Technologien zur Informationssuche im Internet*. DGI, 2005.

[Mil03] Mark Miller. Scaling Lucene and Solr. `http://www.lucidimagination.com/content/scaling-lucene-and-solr`, Januar 2003. [Online; abgerufen am 09.10.2011].

[Min09] Jonathan Minder. Übersicht Lucene und solr. `http://drupal.ayalon.ch/book/export/html/8`, März 2009. [Online; abgerufen am 09.10.2011].

[Doc11] Omikron Data Quality GmbH. *FACT-Finder - Betriebsdokumentation*, April 2011. Version: 6.6.1.

[Ric07] Alexander Richter, Michael Koch und Jochen Krisch. Social Commerce – Eine Analyse des Wandels im E-Commerce. Bericht, Universität der Bundeswehr München - Fakultät für Informatik, 2007.

[Rod10] Lisbeth Rodriguez. Apache Lucene. `http://wiki.computerwoche.de/doku.php/suchmaschinen/lucene`, Februar 2010. [Online; abgerufen am 09.10.2011].

[Sch04] Siegfried Schüle. Qualitätssicherung von Suchmaschinen. Diplomarbeit, Hochschule für Gestaltung, Technik und Wirtschaft, Fachhochschule Pforzheim, März 2004.

[See09] Yonik Seeley. Faceted Search with Solr. Technischer Bericht, Lucid Imagination, Februar 2009.

[Peg01] Pegasus Business Software. eCommerce and the Pegasus Gateway Enabling Business-to-Business eCommerce. Whitepaper, Departement of Computing Science, 2001.

[Ste09] Tobias Steinicke. Apache Solr. `http://wiki.computerwoche.de/doku.php/suchmaschinen/solr`, April 2009. [Online; abgerufen am 09.10.2011].

[Sto07] Wolfgang G. Stock. *Information Retrieval - Informationen suchen und finden.* Oldenbourg Wissenschaftsverlag, 1. Auflage, 2007.

[Ver05] Jay Verkuilen. Assigning Membership in a Fuzzy Set Analysis. In Jay Verkuilen (Hrsg.), *Sociological Methods and Research*, Seiten 462–496. Mai 2005.

[Rij79] C. J. van Rijsbergen. *Information Retrieval.* Butterworth, 1979.

[Zoe01] Henrik Zölzer. Erfolgsfaktoren von Online-Shops. Diplomarbeit, Institut für Wirtschaft, Fachhochschule Nordostniedersachsen in Lüneburg, September 2001.

Abbildungsverzeichnis

Tabellenverzeichnis

Erklärung

Ich erkläre, dass ich die vorliegende Diplomarbeit selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe.

Jena, den 15. Dezember 2011