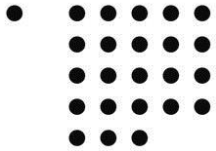

Fachhochschule Köln
Cologne University of Applied Sciences

Entwicklung eines Systems zur automatisierten Extraktion von Metadaten und dessen prototypische Anwendung innerhalb des Cloud-Speichers OX Drive

Masterarbeit

Vorgelegt an der Fachhochschule Köln

Campus Gummersbach

im Studiengang Software Engineering

ausgearbeitet von:

Kevin Ruthmann

Erstprüferin oder Erstprüfer: Prof. Dr. Heide Faeskorn-Woyke

Zweitprüferin oder Zweitprüfer: Dipl.–Ing. Thorben Betten

Gummersbach, im Juli 2015

Abstrakt

Die Menge an Informationen steigt seit Jahren immer weiter an. Dies lässt sich auch leicht an der Entwicklung der Speichermedien feststellen. So bot die erste 5,25-Zoll Festplatte, eine Seagate ST-506, lediglich 5 MB Speicherkapazität. Heutige 3,5-Zoll Festplatten verfügen hingegen über bis zu 8 TB Speicherkapazität und werden ebenso ausgenutzt wie ihre Vorgänger aus der Anfangszeit der Magnetfestplatten. Zusätzlich geht die Tendenz dorthin, alle Daten jederzeit zur Verfügung zu haben. Sei es daheim am Rechner, auf der Arbeit oder per Tablet oder Smartphone unterwegs, dank der immer mehr verbreiteten Cloud-Speicher stehen die Daten jederzeit zur Verfügung. Mit dem enormen Zuwachs an Dateien und auch an Dateiformaten wird es jedoch immer schwieriger, diese Masse zu überblicken und bestimmte Inhalte in annehmbarer Zeit wiederzufinden. Beispielsweise hostet der Internetdienst Flikr die schier unüberschaubare Menge von über 6 Milliarden Bilder. Doch nicht nur die großen Servicedienstleister besitzen große Datenmengen, auch Einzelpersonen haben derweil große Musik- und Bildsammlungen, zumal jedes aktuelle Smartphone über eine Kamera verfügt. Jeder ist somit praktisch zu jeder Zeit in der Lage, ein Foto in hochauflösender Qualität zu schießen und direkt in seine Cloud hochzuladen.

Diese Datenmengen manuell zu ordnen, erfordert einen sehr hohen Aufwand, den nicht alle Menschen gewillt sind zu leisten. Vor allem am Smartphone geht dieses Benennen und Einsortieren aufgrund der vorhandenen Technik nicht so leicht von der Hand. In der Praxis sammeln sich die Aufnahmen mit der Zeit immer weiter an und letztlich befinden sich mehrere hundert wenn nicht gar tausend Bilder in einem Ordner, welche sich namentlich meist nur durch eine fortlaufende Nummer unterscheiden. Diesen Umstand Rechnung tragend, treten Metainformationen immer mehr in den Vordergrund. So speichern die zuvor genannten mobilen Alleskönner meist viele informative Daten mit in den Bilddateien ab. Beispielsweise kann dank der eingebauten GPS-Module der Ort der Aufnahme aus den Bildern ausgelesen werden. Die Dienstleister für Cloud-Speicher nutzen diese Informationen jedoch nur marginal aus und bieten dem Endanwender kaum Unterstützung bei der Suche nach bestimmten Inhalten, wie etwa beim OX Drive, der Cloudlösung der Firma Open-Xchange.

Die vorliegende Master Thesis zeigt, wie dieser Cloud-Speicher, welcher in die Hauseigene OX App Suite integriert ist, um sogenannte Smartfeatures erweitert werden kann. Diese Smartfeatures sollen dem Endanwender helfen, die Daten einfacher – wenn nicht gar automatisch – zu ordnen und somit leichter bestimmte Inhalte wiederzufinden. Kernthema dieser Arbeit ist daher die automatische Extraktion von unterschiedlichen Metadaten aus diversen Dateiformaten. Des Weiteren wird gezeigt, wie diese Daten effizient gespeichert und abgefragt werden können. Die Thesis stellt hierzu den Document Store Elasticsearch vor und vergleicht diesen mit seinem Konkurrenten Apache Solr.

Inhaltsverzeichnis

1 Einleitung

Die Entwicklung der Endgeräte in den letzten zehn Jahren hat gezeigt, dass der Fokus auf einer immer weiter ansteigenden Mobilität liegt. Angefangen mit dem ersten iPhone aus dem Jahr 2007 explodierte der Markt für Smartphones, welche in kürzester Zeit die bis dato gängigen Handys ablösten. Wie eine Statistik von Statista zeigt[1], sinken die Verkäufe für Desktop-PCs und Laptops, wohingegen die Verkäufe von Tablets ansteigen. Bereits heute werden mehr Tablets als Laptops oder Desktop-PCs verkauft. Mit den Wearables scheint sich dieser Trend fortzusetzen. Die Kehrseite der Medaille ist dabei jedoch die erhöhte Anzahl an Endgeräten pro Anwender. Es ist heutzutage gängig, neben einem PC auch ein Smartphone, Tablet usw. zu besitzen und es ist daher oft erforderlich, Dateien, und insbesondere Medien, zwischen diesen Geräten auszutauschen. So wollen die Bilder des letzten Urlaubs sowohl auf dem Smartphone, als auch auf dem Smart TV gezeigt werden oder man möchte seine eigene Musiksammlung von jedem Gerät aus erreichen.

Schnell wurde mit Cloud-Speicher eine Antwort auf dieses Problem gefunden. Mittlerweile existiert eine große Menge von Anbietern für einen solchen online Speicher, jedoch bieten nur die wenigsten Komfortfunktionen an, um die steigenden Datenmengen zu verarbeiten. Dies trifft ebenso auf das OX Drive zu, ein in die OX App Suite integrierten Cloud-Speicher, welcher von Open-Xchange entwickelt wird. Ziel dieser Arbeit ist es daher, das Auffinden von Medien innerhalb des OX Drives für den Endanwender zu erleichtern. Als Grundlage hierfür soll ein System entwickelt werden, welches Metadaten aus diversen Dateiformaten extrahiert und abspeichert. Am Ende soll mithilfe dieses Systems ein Prototyp entstehen, welcher die OX App Suite um Komfortfunktionen – sogenannte Smartfeatures – erweitert. Im Kern untersucht diese Arbeit dabei alle Aspekte der automatischen Extraktion von Metadaten. Hierzu gehören die Identifizierung und Benennung von Dateien, dessen Analyse und die Speicherung der Metadaten.

Insbesondere werden die NoSQL Datenbanken Apache Solr und Elasticsearch als Persistierungsmöglichkeiten näher betrachtet und miteinander verglichen. Durchweg werden dabei bereits bestehende Lösungen untersucht und daraus folgernd eigene Lösungen entwickelt. Neben diesen Themen werden auch die Smartfeatures näher vorgestellt. In diesem Zusammenhang zeigt die Arbeit sowohl Möglichkeiten als auch Grenzen auf. Der Fokus liegt hierbei auf der Verwendung von Metadaten. Andere Ansätze, wie etwa die Inhaltsbasierte Suche, werden nur gestreift und dienen dazu, den verwendeten Ansatz zu bewerten und einordnen zu können. Die Arbeit ist so Aufgebaut, dass Kapitel 2 zunächst die Smartfeatures anhand von Benutzer-Szenarien vorstellt. Es werden einerseits die Funktionalitäten beschrieben und andererseits diese in einen praktischen Kontext

[1] http://de.statista.com/statistik/daten/studie/183419/umfrage/prognose-zum-weltweiten-absatz- von-pcs-nach-kategorie/ (abgerufen am 06.05.2015)

gebracht. Im darauf folgenden Kapitel wird der Begriff Metadaten genauer definiert und ein Überblick über die bestehende Datenlandschaft gegeben. Im Anschluss wird die OX App Suite und insbesondere das OX Drive vorgestellt. Kapitel 4 geht dabei auf den Aufbau der OX App Suite ein und Kapitel 5 schafft anschließend einen Überblick über die Konkurrenzprodukte des OX Drives im Hinblick auf Smartfeatures. Kapitel 6 stellt anschließend das System zur automatischen Extraktion von Metadaten vor. Inhaltlich beschäftigt sich das Kapitel in erster Linie mit den verschiedenen Hürden eines solchen Systems und zeigt dabei jeweils mögliche Lösungen auf. Dieses Kapitel beantwortet auch viele der eingangs gestellten Fragestellungen. Bevor die Implementation des Systems und insbesondere dessen Integration in die OX App Suite in Kapitel 8 beschrieben wird, erfolgt zuvor in Kapitel 7 eine Exkursion zu den Themen Sicherheit und Privatsphäre. Dieses betrachtet die Speicherung von Metadaten in Bezug auf das wachsende Sicherheitsbewusstsein der Anwender und im Rahmen der geplanten Vorratsdatenspeicherung der Deutschen Bundesregierung, welche nach einem Entwurf des BMJV[2] ebenfalls Metadaten speichern will. Hierbei werden die Vor- und Nachteile der Smartfeatures bzw., genauer gesagt, die Offenlegung der Metadaten diskutiert. Zudem werden Möglichkeiten vorgestellt, die in Elasticsearch gespeicherten Daten vor Fremdzugriff zu schützen. Kapitel 9 stellt dann Alternativen bzw. Ergänzungen zum entwickelten System vor, welche bspw. bei mangelnder Metadatenqualität eingesetzt werden können. Dieses Kapitel dient auch der Einordnung der Verwendung von Metadaten im Gegensatz zu anderen Ansätzen. Abgeschlossen wird diese Arbeit durch ein Fazit und einen Ausblick.

[2] http://www.bmjv.de/SharedDocs/Downloads/DE/pdfs/Gesetze/RegE_Hoechstspeicherfrist.pdf?__blob=publicationFile (abgerufen am 21.06.2015)

2 Smartfeatures

Die zugrundeliegende Idee für diese Arbeit sind die sogenannten Smartfeatures. Dahinter verbergen sich Funktionalitäten, um das Auffinden von Medien bzw. Dateien zu erleichtern. Dazu gehört das Strukturieren dieser Daten und das Durchsuchen der Inhalte der Daten. In erster Linie sind damit intelligente Ordner gemeint, welche im Folgenden Smartfolder genannt werden. Smartfolder sind Ordner, welche voll- oder teilautomatisiert Dateien des Anwenders zusammenstellen. Verwendet wird hierzu das sogenannte Tagging, zu Deutsch „mit einem Etikett versehen“. Dabei werden jedem Smartfolder ein oder mehrere bestimmte Tags zugewiesen. Hierdurch zeigt dieser dann alle Dateien an, welche ebenfalls mit einem der Tags versehen sind. Die folgenden Szenarien stellen dabei typische Situationen dar, in denen die Smartfeatures genutzt werden.

2.1 Szenario 1

Max war mit seiner Freundin 2014 im Urlaub und hat die Stadt Paris besucht. Durch das Fotografieren der Pariser Sehenswürdigkeiten kamen viele Bilder zusammen. Zurück in der Heimat hat er die aufgenommenen Fotos in seine Cloud hoch geladen. Bei einem Besuch seiner Eltern im nächsten Jahr wollen diese unbedingt ein paar Schnappschüsse sehen. Da Max in der Zwischenzeit bereits wieder einige Fotos geschossen hat, sind die Fotos nicht direkt auffindbar. Dies stellt für ihn jedoch kein Problem dar, da sein OX Drive ihm direkt nach dem Upload einen Ordner mit dem Namen „Paris 2014“ angelegt hat. Er wählt den Ordner aus und hat alle Fotos seines Urlaubs zur Verfügung und kann sie so seinen Eltern präsentieren.

2.2 Szenario 2

Melanie ist ein absoluter Musikfan und nutzt seit geraumer Zeit die Möglichkeit Musik online zu kaufen. Da sie immer und überall ihre Musik hören will, legt sie die Musik in ihrer Cloud ab. Seit einiger Zeit hört sie das neue Album ihrer Lieblingsband und als sie ihre Freundin Sabrina trifft, will sie es ihr zeigen. Sie öffnet das OX Drive, geht auf den Ordner ihrer Lieblingsband und wählt dann das passende Album. Dank der automatischen Erzeugung der Ordnerstruktur findet sie das Album auf Anhieb.

2.3 Szenario 3

Melani ist später am Tag auf eine Geburtstagsparty eingeladen. Leider hat der Gastgeber versäumt Musik vorzubereiten. Da Melani für ihren guten Musikgeschmack bekannt ist, soll sie schnell eine Playlist zusammenstellen. Mit der Software des Gastgebers ist sie jedoch nicht vertraut. Sie stellt daher schnell Musik in ihrer Cloud zusammen. Anschließend verbindet sie einfach ihr Smartphone mit der Anlage und lässt die Musik im neu erzeugten Ordner abspielen.

2.4 Szenario 4

Tim ist Student der Informatik und studiert bereits seit einigen Jahren. Während seines Studiums hat er viele Dokumente angehäuft und damit er sie immer griffbereit hat, legt er sie in seiner Cloud ab. Die Dokumente sind teilweise selbst geschrieben, teilweise von seinen Dozenten verfasst und teilweise selbst recherchiert. Er befindet sich gerade in einer Vorlesung, wobei es um John von Neumann geht. Tim meint schon mal etwas über ihn gelesen zu haben und will in seiner Cloud nachsehen. Die Namen der Dokumente sind leider meist ungenau oder bestehen lediglich aus einer id. Daher macht er eine Suche in den Metadaten seiner Dokumente nach von Neumann und findet ein von ihm verfasstes Dokument über die Von-Neumann-Architektur.

2.5 Szenario 5

Konrad ist Trauzeuge seines guten Freundes Karl. Er ist daher für einige organisatorische Aufgaben verantwortlich. So muss er ein paar wichtige Musikstücke für die Hochzeit organisieren, die Einladungskarten fertig stellen und ein paar Videos vorbereiten. Da er auch unterwegs immer wieder an diesen Dingen arbeitet, speichert er die Daten in seiner Cloud ab oder nutzt die bereits dort vorhandenen Dateien, wie etwa alte Fotos oder Musik aus seiner Sammlung. Damit er die Übersicht behält, versieht er jede wichtige Datei mit dem Tag „Hochzeit2015“ und legt dafür einen intelligenten Ordner an. Dieser zeigt ihm jederzeit alle relevanten Dateien für die Hochzeit an, egal wo sie in seiner Cloud liegen.

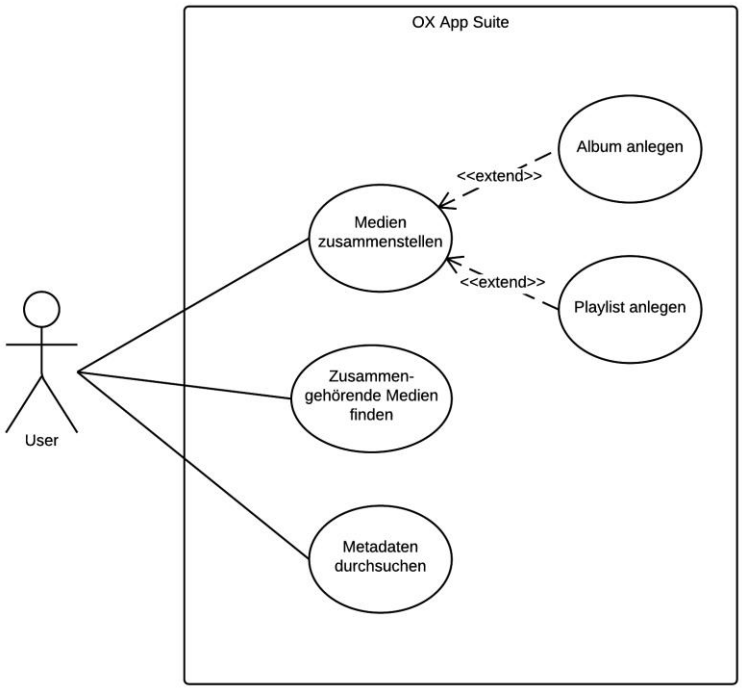

Abbildung 1 - Anwendungsfall Diagramm

Die Szenarien zeigen deutlich auf, dass die Zuordnung der Tags zu einer Datei sowohl automatisiert, als auch manuell erfolgen kann. Bei der automatisierten Variante wird je nach Dokumenttyp unterschiedlich mit der Datei verfahren, sobald sie hoch geladen wird. Bei Bilddaten wird die GPS Position ausgelesen und die Bilder dann anhand des Orts ihrer Aufnahme gruppiert. Audiodateien werden nach Artist und anschließend nach Album aufgeschlüsselt. Zudem soll jedem Dokument eine beliebige Anzahl Tags hinzugefügt werden können, mit dem diese einem der SmartFolders zugewiesen werden. Außerdem sollen die Metainformationen durchsuchbar sein, sodass zusätzlich zur Suche nach Dateinamen diese inne liegenden Informationen verwendet werden. Letztlich ergeben sich aus den Szenarien die folgenden User-Storys:

- Als Benutzer möchte ich, dass meine Bilder automatisch gruppiert werden, damit ich sie einfacher wiederfinden kann.
- Als Benutzer möchte ich, dass meine Audiodateien automatisch nach Interpret und Album gruppiert werden, damit ich sie einfacher wiederfinden kann.
- Als Benutzer möchte ich meine Dateien unabhängig vom Speicherort frei zusammenstellen können, um eigene Sichten auf meine Daten zu erhalten.
- Als Benutzer möchte ich die Metadaten meiner Dateien durchsuchen können, um Inhalte leichter zu finden.

3 OX App Suite

Am Ende dieser Arbeit sollen die in Kapitel 2 vorgestellten Smartfeatures in die OX App Suite integriert werden. Dieses Kapitel stellt daher die bestehende Software vor und zeigt ihre Struktur auf. Die OX App Suite ist eine in Java geschriebene Kollaborationssoftware und zielt damit insbesondere auf Kunden aus dem Segment der Internet Service Provider (ISP) und Telekommunikationsgesellschaften ab. Zum Funktionsumfang der OX App Suite gehört das Bearbeiten von Emails, Kontakten und Terminen. Zudem existiert mit OX Documents eine Funktionalität zum gemeinschaftlichen Arbeiten an Dokumenten und mit OX Messenger eine Möglichkeit der Kommunikation über Chat, Sprachchat und Videokonferenz. Im Fokus dieser Arbeit steht jedoch die Funktionalität OX Drive, welche das Speichern und Teilen von Dateien innerhalb der Cloud ermöglicht. Die so gespeicherten Daten können einfach mithilfe des Addons OX Guard verschlüsselt werden, welches überdies weitere Sicherheitsrelevante Features wie die Verschlüsselung von eMails mitbringt. Im Groben ist die OX App Suite somit ein Konkurrenzprodukt zu Microsoft Exchange[3], bietet jedoch einen größeren Funktionsumfang und ist im Gegensatz zur proprietären Software des Microsoftkonzerns quelloffen und unter der GPL v2 (Backend) veröffentlicht[4]. Sie verfügt ferner über die Möglichkeit, sowohl vertikal als auch horizontal zu skalieren und ist somit auch für große Zahlen von Endnutzern geeignet, da die Last auf mehrere Server bzw. sogar auf eine ganze Serverfarm aufgeteilt werden kann. Intern ist die Software als Pluginarchitektur aufgebaut. Hierzu setzt die OX App Suite auf Equinox[5] auf, einer OSGi[6] Implementierung der Eclipse Foundation.

OSGi

OSGi war ursprünglich die Abkürzung für „Open Service Gateway Initiative“ und steht heutzutage vielmehr als Kurzform für die OSGi Service Platform. Diese ist eine offene Spezifikation für ein dynamisches komponentenbasiertes System auf Basis der Sprache Java. Sie ist somit hardwareunabhängig und erlaubt es, komplexe Softwaresysteme besser zu handhaben, indem die Dienste eines solchen Systems modularisiert werden. Diese Module, im OSGI Kontext Bundles oder Services genannt, werden durch eine Service Registry verwaltet. Hierdurch lassen sich einzelne Komponenten dynamisch während der Laufzeit laden und beenden. Zudem können Abhängigkeiten zwischen den Bundles definiert werden. Entwickelt wird die OSGi Spezifikation seit 1999 von der OSGi Alliance. Zu ihren Mitgliedern zählen neben vielen kleinen Firmen auch Großunternehmen wie

[3] http://products.office.com/de-de/exchange/email (abgerufen am 14.03.15)
[4] https://code.open-xchange.com/ (abgerufen am 14.03.15)
[5] http://www.eclipse.org/equinox/ (abgerufen am 14.03.15)
[6] http://www.osgi.org/Main/HomePage (abgerufen am 14.03.15)

IBM und die deutsche Telekom. Neben der reinen Spezifikation der OSGi Service Plattform stellt die OSGi Alliance unter anderem auch eine Referenzimplementierung und Test Suits zur Verfügung.

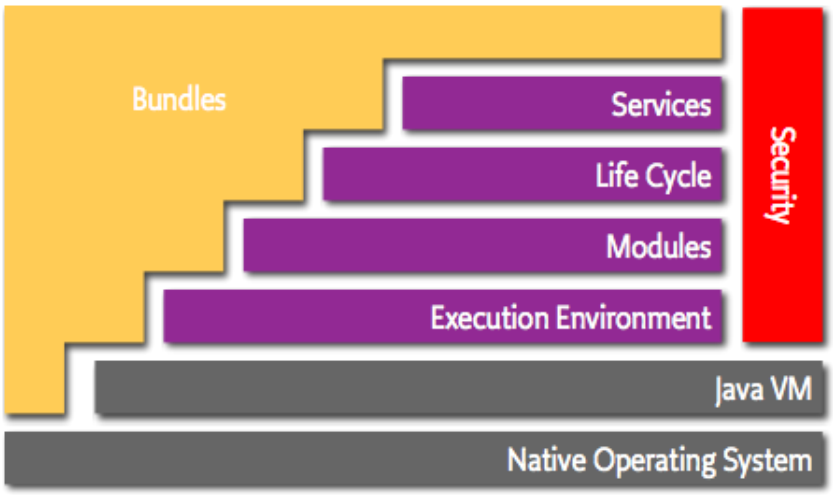

Abbildung 2 - OSGi Layers (Quelle: http://www.osgi.org/Technology/WhatIsOSGi)

Die Struktur von OSGi ist in mehrere Schichten (Layer) aufgeteilt. Die vom Entwickler entwickelten Komponenten werden Bundles genannt. Auf der Serviceebene werden diese Bundles miteinander verbunden. Dies geschieht durch einen dreiteiligen Mechanismus. Zunächst müssen sich einfache Java Objekte (POJOs) unter einer oder mehreren Schnittstellen registrieren (publish). Diese Objekte können dann von anderen Bundles gefunden (find) und anschließend benutzt werden (bind). Dabei ist es ebenfalls möglich, mehrere Objekte unter derselben Schnittstelle zu registrieren. Der Zugriff auf ein bestimmtes Objekt kann dann mittels Properties erfolgen, welche mit dem Objekt bei der Registrierung abgelegt werden. Wird ein solches Objekt (auch Service genannt) von einem Bundle genutzt, muss dieses jedoch dafür sorgen, dass sie es nicht mehr benutzt und alle Referenzen löscht, sobald das Objekt aus der Registry entfernt wird. Dies kann beispielsweise passieren, wenn ein Bundle gestoppt oder deinstalliert wird. Eine Hilfe für diese Aufgabe sind sogenannte ServiceTracker, welche das Auffinden eines Services übernehmen und auf entsprechende Ereignisse lauschen und diese verarbeiten.

Auf die Service Schicht folgt der Life Cycle Layer, welcher die verschiedenen Bundles verwaltet und zuständig für deren Start, Stop, Update und Deinstallation ist. Dieser Mechanismus ist entscheidend für einige Entscheidungen des entwickelten Systems und soll daher an dieser Stelle näher betrachtet werden (siehe Abbildung 3). Der Lebenszyklus eines OSGI Bundles kann dabei sechs Zustände durchlaufen. Als erstes muss ein solches Bundle installiert werden. Dabei wird es dem OSGI Container, einer OSGI Implementation, bekannt gemacht. Dieser versucht in einem nächsten Schritt die Abhängigkeiten aufzulösen. Sind alle Abhängigkeiten für das Bundle dem OSGi Container bekannt, so geht das Bundle in den Zustand *Resolved* über. Von hier aus kann es nun gestartet

werden. Beim Start wird die Methode start des eingetragenen *BundleActivator*'s ausgeführt. In dieser ist es dem Bundle beispielsweise möglich, Services anzufragen oder eigene Services zu registrieren. Während dieses Aufrufs befindet es sich im Zustand *Starting*. Nach erfolgreichem Ausführen dieser Methode geht das Bundle in den Zustand *Active* über. An dieser Stelle kann das Bundle dann wieder gestoppt werden. Im Zustand *Stopping* wird das Äquivalent zur Methode start vom *Activator* aufgerufen. Diese Methode ist dafür vorgesehen, die registrierten Services abzumelden und zu beenden. Im Anschluss kehrt das Bundle in den Zustand *Resolved* zurück. Zusätzlich zu den genannten Aktionen kann ein Bundle auch deinstalliert werden, wenn es sich in einem der Zustände *Installed* oder *Resolved* befindet. Zudem lässt sich ein Bundle in diesen Zuständen auch aktualisieren. Es geht dabei in den Zustand *Installed* über und es wird vom OSGi Container erneut versucht, die Abhängigkeiten aufzulösen. Die Auflösung der Abhängigkeiten wird nach einem Fehlschlag immer dann wiederholt, wenn ein neues Bundle installiert wurde. Dies sorgt dafür, dass bei Bundles mit gescheiterter Abhängigkeitsauflösung diese zu einem späteren Zeitpunkt erfolgen kann, wenn bspw. die Abhängigkeiten durch ein anderes nun installiertes Bundle erfüllt werden.

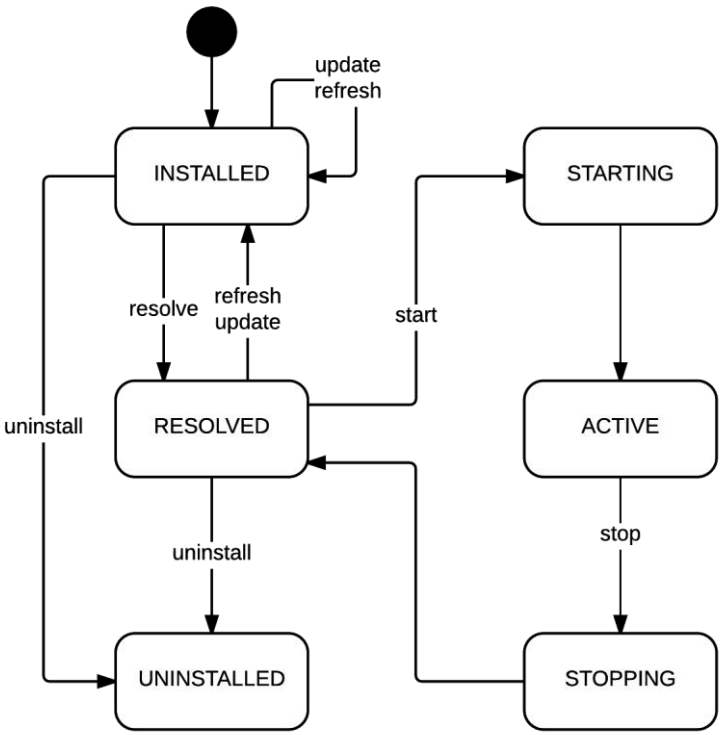

Abbildung 3 - OSGi Lifecycle

Dieses System bietet mehrere Vorteile. Einerseits wird so verhindert, dass Komponenten eingesetzt werden, welche offene Abhängigkeiten besitzen und zum anderen lassen sich damit während des Betriebs einzelne Bestandteile aktivieren, stoppen und aktualisieren. So ist beispielsweise möglich, Updates während der Laufzeit einzuspielen oder Funktionalitäten auf Bedarf zu aktivieren, ohne die

Software gänzlich neu zu starten. Dies hat insbesondere im Kontext der OX App Suite den Vorteil, dass die Endanwender keine Ausfallzeit des Servers erleben und dennoch Funktionalitäten hinzugefügt werden können. Ebenfalls ist es somit möglich, Services während des Betriebs zu tauschen. Beispielhaft würde ein Tausch eines Services folgendermaßen aussehen. Zu Beginn ist das *BundleA* im Zustand *Active* und hat in der Start-Methode seines *BundleActivator's* den Service *ServiceA* unter dem Namen CustomService registriert. Nun wurde ein alternativer Service entwickelt, welcher *ServiceA* ersetzen soll. Dieser wurde ebenfalls in ein eigenes Bundle gepackt. Dieses *BundleB* wird nun installiert und seine Abhängigkeiten werden aufgelöst. Er durchläuft den Zustand *Starting* und registriert den *ServiceB* ebenfalls unter dem Namen CustomService und geht in den Zustand *Active* über. In diesem Moment sind beide Bundles aktiv und beide Services unter CustomService registriert. Wird nun der CustomService angefragt, so wird der Service zurückgegeben, welcher das höhere Ranking besitzt. Ist das Ranking, wie in diesem Fall, gleich, so wird der Service mit der niedrigeren ID zurückgegeben[7]. In diesem Beispiel ist dies der *ServiceA*, da dieser zuerst registriert wurde. Jetzt kann das *BundleA* gestoppt werden. Es läuft dann durch die Stop-Methode des *BundleActivator*'s und meldet den *ServiceA* ab. Dann geht das *BundleA* in den Zustand *Resolved* über. Ab dem Zeitpunkt des Abmeldens des *ServiceA* erhält jeder bei einer Abfrage des CustomService nur noch den *ServiceB*. Dies geschieht ohne jedwede Unterbrechung der Funktionalitäten, welche auf CustomService aufbauen.

Auf die Life Cycle Schicht folgt dann die Modules Schicht. Diese definiert, wie Bundles ihren Code ex- und importieren können. Grundsätzlich versteckt OSGi alle Informationen zwischen den einzelnen Bundles, sodass es zum Beispiel möglich ist, dass mehrere Versionen der gleichen Bibliothek verwendet werden. Insofern ist es notwendig, dass alles explizit ex- und importiert werden muss, wenn ein anderes Bundle Zugriff erhalten soll. Letztlich sorgt die Execution Environment dafür, welche Methoden und Klassen für eine spezifische Plattform verfügbar sind. Sie ist die letzte Schicht über der Java VM und schließt damit das OSGi System ab. Parallel zu diesen Schichten existiert zudem ein Layer Security für die Sicherheit, welcher alle sicherheitsrelevanten Aspekte abdeckt.

7 https://osgi.org/javadoc/r4v43/core/org/osgi/framework/BundleContext.html#getServiceReference%28java.lang.Class%29 (abgerufen am 21.05.2015)

4 OX Drive Alternativen

Wie bereits geschildert, dienen die Smartfeatures dazu, dass Auffinden der Dateien für die Endanwender zu erleichtern. Um die Qualität solcher Funktionen zu erhöhen, ist es ratsam zu untersuchen, ob und wie die Konkurrenz solche Funktionen umsetzt. Dieses Kapitel stellt dazu verschiedenste Dienste und Applikationen vor und bewertet letztlich die aktuelle Marktsituation. Dabei unterteilt sich dieses Kapitel in zwei Abschnitte. Im ersten Teil wird ein Überblick über die Anbieter von Cloud-Speichern gegeben und die populärsten unter ihnen auf eine solche Funktionalität untersucht. Betrachtet werden dabei Lösungen für Endanwender, da sich dort solche Funktionalitäten am ehesten finden lassen. Im zweiten Teil werden Applikationen für mobile Endgeräte untersucht. Das Hauptaugenmerk liegt auf Applikationen für Bilddaten, da der Fokus des zu entwickelndes Systems auf diesen Daten liegt. Zudem kann davon ausgegangen werden, dass die meisten Applikationen für Audiodateien bereits über einen Großteil der Funktionalitäten verfügen.

4.1 Anbieter für Cloud-Speicher

Die Anbieter von Cloud-Speicher lassen sich zunächst in zwei grobe Rubriken einteilen. Es gibt die Anbieter, welche die Sicherheit der Daten und insbesondere die Privatsphäre des Anwenders in den Vordergrund stellen. Diese lassen sich daran erkennen, dass die Daten bereits beim Client vor dem eigentlichen Upload verschlüsselt werden. Man spricht hierbei von der „Zero-Knowledge Privacy“ oder auch „Zero Knowledge Policy“. Das bedeutet, dass der Anbieter kein Wissen über die Daten seiner Kunden hat, da die Daten bereits lokal beim Anwender verschlüsselt werden. Der Schlüssel ist demnach nur dem Anwender selbst bekannt und der Anbieter des Cloud-Speichers ist somit nicht in der Lage, die Daten einzusehen, selbst wenn dieser beispielsweise im Rahmen einer Strafverfolgung dazu aufgefordert würde. Beim Einsatz einer solchen client-seitigen Verschlüsselung ist es jedoch unmöglich, die Daten für den Anwender entsprechend aufzubereiten, da der Server keinen Zugriff auf die Daten besitzt und somit auch nicht in der Lage ist, die Metadaten auszulesen. Ein client-seitiges Auslesen der Daten und dessen unverschlüsselte Speicherung auf dem Server wäre ebenfalls keine Lösung, da damit das ganze Sicherheitskonzept zu einem gewissen Teil korrumpiert werden würde. Die Daten an sich wären damit zwar weiterhin geschützt, jedoch kann der Anbieter des Cloud-Speichers Informationen über die gespeicherten Daten und deren vermutlichen Inhalts in Erfahrung bringen. So ist es zum Beispiel theoretisch möglich, anhand der GPS-Daten eines Bildes zu ermitteln, um welches Motiv es sich handelt. Man kann hierbei erkennen, dass die Metadaten ebenso schützenswert sind, wie die eigentlichen Daten selbst. Das Thema Privatsphäre darf daher auch in diesem Bereich nicht vernachlässigt werden und wird im Kapitel 7 dieser Arbeit gesondert behandelt. Aus diesem Umstand lässt sich weiterhin schlussfolgern, dass Anbieter mit einer client-

seitigen Verschlüsselung keine ähnlichen Funktionalitäten anbieten können und daher nicht näher betrachtet werden müssen.

Bei der zweiten Rubrik werden die Daten hingegen erst – wenn überhaupt – serverseitig verschlüsselt. Dies setzt den Anbieter zwar einerseits in die Lage, die Daten auszulesen, andererseits kann dieser so Smartfeatures theoretisch anbieten. Er ist in der Lage die Daten für eine bessere Aufbereitung zu analysieren. Im Folgenden werden einzelne Cloud-Speicher Dienste, welche nicht client-seitig verschlüsseln, vorgestellt und mit Hinblick auf die Smartfeatures untersucht. Die anderen Anbieter, wie etwa Tresorit[8] oder Spideroak[9], werden nicht weiter betrachtet, da sie durch ihr Konzept eine völlig andere Vermarktungsstrategie verfolgen und demnach keine Konkurrenz darstellen.

4.1.1 Dropbox

Einer der bekanntesten Anbieter für Cloud-Speicher ist die Firma Dropbox Inc., welche mit ihrem gleichnamigen Produkt 2010 die Betaphase verließ. Dropbox ist einer der Marktführer und hat eine sehr hohe Nutzerzahl, so berichtete statista.com[10], dass Dropbox im Mai 2014 über 300 Millionen registrierte Nutzer weltweit besitzt. Untersucht man den Dienst mit Hinblick auf Smartfeatures, so entdeckt man eine Erweiterung von Dropbox namens Carousel[11]. Diese dient der Aufbereitung von Bildern, welche entlang einer Zeitleiste aufgereiht werden. Zudem wird die Lokalität, sofern vorhanden, angegeben. Eine Suchfunktionalität gehört jedoch nicht zum Funktionsumfang, ebenso wenig eine Option zur Sortierung nach einem anderen Kriterium. Der Anwender muss daher die nach Datum gruppierten Fotos durchlaufen, wenn er bestimmte Fotos sucht. Es ist zwar möglich, mithilfe der Zeitleiste den Zeitpunkt relativ schnell einzustellen, jedoch hilft dies nur begrenzt, wenn das Datum der Aufnahme nicht genau bekannt ist. Neben der Ansicht in der Zeitleiste und einer Option zum manuellen Anlegen von Alben verfügt Carousel noch über eine Funktionalität namens Flashback. Flashback soll den Anwender an alte vergangene Aufnahmen erinnern, indem sie ihn zu einem jeweiligen späteren Zeitpunkt erneut darauf hinweist. Für andere Medientypen sind keine besonderen Funktionen implementiert. Dies bedeutet, dass Dropbox den Funktionsumfang der Smartfeatures zwar berührt, jedoch in keinem Fall ausfüllt. Die Flashbacks sind ein Alleinstellungsmerkmal, welches auf den Metadaten der Bilder basiert. Es wäre daher theoretisch möglich, ein ähnliches Feature zu implementieren, da die Metadaten für die Smartfeatures sowieso extrahiert werden.

[8] https://tresorit.com/ (abgerufen am 31.05.2015)
[9] https://spideroak.com/ (abgerufen am 31.05.2015)
[10] http://de.statista.com/statistik/daten/studie/326447/umfrage/anzahl-der-weltweiten-dropbox-nutzer/ (abgerufen am 08.03.15)
[11] https://carousel.dropbox.com/ (abgerufen am 31.05.2015)

4.1.2 Copy

Copy ist ein relativ junger Cloud Storage Dienst, welcher sich schnell einer großen Beliebtheit erfreut hat, da er zur Zeit seiner Einführung mit seinen großzügigen 15 GB kostenlosen Speicherplatz die Konkurrenz ausstach. Inzwischen haben jedoch viele der Konkurrenten diesbezüglich gleichgezogen. Ein Vorteil bietet das 2013 veröffentlichte Produkt der Barracuda Networks durch sein interessantes Empfehlungssystem, bei dem sowohl der Empfehlende, als auch der Empfänger der Empfehlung ganze 5 GB zusätzlichen Speicher auf Lebenszeit bekommen. Da es sich hierbei um eine jüngere Software handelt, geht die Funktionalität im Hinblick auf Smartfeatures jedoch nicht über eine einfache Suchfunktion und einer Sortierung nach Erstellungs- bzw. Veränderungsdatum hinaus. Metadaten werden nicht ausgelesen und verarbeitet.

4.1.3 OneDrive

OneDrive oder ehemals SkyDrive ist der Cloud-Speicher Dienst des Software-Riesen Microsoft. OneDrive zeichnet sich vor allem durch seine starke Integration in Microsofts inzwischen plattformübergreifendes Betriebssystem aus. Ähnlich wie Dropbox setzt OneDrive auf eine starke Aufbereitung der Bilddaten. So existiert ein spezieller Ordner für Bilddaten, in dem die Bilder nach Datum gruppiert und anschließend sortiert werden. In einer Infosicht lassen sich zudem Metainformationen zu einem Bild anzeigen. Dabei wird auch, falls bekannt, die Position der Aufnahme des Bildes auf einer Karte angezeigt. Hier lassen sich auch sogenannte Markierungen (im englischen „tags“) zu einem Bild hinzufügen. Diese Markierungen können als eine Art Gruppierung der Bilder angesehen werden, wobei jedes Bild dabei mehrere Markierungen besitzen kann. In einer zweiten Ansicht kann der Anwender dann eine dieser Markierungen auswählen und sich somit alle mit dieser Markierung versehenen Bilder anzeigen lassen. Zudem besteht, wie auch in Dropbox, die Möglichkeit, manuell Alben anzulegen. Eine Suchfunktion ist ebenfalls vorhanden, erlaubt jedoch nur die Suche nach Dateinamen und Markierungen. Metadaten können hingegen nicht durchsucht werden. Das OneDrive besitzt demnach zwar bereits einige der gewünschten Funktionen, ist jedoch ebenfalls kein vollständiges Äquivalent für die Smartfeatures. Die Informationen beschränken sich hier ebenfalls auf Bilddaten und erfordern entweder manuelles Eingreifen oder sind nicht durchsuchbar. Andere Medienformate werden im Bereich Smartfeatures gar nicht unterstützt.

4.1.4 Google Drive

Auch Google besitzt, wie viele große Softwareunternehmen, einen eigenen Cloud-Speicher Dienst. Diese ist in Googles weit umfassende Online-Plattform integriert und kann als Speichererweiterung der Smartphones und Tablets gesehen werden, welche das hauseigene Betriebssystem Android benutzen. In der Grundfunktionalität bietet Google Drive dabei keine besonderen Funktionalitäten

an, um Mediendateien leichter aufzufinden. Lediglich ist eine einfache Dateinamenssuche vorhanden. Konzeptionell ist Google Drive jedoch so aufgebaut, dass es mithilfe von sogenannten Apps erweitert werden kann. Die bereits verfügbare Auswahl ist hierfür sogar recht umfangreich. Somit lassen sich die Smartfeatures theoretisch durch diese Apps realisieren. Ein Beispiel im Bereich Musik ist die App Drivetunes, welche als eine Art Onlinemusikplayer dient. Sie erlaubt es mp3 und m4a Audiodateien, welche im Drive liegen, online abzuspielen. Dabei ist es irrelevant, wo diese Dateien liegen. Es erlaubt dabei die Daten nach Titel, Album und Interpret zu sortieren und zu durchsuchen. Zudem besteht die Möglichkeit, die Musik nach Interpreten zu gruppieren, wodurch der Anwender in der Lage ist, eine gefilterte Ansicht weiter zu sortieren oder zu durchsuchen. Neben den üblichen Shuffle- und Wiederholungsfunktionen sucht man nach weiteren Funktionalitäten jedoch vergeblich. So ist es nicht möglich die Musik nach anderen Kategorien zu filtern oder gar eigene Playlisten anzulegen. Zudem zeigte sich die Software fehleranfällig: so tauchten manche erwarteten Ergebnisse nicht bei jedem Ordneraufruf auf oder benötigten eine gewisse Zeit, bis sie angezeigt wurden. Ein weiterer Störfaktor ist das völlige Fehlen eines Ladebalkens, sodass der Anwender in keiner Weise darüber informiert wird, ob die Anwendung noch Daten nachlädt oder nicht.

Auch die App Drive Player unterstützt die direkte Wiedergabe von Audiodateien des Google Drives, verfolgt dabei jedoch einen anderen Ansatz. Anstatt die Daten aus der Cloud selbstständig zu identifizieren und zu laden, muss an dieser Stelle der Anwender selbst die Daten in die Playlists hinzufügen. Es werden dabei zwar Metadaten angezeigt, jedoch ist es nicht möglich, die Playlists nach diesen Kriterien zu filtern oder zu sortieren. Der Player ist eher rudimentär aufgebaut und dient lediglich als einfache Wiedergabeoption. Auch für andere Bereiche existieren Anwendungen. Beispielsweise können Metadaten (Exif-Daten) eines Bildes mit der App Jeffrey's Exif Viewer eingesehen werden. Doch auch hier kann der Anwender die Bildsammlung weder sortieren, noch nach diesen Informationen durchsuchen. Zusammenfassend kann man sagen, dass Google Drive klar auf die Verwendung von Apps setzt, diese jedoch nur zu einem geringen Teil der Idee der Smartfeatures entsprechen.

4.1.5 iCloud

Neben mobilen Betriebssystementwicklern Google und Microsoft bietet auch Apple einen eigenen Service für Cloud-Speicher mit dem Namen iCloud an. Der Dienst ist sehr stark auf die hauseigenen Plattformen begrenzt und bietet daher zusätzlich nur für Microsofts Windows einen Client an. Eine Weboberfläche sucht man vergeblich. Zudem bietet Apple nur 5 GB an freiem Speicherplatz an und es besteht keine Möglichkeit, diesen über Empfehlungen oder andere Aktionen zu erhöhen. Die Smartfolder Funktionalitäten werden zwar teilweise unterstützt, jedoch nur im Rahmen einer lokalen

Anwendung des iOS Betriebssystems. Dort lassen sich shared Albums erstellen, welche mit anderen Personen geteilt werden können. Es ist ebenfalls möglich, Inhalte der Bilddaten wie Zeitpunkt, Ort und Albumname zu durchsuchen. Die Anwendung kann zwar mit dem Cloud-Speicher verbunden werden, jedoch sind die Funktionen nur über diese App erreichbar. Zudem verfügt sie über keine automatisierte Zuordnung, sondern nur über ein manuelles Anlegen von Alben.

4.1.6 Amazon Cloud Drive

Auch der Versandhändler Amazon bietet neben seinem Business orientiertem Amazon S3 Cloud-Speicher einen Service für Endanwender an. Dieser ist auf die Sicherung von Bild und Audiodaten spezialisiert. So existiert beispielsweise mit Amazon Photos - Cloud Drive eine App für das Android-Betriebssystem, welche nur die Funktionalität erfüllt, automatisch die Bilddaten des Smartphones mit der Cloud zu synchronisieren. Diese bringt eine einfache Galerie zum Betrachten der Bilder mit, welche die Aufnahmen, ähnlich wie viele andere Applikationen nach Datum, sortiert. Eine andere Sortieroption existiert überdies nicht. Es besteht lediglich die Wahl zwischen Aufnahme- und Hochladedatum. Die Reihenfolge kann zudem auch nicht bestimmt werden.

4.2 Andere Applikationen

Da bis dato innerhalb der Anbieter für Cloud-Speicher keine wesentliche Konkurrenz existiert, ist eine Erweiterung der Analyse auf ein weiteres Gebiet notwendig. Im Folgenden werden daher nun Applikationen für den mobilen Markt betrachtet, welche grundsätzlich einen Cloud-Speicher mit Smartfeatures anreichern können. Der Fokus liegt hierbei vor allem auf Applikationen für Bilddaten, da erwartet werden kann, dass Anwendungen für Audiodaten in den meisten Fällen viele der Funktionen wie Wiedergabelisten, Suchfunktionalität und auch automatische Sortierung beherrschen. Da die Menge solcher Applikationen sehr hoch ist, wurde die Auswahl auf vier Anwendungen aus dem Playstore für das Android-Betriebssystem beschränkt.

4.2.1 Piktures - Bilder und Videos

Die App Piktures[12] bietet die Möglichkeit, die lokalen Bilder und Videos nach Datum zu sortieren und nach einem Ort zu filtern. Dazu stellt sie eine geschachtelte Liste der Orte zur Verfügung, welche im aktuell ausgewählten Verzeichnis vorkommen. Die Ortsliste ist dabei unterteilt in Länder und Städte. Somit ist der Anwender in der Lage, sich alle Bilder eines Landes oder einer Stadt anzeigen zu lassen. Eine andere Aufschlüsselung ist hingegen nicht möglich. Zudem unterstützt die Anwendung die Darstellung einzelner Bilder auf einer Karte mithilfe von Google Maps, sodass sich die genaue Position einer Aufnahme ansehen lässt. Ein Anlegen von eigenen Alben ist nur bedingt möglich, da

[12] http://piktures.diune.com/ (abgerufen am 21.05.2015)

dabei die Bilddaten lediglich in einen neuen physischen Ordner kopiert oder verschoben werden. Will man eine Aufnahme in mehreren Alben haben, so hat man zwangsläufig Duplikate dieser Aufnahme auf dem Gerät gespeichert.

4.2.2 Cyanogen Gallery

Die Cyanogen Gallery[13] stellt zusätzlich zur normalen Ansicht eine weitere Ansicht bereit, in der die Bilder in sogenannten Momenten zusammengefasst werden. Momente sind in dieser App monatliche und von der Lokalität abhängige Zusammenstellungen von Bildern. Die Qualität dieser Funktion ist jedoch fraglich, da bereits bei ersten eigenen Testläufen eigentlich zusammengehörende Aufnahmen in gleichnamige, aber unterschiedliche Momente gepackt wurden. Zudem ist ein Zugriff auf einen speziellen Moment nicht ohne Weiteres möglich. Dieser muss zunächst in der Liste der verfügbaren Momente händisch gesucht werden. Eine Suchfunktion existiert nicht. Auch eine andere Sortierung oder ein schnelles Springen zu einem bestimmten Zeitpunkt ist nicht möglich.

4.2.3 A+ Galerie Fotos & Videos

Die A+ Galerie[14] zeigt die Bilddaten in drei Ebenen an. Auf der obersten Ebene werden die Bilder in Jahren gruppiert und als sehr kleine Vorschaubilder angezeigt. Hat man sich für ein Jahr entschieden, zoomt die Applikation hinein und zeigt nun Sammlungen von Bildern an. Diese Sammlungen gruppieren eng beieinander liegende Bilder und schaffen es bis zu einem gewissen Grad, zusammenhängende Ereignisse zu erkennen. Wählt man nun eine Sammlung aus, so werden die Bilder noch feiner in sogenannten Momenten aufgeschlüsselt. Die Momente fassen dabei Bilder eines einzelnen Tages zusammen. Auch hier existiert jedoch keine Möglichkeit der Suche oder Sortierung.

4.2.4 Fotogalerie

Die Fotogalerie[15] App ist ebenfalls in der Lage, die Bilddaten nach Datum zu sortieren und auf Wunsch nach Ort zu gruppieren. Eine Möglichkeit, Alben zu erstellen, sucht man jedoch ebenso vergeblich, wie eine Suchfunktion. Auch die Sortierreihenfolge ist starr festgelegt und lässt sich nicht anpassen.

4.3 Zusammenfassung OX Drive Alternativen

Letztlich unterstützt kein bekannter Anbieter für Cloud-Speicher den vollen Funktionsumfang der Smartfeatures. Zwar existiert die Dropbox-Erweiterung Caroussel, welche einen Teil der Funktionen

[13] https://play.google.com/store/apps/details?id=com.cyngn.gallerynext&hl=de (abgerufen am 31.05.2015)
[14] https://play.google.com/store/apps/details?id=com.atomicadd.fotos&hl=de (abgerufen am 31.05.2015)
[15] https://play.google.com/store/apps/details?id=com.appmee.gallery&hl=de (abgerufen am 31.05.2015)

anbietet, jedoch entspricht der Umfang nur einem kleinen Teil der Smartfeatures. Ähnlich verhält es sich beim OneDrive. Andere Cloud-Speicher verfügen entweder über gar keine Smartfeatures oder sie sind auf zusätzliche Anwendungen angewiesen. Resümierend kann man sagen, dass kein Cloud-Speicher über eine in die Applikation integrierte Lösung verfügt, welche die Smartfeatures voll umfassend abbildet. Überdies hinaus existieren zwar Applikationen, insbesondere für den mobilen Sektor, welche Teile der Smartfeatures abbilden. Diese zusätzliche Software beschränkt sich dabei jedoch immer auf das jeweilige Endgerät. Sie arbeiten lediglich lokal und dies sorgt dafür, dass beispielsweise erstellte Alben nicht auf allen Geräten zur Verfügung stehen. Zudem bieten die Anwendungen insbesondere im Bereich Suchen und Sortieren nicht den vollen Funktionsumfang der Smartfeatures.

5 Metadaten

Im Rahmen dieser Arbeit wird gezeigt, wie mithilfe der Extraktion von Metadaten die Smartfeatures aufgebaut werden können. Bevor jedoch näher auf die diversen Dateiformate und deren Daten eingegangen werden kann, muss zunächst geklärt werden, was Metadaten überhaupt sind. Das Wort Metadaten setzt sich aus dem Substantiv „Daten“ und dem Präfix „Meta“ zusammen. Meta, ursprünglich aus dem griechischen stammend, hat gemäß Duden die Bedeutung: „drückt in Bildungen mit Substantiven aus, dass sich etwas auf einer höheren Stufe, Ebene befindet, darüber eingeordnet ist oder hinter etwas steht“[16]. Metadaten sind demnach Daten, die anderen Daten übergeordnet sind. Allgemeinhin können Metadaten als Daten über Daten interpretiert werden. Konkret bedeutet dies, dass Metadaten andere Daten beschreiben, diese selbst aber nicht beinhalten. Ein Textdokument kann man beispielsweise durch Autor, Datum und Thema beschreiben. Wie die Daten beschrieben werden, hängt dabei stark vom Typ der Daten ab. Ein Textdokument wird zum Beispiel anders beschrieben als ein Bild und dies wiederum anders als eine Audiodatei. Die eingetragenen Informationen können dabei vielseitig und umfangreich sein.

So zeigt Abbildung 4 beispielhaft eine Auswahl von Metadaten eines Bildes. Die Informationen reichen vom Modell der Kamera, über die Auflösung des Bildes bis hin zur verwendeten ISO-Einstellung. Die Menge an Informationen in den Metadaten ist theoretisch unbegrenzt und wird in manchen Fällen lediglich durch die verwendete Technik limitiert. So konnten bei den ersten Metadaten für MP3-Dateien durch das verwendete Format nur begrenzt Daten abgespeichert werden. In der heutigen Zeit jedoch werden zumeist generische Ansätze verwendet, sodass theoretisch beliebig viele Informationen in Schlüssel-Wert-Paaren abgelegt werden können. Ein Beispiel für ein Schlüssel-Wert-Paar wäre der Schlüssel „Image Height“ und der Wert „2448 pixels“ aus Abbildung 4. Es existieren zwar viele Möglichkeiten, Metadaten mitsamt den Dateien zu speichern, jedoch existiert in diesem Bereich immer noch ein fundamentales Problem. Die Metadaten sind nur sinnvoll und praktisch, wenn ihre Daten auch stimmen. So ist es auch heute noch oft der Fall, dass diese Informationen entweder nicht vollständig oder gar falsch sind. Insbesondere bei Informationen, welche manuell von Nutzern eingetragen werden müssen, können sich Fehler einschleichen und es ist meist unmöglich, diese festzustellen. Beispielsweise reicht es aus, dass der Anwender sich beim Eintragen des Künstlers einer MP3 vertippt hat, um die Datei nicht mehr diesem Künstler zuordnen zu können. Es ist nicht mehr leicht zu erkennen, ob sich die Person vertippt hat oder ob das Stück zu einem Künstler gehört, welcher sich sehr ähnlich schreibt.

Inzwischen werden zwar auch viele Informationen nicht mehr durch den Anwender eingetragen, sondern durch die verwendete Software, doch existieren in diesem Segment eine Vielzahl von

[16] http://www.duden.de/rechtschreibung/meta_ (abgerufen am 10.03.15)

Anwendungen, welche nicht immer die gleichen Informationen eintragen. Dieser Umstand lässt sich jedoch insbesondere für Bilddaten nicht ändern. Vielmehr muss man sich bei der Entwicklung eines Systems, welches mit diesen Daten arbeitet, bewusst sein, dass die Daten unkorrekt und/oder unvollständig sein können. Ein solches System kann daher nur so gut sein, wie die Daten, mit denen es gefüttert wird. Und es muss in der Lage sein, auch mit unvollständigen Daten zu arbeiten, da diese Fälle in der Praxis durchaus auftauchen können. Das Kapitel 9 beschäftigt sich daher mit Alternativen und Ergänzungsmöglichkeiten zur Verwendung von Metadaten und stellt Lösungsmöglichkeiten für diese Problematik vor.

```
[Jpeg] Compression Type - Baseline
[Jpeg] Data Precision - 8 bits
[Jpeg] Image Height - 2448 pixels
[Jpeg] Image Width - 3264 pixels
[Exif SubIFD] Exposure Time - 1/752 sec
[Exif SubIFD] F-Number - F2,6
[Exif SubIFD] Exposure Program - Aperture priority
[Exif SubIFD] ISO Speed Ratings - 80
[Exif SubIFD] Exif Version - 2.20
[Exif SubIFD] Date/Time Original - 2015:03:08 12:38:21
[Exif SubIFD] Date/Time Digitized - 2015:03:08 12:38:21
[Exif SubIFD] Components Configuration - YCbCr
[Exif SubIFD] Shutter Speed Value - 1/752 sec
[Exif SubIFD] Aperture Value - F2,6
[Exif SubIFD] Brightness Value - 2003/256
[Exif SubIFD] Exposure Bias Value - 0 EV
[Exif SubIFD] Max Aperture Value - F2,6
[Exif SubIFD] Metering Mode - Center weighted average
[Exif SubIFD] Flash - Flash did not fire
[Exif SubIFD] Focal Length - 3,7 mm
[Exif SubIFD] FlashPix Version - 1.00
[Exif SubIFD] Color Space – sRGB
[Exif SubIFD] Exif Image Width - 3264 pixels
[Exif SubIFD] Exif Image Height - 2448 pixels
[Exif SubIFD] Exposure Mode - Auto exposure
[Exif SubIFD] White Balance Mode - Auto white balance
[Exif SubIFD] Scene Capture Type - Standard
[Exif SubIFD] Unique Image ID - ZKFJ03
[Exif IFD0] Make - SAMSUNG
[Exif IFD0] Model - GT-N7100
```

Abbildung 4 - Beispielhafte Metadaten eines Bildes

6 Automatische Metadaten Extraktion

Seit der Einführung digitaler Medienformate ist deren Anzahl um ein Vielfaches gestiegen. So listet beispielsweise Wikipedia über 70 verschiedene Formate alleine für Audiodateien. Dies liegt vor allem darin begründet, dass für unterschiedliche Anwendungsfälle unterschiedliche Formate entwickelt wurden. Metainformationen aus einer beliebigen Datei auszulesen ist somit nicht ohne weiteres möglich. Hierzu müssen zunächst einige Probleme gelöst werden. Zudem muss zu Beginn geklärt werden, welche Formate überhaupt über Metainformationen verfügen und welche dieser Informationen im Kontext einer Software für Endanwender wirklich sinnvoll sind. So mag der verwendete Codec einer Audiodatei für Experten, wie etwa Tontechniker, durchaus von Interesse sein, jedoch ist diese Informationen für den durchschnittlichen Anwender in den meisten Fällen nutzlos. Zusätzlich existieren Informationen, welche für den Anwender zunächst aufbereitet werden müssen. So sind etwa die GPS-Informationen in Form von Längen- und Breitengraden für den Endanwender ebenfalls ohne großen Wert. Werden diese Daten hingegen Orten zugeordnet, so sind sie dem Endanwender eine nützliche Information.

Bei der Vielzahl von Dateiformaten ist es möglich, einige für diesen Anwendungsfall auszuklammern, da diese im praktischen Alltag des durchschnittlichen Endanwenders nicht vorkommen. So lässt sich in einem ersten Schritt die über 70 Audioformate auf eine deutlich geringere Zahl reduzieren. Sämtliche andere Formate besitzen entweder keine Metainformationen oder finden im Alltag der Endanwender keine Anwendung. In den anderen Bereichen Bilder, Dokument und Videos verhält es sich ähnlich. Die verbliebenen Formate unterscheiden sich jedoch weiterhin sehr stark voneinander, sodass ein generischer Ansatz zur Extraktion nicht möglich ist. Vielmehr muss für jedes Format ein eigener Parser geschrieben werden, welcher die Metadaten extrahiert. Insgesamt lassen sich drei große Problemstellungen bei der Extraktion der Metadaten identifizieren. Bevor eine Datei überhaupt geparst werden kann, muss bekannt sein, um welches Format es sich handelt, da ansonsten kein passender Parser ausgewählt werden kann. Das nächste Problem ist der Vorgang des Parsens als solcher. Es müssen hierfür ausreichend Parser zur Verfügung stehen. Diese gänzlich selbst zu entwickeln, ist jedoch mit einem sehr hohen Aufwand verbunden, sodass hierfür eine alternative Lösung gefunden werden muss. Zuletzt stellt sich noch die Frage nach der angemessenen Speicherung der erhobenen Daten. Dies beinhaltet nicht nur die verwendete Speichertechnologie, sondern vielmehr auch eine angemessene Aggregation der unterschiedlichen Daten zum Zwecke der Vergleichbarkeit. Findet eine solche Aggregation nicht statt, so lassen sich die erhobenen Daten zwar durchsuchen, jedoch aber nur begrenzt oder mit einem erheblichen Aufwand verarbeiten.

Im Folgenden werden daher zunächst Varianten zur Identifikation von Dateiformaten vorgestellt und deren Eignung untersucht. Im Anschluss wird auf das Problem des Parsens eingegangen. Es werden

einige verfügbare Bibliotheken vorgestellt, welche eine Sammlung von Parsern zur Verfügung stellen. Dabei werden die Vor- und Nachteile dieser Bibliotheken aufgezeigt und am Ende eine Auswahl getroffen und begründet. Abgeschlossen wird das Kapitel mit einem Abschnitt über die Aggregation der Daten und einer Übersicht über die verschiedenen Möglichkeiten der Persistierung. Hier wird insbesondere auf Elasticsearch und seine Unterschiede zu Apache Solr eingegangen. Auch an dieser Stellte wird eine Auswahl getroffen und diese im Kontext der OX App Suite begründet.

6.1 Identifikation der Dateiformate

Um einen passender Parser für eine Datei auszuwählen, muss zunächst das Format der Datei bekannt sein. Eine Identifizierung des Dateiformats ist daher essentiell für die automatische Extraktion von Metadaten. Diese muss zudem zuverlässig funktionieren, da bei einem Fehler ein falscher Parser eingesetzt wird und dies gegebenenfalls zu unvorhersehbaren Fehlern führen kann. Zusätzlich empfiehlt es sich, die Parser derart robust zu entwickeln, dass diese auf falsche Daten reagieren und dies dem System zurückmelden. Anschließend könnte eine solche Datei dann erneut überprüft oder als unbekanntes Format eingestuft werden. Bei einem unbekannten Format würden dann nur einige Grundinformationen wie Dateiname und Erstellungsdatum gespeichert.

Eine erste Grundvoraussetzung für die Identifikation von Dateiformaten ist es, diese benennen und somit unterscheiden zu können. Dies hat insbesondere im Umfeld des Internets eine hohe Relevanz, da dort Dateien zwischen Nutzern oder zwischen Anwendungen getauscht werden. Dies kann dazu führen, dass das Format einer Datei verloren geht. Um dem entgegen zu wirken wurde der MIME Standard entwickelt. MIME ist die Abkürzung für "Multipurpose Internet Mail Extensions" und wurde 1996 in den RFC's 2045 bis 2049[17] definiert. Dieser Standard wurde entwickelt, um den Austausch von Dateien über eine eMail zu vereinfachen und ist eine Erweiterung des Internetstandard RFC 822[18]. Besondere Bedeutung hat hierbei der RFC 2046, welcher ein Format zur eindeutigen Benennung definiert. Dieses besteht aus einem Haupttyp, welcher den Inhalt grob festlegt, einem Untertyp, welcher dies noch weiter eingrenzt bzw. spezifiziert, und einem oder mehreren Parametern, welche feine Details einstellen können. Ein Beispiel für einen Parameter wäre die Art der Codierung eines Textes wie UTF-8. Solche Parameter sind jedoch rein optional und müssen nicht verwendet werden. Die Einteilung in Rubriken und Unterrubriken hilft zudem die unüberschaubare

[17] http://www.rfc-editor.org/rfc/rfc2045.txt
http://www.rfc-editor.org/rfc/rfc2046.txt
http://www.rfc-editor.org/rfc/rfc2047.txt
http://www.rfc-editor.org/rfc/rfc2048.txt
http://www.rfc-editor.org/rfc/rfc2049.txt
(abgerufen am 23.03.15)

[18] http://www.rfc-editor.org/rfc/rfc822.txt (abgerufen am 23.03.15)

Anzahl von Dateiformaten zu überblicken. Der RFC 2046 teilt diese zunächst in zwei grobe Richtungen ein. Er unterscheidet dabei in diskrete (discrete) und zusammengesetzte (composite) Formate. Wie die Namen der Rubriken bereits verraten, beinhalten die diskreten Dateiformate lediglich einen Typ Informationen, wohingegen die zusammengesetzten Formate aus verschiedensten Informationstypen bestehen können. Sowohl Text, Bilder, Videos und Audiodaten können in solchen Dateien enthalten sein. Diese Oberrubriken werden innerhalb des RFC's weiter unterteilt. So teilt sich die Rubrik diskrete Formate in fünf Untergruppen. Die bereits genannten Datentypen Text, Bilder usw. werden hierbei noch um den Typ Applikation ergänzt. Die zusammengesetzten Dateiformate werden hingegen in Nachrichten und Multipartdateien unterschieden. In Nachrichten können bspw. RFC 822 Nachrichten enthalten sein. Der Zweck dieser Rubrik ist in erster Linie somit die Handhabung von anderen eMails. Diese Art der Daten ist daher im vorliegenden Kontext nicht relevant, da sie vor allem im Rahmen der Kommunikation im Internet Verwendung findet. Die Multipart Formate lassen sich in vier Unterrubriken einteilen. Die Rubrik Mixed beinhaltet dabei eine gemischte Sammlung von verschiedensten Dateiformaten mit ggf. auch unterschiedlichen Inhalten. Die Rubrik Alternative hingegen stellt denselben Inhalt in verschiedensten Formaten bereit. Parallel stellt Inhalte zusammen, welche dafür gedacht sind, parallel dargestellt zu werden. Die letzte Rubrik Digest ist Mixed sehr ähnlich. Der Unterschied besteht lediglich darin, dass der standardmäßige Dateityp nicht "text/plain" ist, sondern "message/rfc822". Später wurden die sieben Hauptrubriken im RFC 2077[19] noch um eine achte Rubrik erweitert, die Formate vom Typ Modell enthält. Gemeint sind hierbei Modelle in einem dreidimensionalen Raum, wie sie etwa mit CAD-Anwendungen erstellt werden. Interessant sind für den Anwendungsfall jedoch eher nur die diskreten Medientypen.

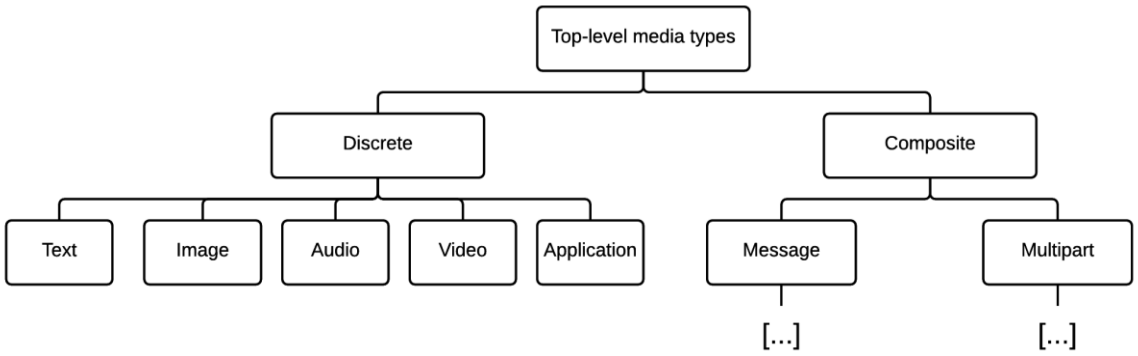

Abbildung 5 - Vereinfachte Übersicht der MIME Ober- und Untertypen nach RFC 2046

[19] http://www.rfc-editor.org/rfc/rfc2077.txt (abgerufen am 23.03.15)

Neben dem Ziel, einen Standard für die Übertragung von Inhalten über eMails zu schaffen, wurde durch den RFC 2046 auch ein Werkzeug kreiert, mit dem Dateiformate eindeutig unterschieden werden können. Dies wurde in RFC 2048 und dessen Erweiterung RFC 4288[20] verwendet, um eine Möglichkeit der Registrierung von Dateiformaten bei der Internet Assigned Numbers Authority (IANA[21]) einzurichten. Durch das einheitliche Format kann jeder dort sein Dateiformat registrieren und es somit öffentlich bekannt machen. Wird das Format einer Datei nun zu diesem Namen zugeordnet, so kann es von anderen ebenfalls zugeordneten Formaten unterschieden werden. Duplikate dieser Formate können dabei im Gegensatz zu Dateiendungen nicht auftreten, da nur eine einzelne Stelle der Registrierung existiert. Ein System zur automatischen Extraktion von Metainformationen sollte daher die Zuordnung eines Formats zu den MIME-Typen unterstützen.

In einem nächsten Schritt müssen die Dateien nun identifiziert bzw. einem MIME-Typ zugeordnet werden. Hierzu existieren mehrere Verfahren, die im Folgenden näher erläutert werden. Die naheliegendste Variante ist die Identifikation über die Dateiendung, auch Dateinamenserweiterung (im Englischen filename extension) genannt. Die Dateiendung ist dabei in aller Regel der letzte Teil eines Dateinamens und wird meistens mit einem Punkt abgetrennt. Beispiele für Dateiendungen, welche oft aber nicht ausschließlich aus drei Buchstaben bestehen, sind „.png“, „.txt“, „.mp3“. Zu beachten ist hierbei, dass ein Dateiformat unterschiedliche Endungen haben kann, wie etwa das Bildformat der Joint Photographic Experts Group, welches sowohl auf „.jpg“ als auch auf „.jpeg“ enden kann. Zudem ist nicht gegeben, dass die Endungen einzigartig sind. So können unterschiedliche Programme die gleiche Endung für ihr Dateiformat nutzen. Die Identifizierung des Formats über die Dateiendung ist die effizienteste Variante, birgt jedoch einige Hindernisse bzw. Schwachstellen und ist daher nicht ausreichend, um eine Datei eindeutig zu identifizieren. Eine andere Option ist die Analyse des Datei-Inhalts. Dies kann auf mehrere Arten erfolgen. So ist es beispielsweise möglich, die Dateistruktur zu analysieren und insbesondere oft vorkommende Headerfiles zu analysieren und abzugleichen. Weit verbreitet ist zudem die Nutzung von sogenannten Magic Numbers. Ursprünglich aus der Unix Welt kommend, definieren diese zwei Byte großen Zahlen das Dateiformat einer Datei. Hierzu werden sie entweder bewusst an den Anfang der Datei geschrieben oder sie ergeben sich durch die Umwandlung der ersten Zeichen einer Datei. So fängt zum Beispiel eine XML Datei immer mit dem Zeichensatz „<?xml“ an. Dieser Zeichensatz kann in eine Zahl umgewandelt werden und somit ebenfalls als Magic Number dienen. Die Verwendung von Magic Numbers haben jedoch auch ihre Grenzen. Bei Containerformaten wie etwa dem ogg-Format kann zwar erkannt werden, dass es sich um ein ogg Format handelt, jedoch kann der Inhalt eines solchen Containers stark variieren. Eine Analyse mittels Magic Numbers alleine bringt für einen

[20] http://www.rfc-editor.org/rfc/rfc4288.txt (abgerufen am 23.03.15)
[21] http://www.iana.org/ (abgerufen am 23.03.15)

solchen Fall somit keinen Erfolg. Eine mögliche Lösung für dieses Problem ist die Verwendung von offsets. Der Aufbau der Containerformate ist in den meisten Fällen gleich und mithilfe der Magic Numbers kann dieses Format identifiziert werden. Um anschließend den eigentlichen Inhalt zu identifizieren, werden statt der ersten beiden Bytes zwei Bytes innerhalb der Datei untersucht. Dazu wird für jedes Containerformat ein offset festgelegt, also eine Verschiebung, welche auf den eigentlichen Inhalt zeigt. Die dort befindlichen Bytes geben dann Aufschluss über den eigentlichen Inhalt.

Eine andere Variante ist die Überprüfung der ganzen Datei auf Basis dessen Spezifikation. Hierzu müssen diese jedoch in einem einheitlichen Format vorhanden sein. Eine solche Datenbank findet sich bspw. in PRONOM[22], welche von The National Archives in Großbritannien seit dem Jahr 2002 entwickelt wird. Ziel von PRONOM ist es, eine zentrale Informationssammlung über die diversen Dateiformate zu erstellen, welche im Nationalarchiv verwendet werden. Neben PRONOM existieren noch weitere Projekte einer umfangreichen Dateiformatsammlung, wie etwa das Programm „Digital Formats for Library of Congress Collections“ der Library of Congress der Vereinigten Staaten von Amerika, jedoch bietet PRONOM deutlich bessere Services in Bezug auf Formatidentifikation an. Die Identifizierung eines Formats mithilfe der Spezifikation hat den Vorteil, dass das Format der Datei nicht nur identifiziert, sondern auch validiert wird. Das bedeutet, dass sichergestellt werden kann, dass die Datei nicht korrupt ist und somit problemlos geparst werden kann. Der große Nachteil dieser Variante ist jedoch die Dauer der Identifizierung, welche im Gegensatz zu den anderen Varianten erheblich länger dauert. Die Identifizierung mithilfe der Magic Numbers scheint somit die bestmögliche Variante zu sein, da sie sowohl performant ist, als auch unabhängig von dem Dateinamen funktioniert. Die etwas langsamere Performance dieser Variante gegenüber dem Auslesen der Dateiendung ist zudem insofern zu vernachlässigen, da die Datei sowieso im Rahmen der Metadatenextraktion geöffnet wird.

6.2 Parsen der Dateiformate

Aufgrund der hohen Anzahl an Dateiformaten ist eine Nutzung von diversen Parsern unabdingbar. Eine eigene Entwicklung dieser Parser scheidet jedoch aufgrund des hohen Aufwands aus. Vielmehr bietet es sich an, bereits bestehende Parser zu nutzen oder gar bestehende Sammlungen zu verwenden. Die verwendeten Parser bzw. Bibliotheken sollten dabei einen Großteil der allgemein verwendeten Dateiformate erkennen und parsen können. Zudem sollten so viele Metadaten, wie möglich, extrahiert werden können. Ein weiterer wichtiger Aspekt ist die Möglichkeit, statt einer

[22] http://apps.nationalarchives.gov.uk/PRONOM/Default.aspx (abgerufen am 11.03.15)

Datei einen Stream zu parsen, da innerhalb des OX Drives die Dateien ebenfalls als Stream verarbeitet werden und bei einer Beschränkung auf Dateien ein Umweg über temporäre Dateien gemacht werden müsste. Zudem wäre vorzuziehen, wenn der Parser oder die Bibliothek in der Sprache Java verfasst wäre, da damit die Kompatibilität gewährleistet ist. Hervorzuheben ist an dieser Stelle die Java Bibliothek Apache Tika[23]. Apache Tika wird seit 2007 von der Apache Software Foundation entwickelt und war zu Beginn ein Unterprojekt von Apache Lucene[24]. Es wurde mit dem Ziel entwickelt, ein Werkzeug bereitzustellen, mit dem Metadaten und Text aus einer Vielzahl von Dateiformaten extrahiert und somit für Apache Lucene bereitgestellt werden konnten. Insofern entspricht Apache Tika sehr genau dem Anforderungsprofil für den hier vorhandenen Anwendungsfall. Apache Tika zeichnet vor allem sein allumfassender Ansatz aus. So ist die Metadatenextraktion nicht auf eine bestimmte Rubrik von Dateien beschränkt. Es existieren sowohl Parser für Audio-, Image- und Textfiles. Insgesamt unterstützt Apache Tika derweil 23 unterschiedliche Formatrubriken[25], welche jeweils eine Sammlung von Formaten beinhalten.

Betrachtet man die Bibliothek näher, so stellt man schnell fest, dass hier ebenfalls eine Sammlung diverser Parser die Extraktion durchführt. Zudem stellt Apache Tika die Möglichkeit bereit, die Datei automatisch zu erkennen und anschließend zu parsen. Hierzu wird eine Mischung verschiedener Detektoren verwendet. Diese Detektoren verwenden jeweils eine andere Herangehensweise, um das Format einer Datei zu bestimmen. Im Groben werden hier ebenfalls die bereits genannten Mechanismen verwendet. Sowohl für die Erkennung anhand der Dateinamensendung existiert ein Detektor, als auch für die Erkennung mittels der Magic Numbers. Neben einzelnen Detektoren für Sonderfälle wie das OLE 2 Compound Document Format findet sich ein eigener Detektor für komprimierte Daten, die bspw. im Zip oder ähnlichen Formaten vorliegen. Somit erfüllt Apache Tika viele der eingangs gestellten Anforderungen. Die Zahl der unterstützen Formate ist hierbei bereits sehr hoch, doch nicht vollständig. Beispielhaft sei hier das Advanced Streaming Format (ASF) genannt, welches von Microsoft entwickelt wurde und beispielsweise bei wma-Dateien Verwendung findet. Hingegen positiv ist die optimale Unterstützung von Streams, sodass diese direkt von Apache Tika verwertet werden können. Ein Umweg über temporäre Dateien ist somit nicht erforderlich.

Schlechter macht dies die ebenfalls in Java verfasste Bibliothek JAudiotagger[26], da sie lediglich in der Lage ist, Dateien zu parsen. JAudiotagger wird von JThink entwickelt und ist unter der GNU LGPL verfügbar. Unterstützt werden hiervon jedoch nur Audio-Formate, dessen Anzahl jedoch größer ist als bei Apache Tika. So kann mittels JAudiotagger auch das ASF-Format geparst und dessen

[23] http://tika.apache.org/ (abgerufen am 11.03.15)
[24] http://lucene.apache.org/core/ (abgerufen am 11.03.15)
[25] http://tika.apache.org/1.7/formats.html (abgerufen am 11.03.15)
[26] http://www.jthink.net/jaudiotagger/ (abgerufen am 12.03.15)

Metadaten gewonnen werden. Ein weiterer Negativpunkt ist die Erkennung des Dateiformats. Diese basiert lediglich auf der Dateiendung und kann somit zu Fehlern führen. Neben JAudiotagger existieren noch weitere auf einen Dateityp spezialisierte Bibliotheken. Eine dieser Bibliotheken ist der Metadata Extractor[27]. Dieser ist auf Bild-Formate spezialisiert und kann somit beispielsweise die Exif-Daten aus einem jpeg Bild extrahieren. Metadata Extractor wird von Drew Noakes seit 2002 entwickelt und ist ebenso wie Apache Tika unter der Apache License (Version 2) verfügbar. Bezogen auf die Breite der unterstützen Datei- und Metadatenformate ist die Situation analog zu JAudiotagger. Auch hier werden mehr Formate unterstützt als bei Apache Tika. Identifiziert werden die Formate mittels der Magic Numbers, also der Analyse der ersten Bytes der Datei. Die Dateiendung wird hingegen gar nicht beachtet. In Anbetracht der Anfälligkeit bei falschen Eingabedaten ist ein solches Vorgehen jedoch legitim.

Für Dokumente existieren hingegen keine universellen Bibliotheken in der Sprache Java. Vielmehr existieren hier Lösungen für einzelne Dateiformate oder kleinere Gruppen von Formaten. Etwa existiert mit Apache PDFBox[28] eine ebenfalls von Apache entwickelte Bibliothek zur Bearbeitung von PDF-Dokumenten, die ebenfalls Möglichkeiten zur Verfügung stellt, die Metadaten auszulesen und zu manipulieren. Ein anderes Beispiel ist Apache POI[29], welches die Microsoft Office-Formate abdeckt. Sofern man nicht die einzelnen Bibliotheken in eine einzelne Anwendung integrieren will, ist man im Bereich Java auf Apache Tika angewiesen, um Metadaten zu extrahieren. Wirkliche Alternativen finden sich nur in anderen Programmiersprachen. So wurde von GNU[30] eine Bibliothek mit dem Namen Libextractor[31] entwickelt, welche hauptsächlich in der Sprache C geschrieben ist, jedoch auch Bindings für die Sprache Java besitzt. GNU typisch wurde Libextractor unter der GNU GPL[32] veröffentlicht. Ähnlich wie Apache Tika unterstützt auch Libextractor eine große Menge an Dateiformaten aus allen Bereichen. Positiv hervorzuheben ist hierbei, dass noch mehr Formate unterstützt werden als bei Apache Tika. So ist Libextractor auch in der Lage ASF-Dateien zu analysieren. Libextractor bietet dabei keine eigene Formatidentifikation an. Stattdessen muss jedes Plugin selbst entscheiden, ob die Datei dem Format des Plugins entspricht. Hierbei wird jedoch ebenfalls oft das Verfahren auf Basis der Magic Numbers verwendet. Es kann aber nicht sichergestellt werden, dass dies immer der Fall ist. Innerhalb der Java Bindings ist eine Behandlung eines Streams zwar nicht möglich, jedoch existiert die Option, ein Byte Array zu analysieren. Insofern ließe sich ein Stream über diesen kleinen Umweg ebenfalls in einem angemessenen Rahmen nutzen. Der große

[27] https://drewnoakes.com/code/exif/ (abgerufen am 12.03.15)
[28] https://pdfbox.apache.org/ (abgerufen am 12.03.15)
[29] http://poi.apache.org/ (abgerufen am 12.03.15)
[30] http://www.gnu.org/gnu/ (abgerufen am 12.03.15)
[31] http://www.gnu.org/software/libextractor/ (abgerufen am 12.03.15)
[32] http://www.gnu.org/copyleft/gpl.html (abgerufen am 12.03.15)

Nachteil von Libextractor ist jedoch die fehlende Portabilität, welche insbesondere im Zusammenhang mit einer in Java geschriebenen Software schwer ins Gewicht fällt. Wollte man Libextractor nutzen, so müsste für jede zu unterstützende Distribution der Libextractor Code kompiliert und mitgeliefert werden. Dies bedeutet vor allem bei der Integration von neuen Libextractor Versionen, etwa nach Sicherheitsupdates oder ähnlichem, einen großen Aufwand.

Zieht man nun abschließend ein Fazit, dann kann man sagen, dass Apache Tika die beste Option darstellt. Es besteht zwar die Möglichkeit, die einzelnen Bibliotheken eigenständig zusammen zu stellen, jedoch müsste man dann einerseits ein eigenes System zur Identifikation des Dateityps entwerfen und andererseits bietet dieser Ansatz deutlich mehr Stellen, an denen Fehler auftreten können. Zudem erhöht dieser Ansatz den Aufwand zur Pflege des Systems, da so mehrere Bibliotheken auf dem neuesten Stand gehalten werden müssen. Diesen Nachteil besitzt Libextractor zwar nicht und besitzt zudem die umfassendste Formatunterstützung, ist jedoch nur mit einem hohen Aufwand in die Software zu integrieren, da sie in der Sprache C verfasst ist. Somit scheint Apache Tika den robustesten Ansatz zu bieten. Die Problematik der fehlenden Formate lässt sich zudem mithilfe eigener Parser egalisieren. Außerdem wird die Entwicklung von Apache Tika sehr aktiv weiterbetrieben, sodass in naher Zukunft damit gerechnet werden kann, dass auch die fehlenden Formate unterstützt werden. Alternativ oder zusätzlich zu dem Ansatz mittels eines eigenen Parsers ließe sich eine der anderen Java-Bibliotheken als Fallback implementieren. So könnte man zunächst versuchen, die Metadaten aller Dateien mittels Apache Tika zu extrahieren und nur bei Misserfolg, etwa beim ASF-Format, die Aufgabe JAudiotagger zu übergeben.

6.3 Speicherung der Metadaten

Im vorherigen Kapitel wurde erläutert, wie mithilfe von Apache Tika und anderen Bibliotheken Metadaten aus den Dateien gewonnen werden können. Diese Informationsblöcke liegen nun vor und müssen zunächst aggregiert werden, da jedes Format mitunter unterschiedliche Metadaten enthält. So werden beispielsweise bei manchen Audioformaten neben dem eigentlichen Interpret auch zusätzliche mitwirkende Künstler abgespeichert, bei anderen Formaten wiederum nicht. Es bietet sich daher an, eine Grundmenge an Attributen zu definieren. Ist diese Grundmenge erst definiert, kann überlegt werden, wie diese Daten zu persistieren sind. Im Folgenden werden die Grundmengen der verschiedenen Rubriken beschrieben. Anschließend werden die Anforderungen an deren Persistierung aufgezeigt. Im letzten Abschnitt dieses Kapitels werden die verschiedenen Möglichkeiten zur Persistierung im Hinblick auf die vorher definierten Anforderungen diskutiert und die Entscheidung für Elasticsearch begründet. Dazu wird ein umfangreicher Vergleich zwischen Elasticsearch und Apache Solr vorgenommen.

6.3.1 Grundmenge der Attribute

Die Grundmenge besteht aus Metainformationen der Formate und aus Informationen, welche aus diesen Metainformationen gewonnen werden können. Jedes Element einer Grundmenge setzt sich dabei aus einem Attributnamen und einem Typ zusammen. Der Typ ist erforderlich, da Metainformationen in unterschiedlichen Dateiformaten in unterschiedlichen Datentypen vorliegen können. So können zum Beispiel die GPS-Informationen in unterschiedlichen Formaten vorliegen, etwa einmal in Grad, Minuten und Sekunden und bei einem anderen Fall in Dezimalgrad. Ein anderes Beispiel sind die Datumsinformationen in den Bilddaten. Diese können in diversen Einheiten angegeben werden. Alleine schon der Unterschied zwischen dem europäischen und dem amerikanischen Format, also die unterschiedliche Reihenfolge von Monat und Tag, zeigt die Problematik auf. Diese Informationen müssen daher in ein einheitliches Format konvertiert werden, ansonsten ist ein Vergleichen der Informationen nicht möglich. Der angegebene Typ ist hierbei das Format, in das alle anderen Formate zunächst konvertiert werden und welches als Grundlage für spätere Persistierung bzw. Verarbeitung der Daten dient. Im Folgenden werden daher für die verschiedenen Dateitypen diese Grundmengen definiert. Weiterhin werden dabei die unterschiedlichen Formen der Metadatenspeicherung vorgestellt.

6.3.1.1 Universell

Zunächst müssen jedoch einige Attribute festgelegt werden, welche alle Dateien gleichermaßen besitzen. Diese Grundattribute sind bei allen Dateien vorhanden und stellen somit das Minimum der

Metainformationen einer Datei dar. Zudem werden zusätzliche Informationen abgelegt, welche im Kontext des OX Drive Relevanz haben.

Tabelle 1 - Universelle Attributmenge einer Datei

Attributname	Attributtyp	Beschreibung
NAME	String	Der Dateiname.
DATE	Long	Zeitpunkt der Erstellung der Datei in Millisekunden seit dem 01.01.1970.
TAGS	String Array	IDs der zugeordneten Tags
DOCUMENT_ID	String	OX Drive ID des Dokuments
DOCUMENT_FOLDER_ID	String	OX Drive ID des Ordners in dem sich das Dokument befindet.

6.3.1.2 Bilder

Die Bilddaten werden vor allem im Exchangeable Image File Format – kurz Exif – gespeichert. Dieses Format besitzt mehrere hundert unterschiedliche Attribute. Von Brennweiten über Bildgrößen bis hin zur Marke der Kamera, der Informationsumfang ist enorm. Von professionellen Fotografen und Fotografieenthusiasten einmal abgesehen, können die meisten Anwender mit vielen der Informationen jedoch wenig bis gar nichts anfangen. Die hier erfolgte Auswahl beschränkt sich daher auf die Informationen, welche dem normalen Endanwender beim Wiederauffinden seiner Bilder behilflich sind. Insbesondere der Ort und der Zeitpunkt der Aufnahme sind hilfreich, um ein bestimmtes Bild wiederzufinden.

Tabelle 2 - Attributmenge für Bilddaten

Attributname	Attributtyp	Beschreibung
DATE_TIME_ORIGINAL	Long	Der Zeitpunkt der Aufnahme in Millisekunden seit dem 01.01.1970.
GPS_LATTITUDE	Double	Die Geographische Breite zum Zeitpunkt der Aufnahme.
GPS_LONGITUDE	Double	Die Geographische Länge zum Zeitpunkt der Aufnahme.
LOCATION_ADDRESS	String	Der Straßenname und die Hausnummer der GPS Position.
LOCATION_POSTAL	String	Der Postcode der GPS Position.
LOCATION_CITY	String	Der Name der Stadt der GPS Position.
LOCATION_REGION	String	Der Name der übergeordneten Verwaltungsinstanz der GPS Position. Bspw. Bundesland o.ä.
LOCATION_COUNTRY	String	Der Name des Landes in dem sich die GPS Position befindet.
IMAGE_WIDTH	Long	Die Breite des Bildes in Pixeln.
IMAGE_HEIGHT	Long	Die Höhe des Bildes in Pixeln.

6.3.1.3 Audiodateien

Die Metadaten von Audiodateien werden unterschiedlich gespeichert. Am meisten verbreitet ist jedoch das ID3-Tag, welches beim MP3 Format eingesetzt wird. Vom ID3-Tag existieren mehrere Versionen welche zum Teil zueinander inkompatibel sind. Neben dem sehr weit verbreiteten MP3-Format findet das FLAC-Format in vielen Bereichen Anklang, da es im Gegensatz zum MP3 Format verlustfrei ist. In diesem wird der Vorbis Comment zur Speicherung von Metadaten eingesetzt. Neben dem FLAC-Format findet er auch bei den Codecs Vorbis und Speex Anwendung, welche allesamt von der Xiph.org Foundation entwickelt werden.

Tabelle 3 - Attributmenge für Audiodaten

Attributname	Attributtyp	Beschreibung
TITLE	String	Der Titel der Audiodatei
ALBUM	String	Das zugehörige Album der Audiodatei.
YEAR	String	Das Jahr der Veröffentlichung.
INTERPRET	String	Der Name des Interpreten.
GENRE	String	Das Genre der Audiodatei.
TITLE_NUMBER	Int	Die Nummer der Audiodatei auf dem Album.

6.3.1.4 Dokumente

Im Bereich von (Text)-Dokumenten sind die Lösungen zur Speicherung von Metainformationen weit gestreut, sodass jedes Dokumentformat die Informationen auf eigene Weise abbildet. Unter den Formaten, welche über Metainformationen verfügen, sind drei Typen weit verbreitet. Einen Großteil bilden die Dokumente, welche von den diversen Microsoft Office Versionen generiert werden und beispielsweise auf „.doc“ und „.docx“ enden. Ebenfalls sehr weit verbreitet ist das Portable Document Format, besser bekannt unter dem Namen PDF, welches von der Firma Adobe Systems entwickelt wurde und in der heutigen Zeit von den meisten Textverarbeitungsprogrammen unterstützt wird. Die letzte Gruppe besteht aus Dokumenten im Open Document Format, wie etwa die Textdokumente mit der Endung „.odt“. Letzterer wurde von der Organization for the Advancement of Structured Information Standards (OASIS) als offener Textstandart entwickelt und ist sogar in der ISO bzw. IEC (International Electrotechnical Commission) Norm 26300:2006 festgehalten. Benutzt wird das Format vor allem aber nicht ausschließlich von den MS Office Alternativen Apache OpenOffice und LibreOffice.

Tabelle 4 - Attributmenge für Dokumentdaten

Attributname	Attributtyp	Beschreibung
AUTHOR	String	Der Name des Autors.
DATE_OF_CREATION	Long	Der Zeitpunkt der ersten Erstellung in Millisekunden seit dem 01.01.1970.
DATE_OF_MODIFICATION	Long	Der Zeitpunkt der letzten Modifikation in Millisekunden seit dem 01.01.1970.
SUBJECT	String	Das Thema bzw. der Gegenstand des Dokuments.
DOC_TITLE	String	Der Titel des Dokuments.
KEYWORDS	String	Eine Liste von Stichworten, welche das Dokument Kategorisieren.
PAGES	Long	Die Anzahl der Seiten des Dokuments.
WORDS	Long	Die Anzahl der Wörter im Dokument.

6.3.2 Persistierungsmöglichkeiten

Sobald man die Informationen aus den Metadaten gewonnen hat, ist es notwendig, diese persistent abzulegen, damit der rechenintensive Vorgang des Extrahierens nicht wiederholt werden muss. Eine Möglichkeit bestünde darin, die bereits verwendete mySQL Datenbank der OX App Suite zu verwenden, indem man diese um die nötigen Einträge erweitert. Dies birgt jedoch einige Nachteile. So ist eine solche Anpassung sehr invasiv und würde daher mit hoher Wahrscheinlichkeit zu vielen nötigen Anpassungen im Code führen. Ebenso wird durch diesen Ansatz der Server als solches stärker belastet, was beispielsweise bei einer zusätzlichen extern eingerichteten Datenbank deutlich weniger auftreten sollte. Zudem sind andere Datenbanken für diesen Anwendungsfall deutlich besser geeignet. Besonders empfehlenswert sind hierbei die dokumentorientierten Datenbanken, da diese für einen solchen Anwendungsfall konzipiert sind. Es empfiehlt sich daher, innerhalb dieser NoSQL Datenbanken nach einer passenden Lösung zu suchen. Zu den bekanntesten Vertretern der dokumentorientierten Datenbanken zählen vor allem MonogDB und CouchDB. Diese dienen jedoch in erster Linie dem Speichern von sehr großen Datenmengen. Besser passen an dieser Stelle Elasticsearch und Apache Solr. Beide basieren auf der Programmbibliothek Apache Lucene und bieten neben einem Document Store eine Volltextsuche auf diese Daten an. Die Daten werden innerhalb von Lucene als JSON Strings abgelegt (siehe Code 1). Lucene baut über diese Daten einen Index auf.

```
{
  "NAME": "img1234.jpg",
  "DATE": 1433256077 000,
  "TAGS": [],
  "DOCUMENT_ID": "123",
  "DATE_TIME_ORIGINAL": 1433256077 000,
  "GPS_LATTITUDE": 51.029752 ,
  "GPS_LONGITUDE": 7.850157,
  "LOCATION_ADDRESS": "Martinstraße 41",
  "LOCATION_POSTAL": "57462 ",
  "LOCATION_CITY": "Olpe",
  "LOCATION_REGION": "Nordrheinwestfalen",
  "LOCATION_COUNTRY": "Deutschland",
  "IMAGE_WIDTH": "800",
  "IMAGE _HEIGHT": "600",
}
```

Code 1 - Beispiel Dokument im JSON Format

Das ermöglicht ein schnelles und flexibles Abfragen der Informationen, womit ideal das Ziel unterstützt wird, Dateien schneller und einfacher auffinden zu können. Diese Stärke zeigt sich vor allem dann, wenn Elasticsearch und Apache Solr in Ergänzung zu einer anderen (bspw. relationalen) Datenbank eingesetzt werden. Dies wiederum entspricht exakt dem vorliegenden Anwendungsfall.

6.3.3 Elasticsearch vs. Apache Solr

Die Entscheidung, ob Apache Solr oder Elasticsearch als Datenbank gewählt wird, hängt von kleinen Faktoren ab. Die Unterschiede zwischen den beiden Suchplattformen sind eher gering. Einer der wesentlichen Unterschiede zwischen Elasticsearch und Apache Solr liegt in der Art und Weise, wie ein Index innerhalb eines Clusters aufgebaut ist. Viele der Ungleichheiten in der Handhabung und innerhalb der Konfiguration der beiden Server beruhen auf diesem Unterschied, daher sollen die Konzepte an dieser Stelle vorgestellt und verglichen werden. Dies ist notwendig, um später die Auswirkungen dieser Unterschiede nachvollziehen zu können.

In einer Cloud besitzt eine Apache Solr Instanz ein oder mehrere Cores. Jeder Core ist hierbei eine Instanz eines Apache Lucene Servers. Ein logischer Index, also eine Informationssammlung, wird Collection genannt und kann sich über die gesamte Cloud erstrecken. Eine solche Collection kann hierzu in mehrere Teile, sogenannte Shards, unterteilt werden. Diese können wiederum eine beliebige Anzahl an Kopien besitzen, welche Replicas genannt werden. Die Shards und Replicas werden von Solr auf die vorhandenen Cores verteilt. Dies sorgt für eine hohe Ausfallsicherheit. Fällt nun ein Shard aus, so wird unter den Replicas dieses Shards ein Leader gewählt, welcher fortan als Shard dient. Werden Dokumente in einer Collection indexiert, so werden diese an den entsprechenden Shard weitergegeben, welcher die Information abspeichert und diese dann an seine Replicas weiterleitet.

In einem Elasticsearch Cluster existiert ein ähnliches Konzept zur Collection, genannt Index. Ähnlich wie auch bei der Collection ist es mithilfe von Elasticsearch möglich, mehrere Indices anzulegen. Jeder Index besteht dabei ebenso aus einem oder mehreren Shards, welche gleichfalls Replica Shards (kurz Replicas) besitzen können. Ein Shard ist hierbei jeweils eine eigene Instanz von Apache Lucene. Diese liegen jedoch nicht in einem Core, sondern sind direkt einer Elasticsearch Instanz zugeordnet. Zusätzlich werden die Dokumente innerhalb eines Indexes in Typen kategorisiert. Jeder Typ verfügt dabei über ein eigenes Mapping, also eine eigene Definition der Daten. Dazu zählen die Datentypen, die Analyseverfahren dieser Daten und andere Einstellungen.

Trotz dieser konzeptionellen Unterschiede verfügen beide über eine ähnlich gute Performance, da sie beide auf Apache Lucene aufbauen. Letztlich entscheiden der Anwendungsfall und die Art der Anfragen darüber, welches der beiden Systeme den Vorzug erhält. Ein Vergleich der beiden Systeme wird zudem dadurch erschwert, dass die Entwicklung bei beiden stetig vorangetrieben wird und somit ältere Informationen an Aussagekraft einbüßen. Es muss daher davon ausgegangen werden, dass sie nur für bestimmte Versionen Gültigkeit besitzen. Insbesondere Vergleiche der Performance sind schwierig zu betrachten, da die gestellten Anfragen nicht dem hier vorliegenden Anwendungsfall entsprechen und somit das Ergebnis gegebenenfalls darüber hinaus verfälschen. Dass die Entwicklung stetig voranschreitet, lässt sich auch an der großen Anzahl an Veröffentlichungen neuer Versionen erkennen, welche in letzter Zeit getätigt wurden. Seit Jahresbeginn wurden fünf neue Versionen von Elasticsearch und zwei neue Versionen von Apache Solr veröffentlicht (Stand 06.05.2015). Die aktuellsten Versionen sind momentan 5.1 für Solr und 1.5.2 für Elasticsearch. Ein Vergleich sollte daher auch auf Basis dieser Versionen stattfinden. Daraus folgt, dass man bezogen auf die Performance nur dann ein Urteil fällen kann, wenn beide Systeme mit einer aktuellen Version in einem praktischen Einsatz getestet werden. Eine Möglichkeit wäre hierfür die Simulation eines Produktivsystems. Da die OX App Suite in unterschiedlichen Umgebungen eingesetzt wird, würde jedoch selbst ein solcher Test nicht zu allgemein gültigen Ergebnissen führen. Daher und vor dem Hintergrund, dass beide Systeme über eine sehr gute Performance verfügen und beliebig skaliert werden können, wurde im Rahmen dieser Arbeit auf eine aufwendige Untersuchung der Performance verzichtet.

Stattdessen sollen die anderen Unterschiede zwischen den beiden Datenbanken betrachtet werden. Es ist dabei zu beachten, dass die Historie der beiden Systeme wesentlich für ihre Entwicklung ist. So ist beispielweise aufgrund des höheren Alters Apache Solr deutlich voluminöser. Die gepackte Größe der aktuellen Version liegt bei 132MB. Das Archiv von Elasticsearch benötigt hingegen nur 30MB Festplattenspeicher und ist damit um mehr als das Vierfache kleiner. Apache Solr bringt dabei eine Vielzahl an Funktionen mit der Grundinstallation bereits mit. Für Elasticsearch können und müssen

hingegen viele Plugins nachträglich installiert werden, wodurch der Größenunterschied nicht mehr allzu gravierend ausfällt. Elasticsearch erlaubt jedoch dabei eine freiere Konfiguration. Ein wesentlicher Unterschied zwischen den beiden Datenbanken ist die anfängliche Ausrichtung. So ist Elasticsearch mit Hinblick auf Cloud Computing entwickelt worden, wohingegen Apache Solr diese Funktionalität erst mit der Version 4 erhalten hat. Apache Solr, oder genauer Solr Cloud, baut dabei auf den Apache Zookeeper auf. Der Apache Zookeeper ist ein System um u.a. verteilte Services zu koordinieren. Es war ursprünglich ein Unterprojekt von Apache Hadoop und wurde 2008 zu einem Top-Level-Projekt der Apache Software Foundation.

Elasticsearch verwendet hingegen einen hauseigenen Mechanismus, Zen-Discovery genannt, um die Verteilung vorzunehmen. Auch hier schreitet die Entwicklung stetig voran. Lange war es der Fall, dass der Mechanismus von Elasticsearch mit dem Split-Brain Problem zu kämpfen hatte. Seit einem Commit vom 01. September 2014 ist dieses Problem jedoch ebenfalls behoben. Insgesamt kann gesagt werden, dass beide Systeme sich vom Funktionsumfang aneinander annähern. Viele Funktionalitäten, welche zunächst nur von einem Produkt unterstützt wurden, werden inzwischen auch von der anderen Datenbank angeboten. Dazu zählen das automatische Balancieren von Shards, die Verfügbarkeit von schemaloser Speicherung, die Unterstützung von geschachtelten Dokumenten, Auto-vervollständigung, Textkorrektur und viele andere mehr. Gravierende Unterschiede findet man somit inzwischen nur noch selten. Eine dieser Funktionalitäten ist die Percolation Funktionalität von Elasticsearch. Diese erlaubt die Speicherung von Anfragen innerhalb der Indices und ermöglicht es somit, eine invertierte Anfrage zu stellen. Dabei wird, statt eine Anfrage zu stellen, einfach ein Dokument an Elasticsearch übergeben. Als Antwort erhält man die Anfragen zurück, welche das Dokument selbst als Antwort zurückgeben würden. Beispielsweise könnte man die Anfrage „Welche Personen haben den Vornamen Max“ in Elasticsarch speichern. Nutzt man nun die Percolation Funktionalität und sendet dabei ein Dokument, welche eine Person mit dem Namen „Max Mustermann“ beschreibt, so erhält man als Antwort die eingangs gespeicherte Anfrage zurück. Hierbei werden alle Anfragen zurückgegeben, welche auf das übergebene Dokument zutreffen. Eine solche Möglichkeit existiert zurzeit für Apache Solr nicht, doch es gibt einen offenen Antrag für ein derartige Funktionalität[33].

Es ist daher aufgrund der stetigen Entwicklung zu erwarten, dass eine solche Funktionalität auch in Solr umgesetzt werden wird. Elasticsearch ist diesbezüglich etwas im Vorteil, da durch den deutlich kompakteren Arbeitsprozess Funktionalitäten schneller entwickelt werden können, als bei Apache Solr. Ein anderer Unterschied findet sich in der dynamischen Anpassung der Shards. Apache Solr erlaubt das nachträgliche Hinzufügen und Splitten von bereits bestehenden Shards. Elasticsearch

[33] https://issues.apache.org/jira/browse/SOLR-4587 (abgerufen am 21.05.2015)

hingegen erlaubt nur das Erhöhen der Replicas. Jedoch ist es in Elasticsearch möglich, neue Indices anzulegen und die Anfragen über mehrere Indices laufen zu lassen. Mit diesem Workaround lässt sich somit die Anzahl an Shards erhöhen, ohne die Dokumente neu indexieren zu müssen. Ein Splitting ist jedoch weiterhin nicht möglich und die Anfragen müssen an diesen Umstand in der Art angepasst werden, als dass sie nun die Anfragen nicht nur auf einem Index ausführen, sondern auf mehreren. Dies kann ggf. zu einem erhöhten Aufwand führen, wenn ein bestehendes Cluster vergrößert werden soll. Außerdem ist es mit Apache Solr möglich, mehr Dateiformate zu verwenden. Wo Elasticsearch auf JSON beschränkt ist, unterstützt Solr von Haus aus eine Menge unterschiedlicher Formate. Insbesondere XML und CSV werden unterstützt und können auch für die Eingabe verwendet werden. Elasticsearch ist für solche Anwendungsfälle auf Plugins angewiesen. Dank der sehr aktiven Community existieren viele dieser Plugins, sodass dieser Nachteil ebenfalls nicht all zu groß ausfällt. Beispielhaft sei hier zur Verarbeitung des XML Formats das Elasticsearch XML Plugin[34] genannt. Der Nachteil von Plugins ist neben dem erhöhten Aufwand der Einrichtung noch die Beschränkung auf alte Versionen, da oftmals für neuere Versionen von Elasticsearch noch keine Commits der Plugins vorliegen. Zudem ist man darauf angewiesen, dass die Entwicklung eines solchen Plugins fortgesetzt wird. Insgesamt sind die funktionalen Unterschiede zwischen Apache Solr und Elasticsearch somit eher gering. Solr besitzt mit dem Zookeeper zwar das ursprünglich robustere System, doch die Zen Discovery hat sich enorm entwickelt. Bei den Unterschieden stechen einzig die Percolation API von Elasticsearch und die Möglichkeit des Shard Splitting bei Solr hervor.

Vergleich Konfiguration und Anfragen

Die Unterschiede der beiden Suchengines finden sich somit meist nur im Detail. Die eher funktionsorientierte Analyse des vorigen Abschnitts soll daher nun um eine praktische Betrachtung der beiden Systeme ergänzt werden. Hierzu wird einerseits eine exemplarische Inbetriebnahme von Elasticsearch und Apache Solr beschrieben. Zum anderen werden die Formulierungen der Anfragen (Query‘s) für den vorliegenden Anwendungsfall beider Systeme vorgestellt und miteinander verglichen.

Als ersten Schritt der Inbetriebnahme beider Datenbanken muss ein Cluster aufgesetzt werden. Als Basis dient hierfür in beiden Fällen ein Rechner mit Ubuntu Betriebssystem in der Version 15.04. Auf diesem soll für dieses Beispiel lokal ein Cluster mit drei Knoten aufgesetzt werden.

[34] https://github.com/jprante/elasticsearch-xml (abgerufen am 21.05.2015)

Die Inbetriebnahme des Elasticsearch Servers gestaltet sich hierbei als sehr einfach. Die einzige Voraussetzung ist eine installierte Java JDK Version, vorzugsweise von Oracle. Da dies jedoch auch für Apache Solr gilt, ist dieser Umstand zu vernachlässigen. Nach dem Download des Pakets[35] muss dieses nun entpackt werden. In der dabei entstandenen Ordnerstruktur finden sich zwei Dokumente zur Konfiguration des Servers, welche im Ordner config liegen. In der elasticsearch.yml Datei werden die Einstellungen für den Elasticsearch Server festgelegt. Jedoch ist keine der Einstellungen für dieses Beispiel zwingend zu konfigurieren, da bereits gute voreingestellte Werte vorliegen. In einem Produktivsystem sind diese Werte jedoch unbedingt zu prüfen. Zu den wichtigen Einstellungen zählen hierbei der Name des Knotens und des Clusters, ob der Knoten ein Master-Knoten ist und ob dieser Daten beinhaltet, die Anzahl der Shards und Replicas pro Index, die Pfade zu den Speicherorten der Daten und einige weitere Einstellungen bezüglich des Arbeitsspeichers und der Netzwerkkonfiguration wie Port, Gateway etc.. Die zweite Datei mit dem Namen logging.yml befindet sich im gleichen Ordner und erlaubt die Konfiguration des Loggins des Servers. Es nutzt dazu das weit verbreitete Log4j. Für eine exemplarische Inbetriebnahme ist eine Konfiguration dieses Systems jedoch ebenfalls nicht erforderlich. Im Anschluss an die Konfiguration ist man in der Lage, den Server mit dem Befehl aus Code 2 zu starten.

```
1  /bin/elasticsearch
```

Code 2 - Befehl zum Starten eines Elasticsearch Servers

```
1  {
2  "status" : 200,
3  "name" : "Node1",
4  "cluster_name" : "myCluster",
5  "version" :
6  {
7        "number" : "1.5.2",
8        "build_hash" : "62ff9868b4c8a0c45860bebb259e21980778ab1c",
9        "build_timestamp" : "2015-04-27T09:21:06Z",
10       "build_snapshot" : false,
11       "lucene_version" : "4.10.4"
12 },
13 "tagline" : "You Know, for Search"
14 }
```

Code 3- Statusmeldung eines einsatzbereiten Elasticsearch Servers

[35] https://www.elastic.co/downloads (abgerufen am 29.05.2015)

Dieser startet den ersten Server und somit den ersten Knoten. Sofern nicht anders konfiguriert, kann man diesen nun unter der Adresse localhost:9200 finden. Wenn der Server einsatzbereit ist, erhält man eine Meldung mit dem Status 200 ähnlich zu der aus Code 3.

In diesem Beispiel wurde dem Knoten der Name „Node1“ innerhalb der elasticsearch.yml zugewiesen und das Cluster „myCluster“ benannt. Zudem zeigt der Status die verwendete Elasticsearch und Lucene Version an (1.5.2 und 4.10.4). Sofern der Server nicht mit der Option -d als Hintergrundprozess gestartet wurde, gibt dieser in der Konsole einige Informationen aus (siehe Code 4).

```
1  [2015-05-27 15:09:33,652][INFO ][node] [Node1] version[1.5.2], pid[10138],
   build[62ff986/2015-04-27T09:21:06Z]
2  [2015-05-27 15:09:33,652][INFO ][node] [Node1] initializing ...
3  [2015-05-27 15:09:33,657][INFO ][plugins] [Node1] loaded [], sites []
4  [2015-05-27 15:09:36,074][INFO ][node] [Node1] initialized
5  [2015-05-27 15:09:36,075][INFO ][node] [Node1] starting ...
6  [2015-05-27 15:09:36,135][INFO ][transport] [Node1] bound_address
   {inet[/0:0:0:0:0:0:0:0:9301]}, publish_address {inet[/192.168.32.144:9301]}
7  [2015-05-27 15:09:36,148][INFO ][discovery] [Node1]
   myCluster/HFrabxnJTZy6ax8yiKdd5w
8  [2015-05-27 15:09:39,198][INFO ][cluster.service] [Node1] detected_master
   [Node1][qXQ4jWhuRzikYo_Ut6AehQ][kevinruthmann-P5QL-
   PRO][inet[/192.168.32.144:9300]], added
   {[Node1][qXQ4jWhuRzikYo_Ut6AehQ][kevinruthmann-P5QL-
   PRO][inet[/192.168.32.144:9300]],}, reason: zen-disco-receive(from master
   [[Node1][qXQ4jWhuRzikYo_Ut6AehQ][kevinruthmann-P5QL-
   PRO][inet[/192.168.32.144:9300]]])
9  [2015-05-27 15:09:39,233][INFO ][http] [Node1] bound_address
   {inet[/0:0:0:0:0:0:0:0:9201]}, publish_address {inet[/192.168.32.144:9201]}
10 [2015-05-27 15:09:39,233][INFO ][node] [Node1] started
```

Code 4 - Ausgabe eines Elasticsearch Servers

Der Server initialisiert sich und beginnt dann andere Knoten mit dem Clusternamen „myCluster“ zu suchen. Da es bisher keine gibt, bestimmt er sich selbst als Master Knoten (Zeile 8). Der Master Knoten ist dafür verantwortlich, den Zustand des Clusters zu überwachen und die Verteilung der Daten (Shards und Replicas) zu verwalten. Zudem wird angezeigt, unter welcher Adresse und unter welchem Port sich der Server registriert. Sollen nun weitere Server dem Cluster hinzugefügt werden, so wird lediglich das Verzeichnis mitsamt den Konfigurationsdateien kopiert. Diese Kopien werden anschließend an den neuen Server angepasst (Bspw. ein neuer Name vergeben). Danach kann der zweite Server auf die gleiche Weise wie der erste gestartet werden. Auch dieser gibt im Terminal ähnliche Informationen wie der erste Server aus (siehe Code 5). Diesmal wird jedoch ein bestehendes Cluster gefunden und Node1 als Master Knoten erkannt (Zeile 8). Zudem wird der Server unter einem

anderen Port registriert. Dies erlaubt es auch, den Status des zweiten Servers zu verifizieren. Die Auswirkungen lassen sich ebenfalls beim Master Knoten Node1 erkennen. Dieser entdeckt den neuen Server und gibt dies in der Konsole aus (siehe Code 6).

```
1  [2015-05-27 15:16:54,319][INFO ][node] [Node2] version[1.5.2], pid[10314],
   build[62ff986/2015-04-27T09:21:06Z]
2  [2015-05-27 15:16:54,319][INFO ][node] [Node2] initializing ...
3  [2015-05-27 15:16:54,324][INFO ][plugins] [Node2] loaded [], sites []
4  [2015-05-27 15:16:56,748][INFO ][node] [Node2] initialized
5  [2015-05-27 15:16:56,749][INFO ][node] [Node2] starting ...
6  [2015-05-27 15:16:56,809][INFO ][transport] [Node2] bound_address
   {inet[/0:0:0:0:0:0:0:0:9302]}, publish_address {inet[/192.168.32.144:9302]}
7  [2015-05-27 15:16:56,823][INFO ][discovery] [Node2]
   myCluster/OJs15kc2QNaetGa_XFbpUw
8  [2015-05-27 15:16:59,869][INFO ][cluster.service] [Node2] detected_master
   [Node1][qXQ4jWhuRzikYo_Ut6AehQ][kevinruthmann-P5QL-
   PRO][inet[/192.168.32.144:9300]], added
   {[Node1][qXQ4jWhuRzikYo_Ut6AehQ][kevinruthmann-P5QL-
   PRO][inet[/192.168.32.144:9300]],[Node1][HFrabxnJTZy6ax8yiKdd5w][kevinruthmann
   -P5QL-PRO][inet[/192.168.32.144:9301]],}, reason: zen-disco-receive(from master
   [[Node1][qXQ4jWhuRzikYo_Ut6AehQ][kevinruthmann-P5QL-
   PRO][inet[/192.168.32.144:9300]]])
9  [2015-05-27 15:16:59,903][INFO ][http] [Node2] bound_address
   {inet[/0:0:0:0:0:0:0:0:9202]}, publish_address {inet[/192.168.32.144:9202]}
10 [2015-05-27 15:16:59,903][INFO ][node] [Node2] started
```

Code 5 - Terminalausgabe des zweiten Elasticsearch Servers

```
1  [2015-05-27 15:16:59,862][INFO ][cluster.service] [Node1] added
   {[Node2][OJs15kc2QNaetGa_XFbpUw][kevinruthmann-P5QL-
   PRO][inet[/192.168.32.144:9302]],}, reason: zen-disco-receive(from master
   [[Node1][qXQ4jWhuRzikYo_Ut6AehQ][kevinruthmann-P5QL-
   PRO][inet[/192.168.32.144:9300]]])
```

Code 6 - Bestätigung des ersten Servers, dass dieser den zweiten Server erkannt hat

Sofern die Server sich innerhalb eines Netzes finden können, bilden diese gemeinsam direkt einen Cluster. Der Vorgang ist bei weiteren Servern identisch.

Die Einrichtung einer Apache Solr Cloud ist hingegen etwas aufwendiger, da es ein funktionsfähiges Apache Zookeeper Cluster benötigt. Es existiert zwar ein Script, mit dem ein einfaches lokales Setup mit integriertem Zookeeper aufgesetzt werden kann, doch ist dieses nicht für Produktions-

umgebungen konzipiert. Dies liegt daran, dass bei einem integrierten Zookeeper dieser in derselben JVM läuft wie der Solr Server. Dies kann zu führen, dass diese sich gegenseitig negativ beeinflussen.

Um einen Zookeeper Cluster aufzusetzen, benötigt man zunächst ein Release des Apache Zookeeper, welcher bspw. unter http://apache.mirrors.tds.net/zookeeper/ zu finden ist. Nach dem Entpacken des Archivs muss eine Konfigurationsdatei mit dem Namen zoo.cfg unter dem Ordner conf angelegt werden. An dieser Stelle müssen einige wichtige Einstellungen vorgenommen werden. Dazu gehören unter anderem, dass der Pfad für die Daten angegeben, der Port für die Clients festgelegt und die Zookeeper Instanzen eingetragen werden. Code 7 zeigt den Aufbau einer Beispielkonfiguration für ein Setup mit drei lokalen Zookeeper-Instanzen.

```
1  tickTime=2000
2  initLimit=10
3  syncLimit=5
4  dataDir=/home/kevinruthmann/storage/Zookeeper/data
5  clientPort=2181
6  server.1=localhost:2888:3888
7  server.2=localhost:2889:3889
8  server.3=localhost:2890:3890
```

Code 7 - zoo.cfg für einen lokalen Cluster mit drei Instanzen

Für ein lokales Setup ist es erforderlich, die Ports der Instanzen zu variieren. Dies gilt sowohl für den clientPort (Zeile 5) als auch für die Ports der Server untereinander (Zeile 6-8). Für jeden Server muss nun der Verzeichnisbaum kopiert und die zoo.cfg angepasst werden. Zudem muss bei jedem Server eine Datei im dataDir Ordner mit dem Namen myid angelegt werden, welche lediglich die Servernummer des Servers enthält (jeweils 1, 2 oder 3). Ist die Vorbereitung abgeschlossen, so kann jeder Server mit dem Befehl aus Code 8 gestartet werden. Die Server führen dann automatisch eine Wahl durch und bilden anschließend ein Cluster.

```
1  bin/zkServer.sh start-foreground
```

Code 8 - Befehl zum Starten eines Zookeeper Servers

```
1  bin/solr start -f -c -h solr1 -p 8983 -s data1 -z
   localhost:2181,localhost:2182,localhost:2183
```

Code 9 - Befehl zum Starten eines Solr-Servers

Wenn der Zookeeper Cluster einsatzbereit ist, können die Solr-Instanzen gestartet werden. Hierzu wird zunächst das Solr Archiv heruntergeladen und entpackt. Anschließend wird ein Ordner für die Daten angelegt und eine solr.xml Datei darin platziert. Für diesen Fall reicht es aus, die solr.xml Datei aus dem example/cloud Ordner zu kopieren. Anschließend kann die Instanz mit dem Befehl aus Code 9 gestartet werden.

Dieser verfügt über eine Reihe von Einstellungsmöglichkeiten, wie etwa der Servername, der verwendete Port und die Liste der Zookeeperinstanzen. Der Port und Servername sind dabei je nach Instanz anzupassen. Zudem ist darauf zu achten, dass die Adressen und Ports der Zookeeper-Server exakt die gleichen sind, wie sie in der zoo.cfg eingetragen wurden. Als Port ist hierbei der clientPort zu verwenden. Die Option -f startet den Server im Vordergrund und kann daher innerhalb eines Produktivsystems weggelassen werden. Für dieses Beispiel ist dies jedoch nützlich, da so direkt angezeigt wird, ob sich die Solr-Instanzen erfolgreich mit dem Zookeeper verbunden haben. Dies kann man anhand der Zeile aus Code 10 erkennen. Nun muss lediglich noch eine Collection mit dem Befehl aus Code 11 angelegt werden.

```
1  2060 [main] INFO  org.apache.solr.common.cloud.ConnectionManager  [   ] – Client is
   connected to ZooKeeper
```

Code 10 - Bestätigung, dass sich der Server mit dem Zookeeper verbunden hat

```
1  bin/solr create -c myCollection -shards 3
```

Code 11 - Befehl zum Erstellen einer Collection

Dies erzeugt eine Collection mit dem Namen myCollection, welche auf alle drei solr Instanzen aufgeteilt wird. Ggf. ist es nötig, die Adressen innerhalb der etc/hosts Datei mit dem Servernamen einzutragen. Der Status der Installation lässt sich nun unter localhost:8983/solr/#/~cloud prüfen (siehe Abbildung 6).

Abbildung 6 - Solr Admin Panel

Anhand der Grafik lässt sich erkennen, dass eine Collection angelegt und dass ihre Shards auf die drei verfügbaren Server verteilt wurden. Bei der Verwendung von Replicas würden diese ebenfalls an dieser Stelle angezeigt werden.

Insgesamt ist die Erstellung eines Elasticsearch Servers somit schneller durchzuführen als die Inbetriebnahme einer Solr Cloud. Zudem ist die Konfiguration mithilfe der Konfigurationsdateien bei Elasticsearch übersichtlicher und bietet dank guter voreingestellter Werte einen unkomplizierten Start. Steht im Produktionsumfeld jedoch bereits ein aktiver Zookeeper zur Verfügung, so ist der Vorteil von Elasticsearch nur noch minimal.

Neben den Abweichungen bei der Einrichtung der Cloud existieren noch weitere Unterschiede bei der Indexierung und der Abfrage von Dokumenten. Für den vorliegenden Anwendungsfall ist vor allem der Java-Client interessant, welcher von beiden Systemen selbst mitgebracht wird. Die damit erstellten Anfragen sollen nun im Folgenden miteinander verglichen werden. Als Beispiele dienen hierfür die Anfragen aus dem vorliegenden Kontext. Daher wird auch das zuvor erarbeitete Datenschema verwendet.

Bevor eine Anfrage an den jeweiligen Cluster gestellt wird, muss eine Verbindung zu diesem aufgebaut bzw. ein Verbindungsobjekt erstellt werden. Apache Solr bietet für einen Server im Cloud Modus den *CloudSolrClient* an. Dieser wird lediglich mit den Adressen der Instanzen initialisiert und ist ab diesem Zeitpunkt einsatzbereit (siehe Code 12). Zudem empfiehlt es sich, eine DefaultCollection einzustellen. In dem vorliegenden Beispiel ist dies „myCollection“ (Zeile 3).

```
1  String zkHostString =
   "localhost:2181,localhost:2182,localhost:2183";
2  CloudSolrClient solr = new CloudSolrClient(zkHostString);
3  solr.setDefaultCollection(COLLECTION);
```

Code 12 - Initialisierung eines CloudSolrClient Objekts

Elasticsearch besitzt mit dem *TransportClient* ein ähnliches Konstrukt. Diesem wird jedoch zusätzlich ein *Settings* Objekt mitgegeben, indem sich der Clustername befindet. Steht zudem die Option client.transport.sniff auf true, so kann der Client mithilfe eines funktionierenden Knotens alle andere Knoten finden und dem Client hinzufügen. Ebenso wie bei Solr werden die Adressen des Servers mitgegeben. Elasticsearch setzt hierbei das Fluent Interface Entwurfsmuster ein, bei dem die set oder add Methoden eines Objekts immer das Objekt selbst zurückgeben. Somit ergeben sich Satz-artige Gebilde, welche den Code übersichtlich und kompakt halten (siehe Code 13).

```
1  Settings settings = ImmutableSettings.settingsBuilder()
   .put("cluster.name", clusterName)
   .put("client.transport.sniff", true)
   .build();
2  TransportClient client =    new TransportClient(settings)
   .addTransportAddress(new InetSocketTransportAddress(host,
   port));
```

Code 13 - Erstellung eines TransportClient Objekts

Mit diesen Verbindungen ist es nun möglich, Dokumente in den Servern abzulegen. Bei Elasticsearch muss hierzu das Dokument im JSON Format vorliegen. Eine Variante, um ein solche Objekt zu erzeugen, ist die Nutzung eines *XContentBuilder*. Mithilfe der Befehle startObject() und stopObject() können neue Unterobjekte erstellt und geschlossen werden und mit dem Befehl field(key,value) lassen sich Schlüssel-Wert-Paare ablegen. Abschließend kann der *XContentBuilder* mit der Methode string() in einen JSON-String konvertiert werden (siehe Code 14).

```
1  XContentBuilder xb;
2  String jsonString;
3  xb = XContentFactory.jsonBuilder().startObject();
4  xb.field("id", "1");
5  xb.field("name", "image1");
6  xb.field("tags", new String[]{});
7  xb.field("DATE_TIME_OIGINAL", System.currentTimeMillis());
8  xb.field("GPS_LATTITUDE", 51.029591);
9  xb.field("GPS_LATTITUDE", 7.850635);
10 xb.field("LOCATION_ADDRESS", "Martinstraße 41");
11 xb.field("LOCATION_POSTAL", "57462");
12 xb.field("LOCATION_CITY", "Olpe");
13 xb.field("LOCATION_REGION", "Nordrheinwestfalen");
14 xb.field("LOCATION_COUNTRY", "Deutschland");
15 xb.field("IMAGE_WIDTH", "800");
16 xb.field("IMAGE_HEIGHT", "600");
17 xb.endObject();
18 jsonString = xb.string();
```

Code 14 - Erzeugung eines JSON String mithilfe von XContentBuilder

Dieser wird als Source bei der Indexierung übergeben. Ferner muss noch die IndexID und der Typ angegeben werden (siehe Code 15).

```
1  IndexResponse response =
   client.prepareIndex(index,type).setSource(jsonString)
   .execute().actionGet();
```

Code 15 - Befehl zur Indexierung bei Elasticsearch

Dieser gibt ein IndexResponse-Objekt zurück, welches Informationen über den Erfolg oder Misserfolg der Operation beinhaltet. Dieser beinhaltet auch die ID des Dokuments, welches beispielsweise für eine Abfrage des Dokuments erforderlich ist (siehe unten).

Bei Apache Solr wird stattdessen ein binary Objekt verwendet. Dieses wird mithilfe von *SolrInputDocument* gebildet. Auch hier lassen sich Schlüssel-Wert-Paare hinzufügen, jedoch wird in diesem Fall auf eine Strukturierung verzichtet. Zudem lässt sich das so entstandene Objekt direkt verwenden. Ähnlich, wie auch bei Elasticsearch, wird ein Objekt mit der Antwort zurückgegeben. Es ist bei Apache Solr jedoch zusätzlich notwendig, ein commit auf die Daten auszuführen, damit diese final im Server abgelegt werden (siehe Code 16).

```
1  SolrInputDocument document = new SolrInputDocument();
2  document.addField("id", "1");
3  document.addField("name", "image1");
4  document.addField("tags", new String[]{});
5  document.addField("DATE_TIME_OIGINAL",
                     System.currentTimeMillis());
6  document.addField("GPS_LATTITUDE", 51.029591);
7  document.addField("GPS_LONGITUDE", 7.850635);
8  document.addField("LOCATION_ADDRESS", "Martinstraße 41");
9  document.addField("LOCATION_POSTAL", "57462");
10 document.addField("LOCATION_CITY", "Olpe");
11 document.addField("LOCATION_REGION",
                     "Nordrheinwestfalen");
12 document.addField("LOCATION_COUNTRY", "Deutschland");
13 document.addField("IMAGE_WIDTH", "800");
14 document.addField("IMAGE_HEIGHT", "600");
15 UpdateResponse response = solr.add(document);
16 solr.commit();
```

Code 16 - Indexierung eines Dokuments in Apache Solr

Um ein Dokument zu einem späteren Zeitpunkt wieder abzufragen, wird bei Elasticsearch ein GetRequest verwendet. Dieser beinhaltet neben der IndexID und dem Typ des Dokuments zusätzlich die ID des Dokuments selbst. Rückgabewert dieser Funktion ist ein GetResponse, welcher bei Erfolg das Dokument enthält (siehe Code 17).

```
1  GetResponse getResponse = client.get(new GetRequest(index,
   type, id)).actionGet()
```

Code 17 - Abfrage eines Dokuments in Elasticsearch

Ähnlich funktioniert dies auch bei Apache Solr, wobei dort eine normale Query eingesetzt wird, welche einen besonderen RequestHandler benutzt. Die zurückgegebene QueryResponse beinhaltet bei Erfolg, ähnlich wie auch bei Elasticsearch, das Dokument (siehe Code 18)

```
1  SolrQuery q = new SolrQuery();
2  q.setRequestHandler("/get");
3  q.set("id", "1");
4  QueryResponse response = solr.query(q);
```

Code 18 - Abfrage eines Dokuments in Apache Solr

Eine wesentliche Anfrage ist die nach Dokumenten, welche einem bestimmten Tag zugeordnet sind. Im Kontext von Elasticsearch wird hierzu eine MatchQuery verwendet (siehe Code 19). Da Dokumente mit verschiedenen Typen abgefragt werden sollen, werden die Typen für Bilder, Audiodaten und Dokumente eingetragen (Zeile 3). Dies sorgt dafür, dass die Suche über den festgelegten Typen durchgeführt wird. Alternativ könnte auf die Angabe verzichtet werden und stattdessen null angegeben werden. Dies würde dafür sorgen, dass der Index vollständig, das heißt über alle Typen hinweg, durchsucht würde. Da die Suche ggf. über viele Einträge erfolgen kann werden diese mithilfe der Scrolls API abgefragt. Bei dieser API werden die Ergebnisse einer Anfrage Seitenweise abgefragt, anstatt alle Ergebnisse direkt zu erhalten. Hierzu wird der SearchType auf Scan festgelegt (Zeile 4), dies sorgt zudem dafür, dass die Ergebnisse nicht mehr sortiert werden. Des Weiteren wird festgelegt, wie lange die Ergebnisse der Anfrage vorgehalten werden. Wird innerhalb dieses Zeitraums eine nächste Seite aufgerufen, wird der Countdown zurückgesetzt. Geschieht dies nicht, sind die Ergebnisse nicht mehr zugänglich und die Anfrage muss erneut gestellt werden. Um eine weitere Seite anzufordern, ist lediglich die ScrollID notwendig (siehe Code 20).

```
1  QueryBuilder builder = QueryBuilders.matchQuery("TAGS",
   tagID);
2  SearchRequestBuilder result =
   ESIndex.getInstance().client.prepareSearch(index);
3  result.setTypes(image_type, document_type, audio_type);
4  result.setSearchType(SearchType.SCAN)
              .setScroll(new TimeValue(scrollTime))
              .setQuery(builder);
5  SearchResponse response = result.execute().actionGet();
```

Code 19 - Matchquery unter Verwendung der Scrolls API bei Elasticsearch

```
1  SearchResponse = client.prepareSearchScroll(scrollID)
   .setScroll(new TimeValue(ESIndex.getScroll_time()))
   .execute()
   .actionGet();
```

Code 20 - Benutzung der Scrolls API bei Elasticsearch

Bei Apache Solr existiert mit Cursor ein ähnliches Konzept (siehe Code 21). Im Unterschied zu Elasticsearch ist hierbei jedoch zwingend eine Sortierung erforderlich (Zeile 2). Wenn der CURSOR_MARK_PARAM Parameter gesetzt ist (Zeile 7), so enthält die Response einer Query die nächste Position den Cursors (Zeile 10). Solange diese nicht mit der alten Position übereinstimmt existieren zusätzliche Daten, welche durch das Setzen des neuen Cursors innerhalb der Query abgefragt werden können (Zeile 7+8). Wie man hier deutlich erkennen kann setzt Solr für diesen Bereich ebenfalls auf die Fluent API.

```
1  SolrQuery query = new SolrQuery();
2  query.setQuery("TAGS:"+tagID).setStart(0)
   .setRows(numOfRows).setSort(SortClause.asc("id"));
3  String cursorMark = CursorMarkParams.CURSOR_MARK_START;
4  boolean running=true;
5  while(running)
6  {
7  query.set(CursorMarkParams.CURSOR_MARK_PARAM, cursorMark);
8  QueryResponse response = solr.query(query);
9  //process results
10 String nextCursorMark = response.getNextCursorMark();
11 if(cursorMark.equals(nextCursorMark))
12       running=false;
13 cursorMark = nextCursorMark;
14 }
15 solr.close();
```

Code 21 - Verwendung von Cursor bei Apache Solr

Letztlich fallen die Unterschiede zwischen den Anfragen beider Systeme für den vorliegenden Anwendungsfall eher gering aus. Beide arbeiten mit sehr ähnlichen Konstrukten, wie etwa Response Objekten. Insgesamt ist die API von Elasticsearch etwas übersichtlicher. Insbesondere das Abfragen eines einzelnen Dokuments ist aufgrund des GetRequest's etwas intuitiver. Zudem erlaubt Elasticsearch bei seiner Scrolls API das Konfigurieren der Ablaufzeit. Eine solche Möglichkeit existiert bei Solr indes nicht.

7 Datensicherheit und Privatsphäre

Die Themen Datensicherheit und Privatsphäre stehen in den letzten Jahren immer mehr im Fokus des öffentlichen Interesses. Dies gilt auch insbesondere für den Bereich Cloud-Speicher. So berichtete erst kürzlich Spiegel Online im Januar 2015[36], dass Microsoft die Dateien seiner OneDrive Kunden durchsucht, nach kinderpornografischen Inhalten durchstöbert und sogar in bestimmten Fällen eigenständig eine Strafverfolgung initiiert. Dass die Dienste für Cloud-Speicher bereits einen schlechten Ruf besitzen, zeigt sich schon am ersten Satz des Artikels: „Ein aktueller Fall zeigt einmal mehr, dass Cloud-Speicher nicht notwendigerweise als privat zu betrachten sind“. Im Rahmen einer Verfolgung von Kinderschändern mögen einige eine solche Maßnahme sogar gutheißen, doch sollte auch jeder unschuldige Bürger ein flaues Gefühl dabei haben, dass seine Dateien vom Dienstleister systematisch durchsucht werden. Insbesondere auch dann, wenn der Anwender sensible Daten in seiner Cloud lagert. Es ist somit entscheidend, wie ein Hoster mit diesen Daten umgeht. Einerseits wird zwar zumeist versucht, die Daten vor Fremdpersonen zu schützen, indem diese verschlüsselt übertragen und auch gespeichert werden. Andererseits besitzen die Hoster jedoch in den meisten Fällen immer noch selbst Klartextzugriff auf die Dateien. Dies kann, wie im Fall von Microsoft, dazu führen, dass die Daten durchsucht werden. Es stellt sich somit die Frage, ob der Endanwender dem Dienstleister vertraut oder vielmehr vertrauen kann.

Dies trifft insbesondere auch auf amerikanische Unternehmen zu, welche beispielsweise durch den USA PATRIOT Act dazu verpflichtet sind, Daten an die Regierungsbehörden auszuhändigen. Daher wird vielerorts empfohlen, die eigenen Daten vor dem Upload eigenständig zu verschlüsseln, sofern es sich um sensible Daten handelt. Hierzu stehen bereits einige Tools bereit, wie etwa die Software Boxcryptor[37], welche speziell für diesen Fall entwickelt wurde und die Daten vor dem Upload in eine beliebige Cloud verschlüsselt. Dies ist jedoch nur ein Beispiel, denn es existieren einige Softwarelösungen, die eine sichere Verschlüsselung der Daten versprechen. Doch auch hier ist der Endanwender dazu gezwungen, dieser Software zu vertrauen. Ein solcher Ansatz wird auch von einigen Anbietern von Cloud-Speicher wie etwa dem Dienst Wuala[38] von der Firma LaCie verfolgt. Dabei werden die Daten clientseitig mit einem nur dem Nutzer bekannten Schlüssel verschlüsselt, bevor sie in die Cloud hoch geladen werden. Es existieren somit Möglichkeiten, die Daten sicher zu speichern, jedoch besitzt diese Erhöhung der Sicherheit eine Kehrseite. Der Vorteil der Verschlüsselung ist gleichzeitig auch ihr Nachteil, denn die verschlüsselten Daten können nicht im Kontext der Smartfeatures aufbereitet werden. Es bestünde lediglich die Möglichkeit, die Metadaten

[36] http://www.spiegel.de/netzwelt/web/kinderpornografie-hausdurchsuchung-nach-hinweis-durch-microsoft-a-1012713.html (abgerufen am 10.03.15)

[37] https://www.boxcryptor.com/de (abgerufen am 10.03.15)

[38] https://www.wuala.com/de/ (abgerufen am 10.03.15)

clientseitig zu gewinnen und diese dann entweder dort zu speichern oder unverschlüsselt in der Cloud abzulegen. Letzteres würde zwar die eigentlichen Daten vor Fremdzugriff schützen, es wäre aber so dennoch möglich, einige Informationen über die abgelegten Daten in Erfahrung zu bringen. Migletz zeigt in (Migletz, 2008) deutlich auf, welcher Informationsgehalt aus Sicht der Strafverfolgung in den Metadaten zu finden ist. Dies gilt natürlich nicht nur für Daten von kriminellen, sondern auch für Daten von unbescholtenen Bürgern. So lässt sich bspw. unter der Verwendung der GPS-Daten in Bildern bestimmen, wo sich eine Person zu gewissen Zeitpunkten aufgehalten hat. Eine unverschlüsselte Speicherung der Metadaten innerhalb der Cloud würde demnach dem eigentlichen Sinn der Verschlüsselung widersprechen.

Die Smartfeatures lassen sich daher nur bedingt umsetzten. So könnten diese Features rein clientseitig ausgeführt werden, was jedoch mit einigen Hindernissen verbunden ist. Zum einen müssten die Informationen clientseitig abgelegt und vor allem aktuell gehalten werden. Das bedeutet, dass es bei der Nutzung von mehreren Geräten zu häufigen Aktualisierungen kommt, da jeder Client, sobald eine Änderung der Daten eintritt, diese direkt analysieren müsste. Zudem ließen sich so nur automatisierte Features abbilden. Manuelle Eingriffe des Anwenders müssten weiterhin in der Cloud abgelegt werden, wobei diese jedoch verschlüsselt gespeichert werden könnten. Ausgenommen sind hierbei jedoch temporär genutzte Clients. Etwa die Benutzung des Webzugriffs mittels eines fremden Computers. Da hierbei immer alle Dateien für den Client neuartig wären, müsste diese vollständig analysiert werden. Dies hätte jedoch einen erheblichen Aufwand zur Folge, da es sich um eine Großzahl von Dateien handeln könnte. Insofern scheint eine clientseitige Verschlüsselung der Daten dem Konzept der Smartfeatures zu widersprechen.

Der Endanwender muss sich demnach entscheiden, ob er seine Daten sicher vor Zugriffen schützen oder ob er alternativ die beschriebenen Smartfeatures nutzen will. Letzteres setzt dabei natürlich das Vertrauen voraus, dass der Anbieter die Daten nur im Rahmen der Aufbereitung verwendet. Eine wirkliche Alternative besteht hierbei lediglich in der sogenannten OwnCloud[39]. Dabei handelt es sich um eine Cloudsoftware, die jeder Anwender theoretisch selbst betreiben kann. Der Anwender verwaltet seine Daten somit selbstständig und muss diese nicht mehr in die Hand eines Unternehmens geben. So kann er sich sicher sein, dass seine Daten nicht anderweitig genutzt werden. Der Nachteil dieser Variante ist dabei der erhöhte Aufwand für den Anwender. Er muss einen Server mieten oder bereitstellen und ist selbst verantwortlich für dessen Wartung. Er muss beispielsweise selbst dafür sorgen, dass der Speicherplatz ausreicht oder dass die Software die nötigen Sicherheitsupdates erhält. Zudem besitzt der Anwender kein freies Kontingent, das heißt, dass bei Inbetriebnahme für den Anwender bereits Kosten entstehen. Entweder durch die Kosten der

[39] https://owncloud.org/ (abgerufen am 10.03.15)

einmaligen Anschaffung eines eigenen Servers oder durch die fortlaufenden Mietkosten eines solchen. Die meisten Anbieter für Cloud-Speicher bieten im Gegensatz dazu einige Gigabyte an kostenlosem Speicher an. Will der Anwender jedoch hingegen große Datenmengen speichern, so kann die Betreibung einer eigenen Cloud natürlich günstiger sein. Ein weiterer Makel ist zudem die hohe Ausfallwahrscheinlichkeit beim Betrieb eines einzelnen Servers. Überdies besitzt der durchschnittliche Nutzer von Cloud-Speicher nicht die nötige Expertise, um einen solchen Server zu betreiben. Die OwnCloud bleibt damit für einen durchschnittlichen Anwender eher ein theoretisches Konzept.

Um die Bedenken bezüglich der Privatsphäre einordnen zu können, ist es erforderlich, die aktuelle Meinung der Anwender näher zu beleuchten. Diese wissen oftmals gar nicht, ob und wie ihre Daten genutzt werden. Die Entwicklung zeigt jedoch, dass die Menschen inzwischen ein Bewusstsein dafür entwickeln, welche Informationen sie über sich selbst im Internet preisgeben. Dies geht auch aus dem Consumer Openness Index 2015 (Open-Xchange, 2015) hervor, einer Umfrage, welche die Open-Xchange AG mit 3000 Nutzern aus Deutschland, Großbritannien und den Vereinigten Staaten durchgeführt hat. So stimmten 80 % der Befragten zu, dass es in der Verantwortung der Nutzer liegt, die persönlichen Informationen zu kontrollieren, die sie online einstellen. Über 60 % sprachen sich überdies dafür aus, dass die Regierung die Privatsphäre der Bürger schützen soll. Es ist somit erforderlich, sensibel mit den Daten der Anwender umzugehen. Es sollte daher reiflich überlegt werden, ob man ein System einsetzt, welches auf den Metadaten der Anwender, also ihren privaten Informationen, basiert. Auch wenn sorgfältig mit den Daten aus Sicht des Betreibers umgegangen wird, so kann dies dennoch zu einer negativen Außenwirkung führen. Ein Risiko, welches auch als ein Faktor dafür angesehen werden kann, dass andere Anbieter für Cloud-Speicher bisher kaum Smartfeatures in ihren Applikationen anbieten. Dies gilt insbesondere, da die Konkurrenz in diesem Segment sehr hoch ist und ein etwaiger Imageverlust verheerende Auswirkungen auf ein Unternehmen haben könnte. Gleichzeitig stellt es jedoch auch eine Möglichkeit dar, sich von der Konkurrenz abzuheben und sich damit einen Vorteil zu verschaffen.

Das Feature Caroussel von Dropbox scheint hierbei einen Kompromiss zu fahren, um sich genau einen solchen Vorteil zu verschaffen. Es wird nur auf ein Metadatum – dem Aufnahmedatum – zugegriffen und auf dessen Basis einige wenige der Smartfeatures angeboten. Ein ähnlicher Kompromiss ließe sich im Produkt OX Guard finden. Diese Erweiterung der OX App Suite erlaubt die Verschlüsselung von Dateien. Dabei ist es auch möglich, dass der Anwender im Einzelfall entscheidet, ob er die Verschlüsselung nutzen will. Somit kann er die Daten mit sensiblen Inhalten schützen und gleichzeitig die Smartfeatures für all anderen Dateien nutzen. Der Anwender gibt hierdurch zwar immer noch Informationen über sich preis, doch er kann selbst entscheiden, welche er preisgeben

möchte. Wie (Open-Xchange, 2015) gezeigt hat sieht sich der Anwender selbst in der Verantwortung und hat mit dieser Erweiterung wieder selbst die Entscheidungsmöglichkeit.

Neben der Problematik, einem Anbieter für Cloud-Speicher seine Daten anzuvertrauen, besteht zusätzlich noch das Problem, die erhobenen Metadaten vor Zugriff durch Dritte zu schützen. Das heißt in diesem Fall, dass der Zugriff auf die Elasticsearch Server in der Art beschränkt werden muss, dass nur Zugriffsberechtigte an die Daten heran kommen. Elasticsearch bietet hierfür ein Sicherheitssystem mit dem Namen Shield an. Der an eine fiktive Organisation aus dem Marvel Universum angelehnte Name verrät bereits die wichtigste Funktion des kostenpflichtigen Plugins: es soll die Daten beschützen. Hauptbestandteil von Shield ist die Verschlüsselung der Kommunikation zwischen den Elasticsearch-Knoten mithilfe von SSL/TLS. Zudem beinhaltet es eine rollenbasierte Zugriffskontrolle, mit dem der Zugriff auf bestimmte Daten ganz oder nur teilweise eingeschränkt werden kann. Neben der Unterstützung von LDAP basierten Authentifikationssystemen stellt Shield die Möglichkeit bereit, IP-Filter einzurichten und somit den Zugriff auf die festgelegten Adressen zu beschränken. Neben dem hauseigenen Shield stehen weitere Plugins zur Sicherung eines Elasticsearch-Servers bereit, welche über einen ähnlichen Funktionsumfang verfügen. Zu diesen gehören unter anderem das Elasticsearch Auth Plugin[40], der Elastic Defender[41] oder auch das Jetty Plugin[42]. Im Gegensatz zu Shield sind diese jedoch oftmals kostenfrei nutzbar. Zudem steht immer die Möglichkeit bereit, den Elasticsearch Server lediglich in einem privaten Netz zu betreiben, welches keinen oder nur stark beschränkten Zugriff von außen erlaubt.

[40] https://github.com/codelibs/elasticsearch-auth (abgerufen am 20.06.2015)
[41] https://github.com/salyh/elastic-defender (abgerufen am 20.03.2015)
[42] https://github.com/sonian/elasticsearch-jetty (abgerufen am 20.06.2015)

8 Smartfeature Integration

In Kapitel 6 wurde beschrieben, wie ein System zur automatisierten Extraktion von Metadaten aufgebaut sein muss. Dabei wurde insbesondere auf die zu verwendenden Technologien eingegangen. Dieses Kapitel gibt nun einen Überblick über die gewählte Architektur des entstandenen Systems und beschreibt dessen Integration in die OX App Suite. Es wird dabei auf wesentliche Probleme und dessen Lösungen eingegangen. Ausgangspunkt für die Entwicklung des Systems war die zu Beginn getroffene Annahme, dass die Smartfeatures als eine Erweiterung des OX Drives angesehen wurden. Insofern sollten die Anpassungen an das bereits bestehende System minimal invasiv sein. Daher wurde ein Ansatzpunkt auf einer höheren Ebene innerhalb des Systems gewählt. Zu diesem Zweck wurde der Wrapper *SmartFolderFileAccess* geschrieben, welcher eine Alternative zur Klasse *FileStorageFileAccess* darstellt. Hierzu wird das sogenannte Delegation Entwurfsmuster eingesetzt. Dabei führt eine Instanz der Klasse die Aufgaben nicht selbst durch, sondern leitet diese an ein anderes Objekt weiter. In diesem Fall an eine Instanz der Klasse *FileStorageFileAccess*. Hierbei ist es irrelevant, wie die dahinter liegende Implementation aussieht. Die Befehle werden einfach an das Delegations-Objekt weitergereicht und das Ergebnis anschließend zurückgegeben. Über diese Basisschnittstelle hinaus implementiert die Klasse *SmartFolderFileAccess* weitere Schnittstellen, wie etwa *FileStorageVersionedFileAccess*. Wird eine Methode dieser erweiternden Schnittstellen aufgerufen, wird zunächst geprüft, ob das Delegations-Objekt diesem genügt. Wenn nicht, wird eine Exception geworfen. Zudem wird durch die *FileStorageFileAccessDelegation* Schnittstelle dafür gesorgt, dass bei einer Überprüfung der Klasse

```
1  public class SmartFolderFileAccess implements
2        FileStorageFileAccess,
3        FileStorageFileAccessDelegation,
4        FileStorageVersionedFileAccess,
5        FileStorageETagProvider,
6        FileStorageIgnorableVersionFileAccess,
7        FileStoragePersistentIDs,
8        FileStorageRandomFileAccess,
9        FileStorageAdvancedSearchFileAccess,
10       FileStorageSequenceNumberProvider,
11       ThumbnailAware,
12       FileStorageEfficientRetrieval,
13       FileStorageLockedFileAccess,
14       ObjectPermissionAware,
15       FileStorageRangeFileAccess
16 {
17 [...]
```

Code 22 - Schnittstellen der Klasse SmartFolderFileAccess

nicht die Schnittstellen des *SmartfolderFileAccess*, sondern die des Delegations-Objekts abgefragt werden. Eine Übersicht der Schnittstellen kann Code 22 entnommen werden.

Sind die Smartfeatures eingeschaltet, so wird an manchen Stellen vor dem Aufruf der Methode des Delegations-Objekts Funktionen der Smartfeatures aufgerufen. Diese Funktionalität verbirgt sich hinter dem Bundle *com.openexchange.smartfolder*. Es stellt dabei jedoch lediglich eine Schnittstelle zur Verfügung, in der die wesentlichen Methoden der Smartfeatures definiert werden. Was dafür sorgt, dass die Funktionalitäten der Smartfeatures gekapselt sind und unterschiedliche Implementierungen vorgenommen werden können. Beispielsweise ist man so in der Lage, eine andere Datenbank zur Persistierung, wie etwa Apache Solr, einzusetzen. Die eigentliche Funktionalität befindet sich somit innerhalb eines eigenen OSGi Bundles, welches zur Laufzeit ausgetauscht werden kann. Für den entwickelten Prototyp wurde, wie in Kapitel 6.3 zu entnehmen, eine Implementierung mithilfe von Elasticsearch vorgenommen. Das dadurch entstandene Bundle *com.openexchange.smartfolder.es* stellt in erster Linie eine Implementierung der Schnittstelle zur Verfügung und publiziert eine Instanz in der Starting Phase des Bundles. Dies sorgt dafür, dass andere Bundles und insbesondere die Klasse *SmartFolderFileAccess* auf den Service zugreifen können. Abbildung 7 gibt hierzu nochmal eine Übersicht über die einzelnen Komponenten und deren Verbindungen.

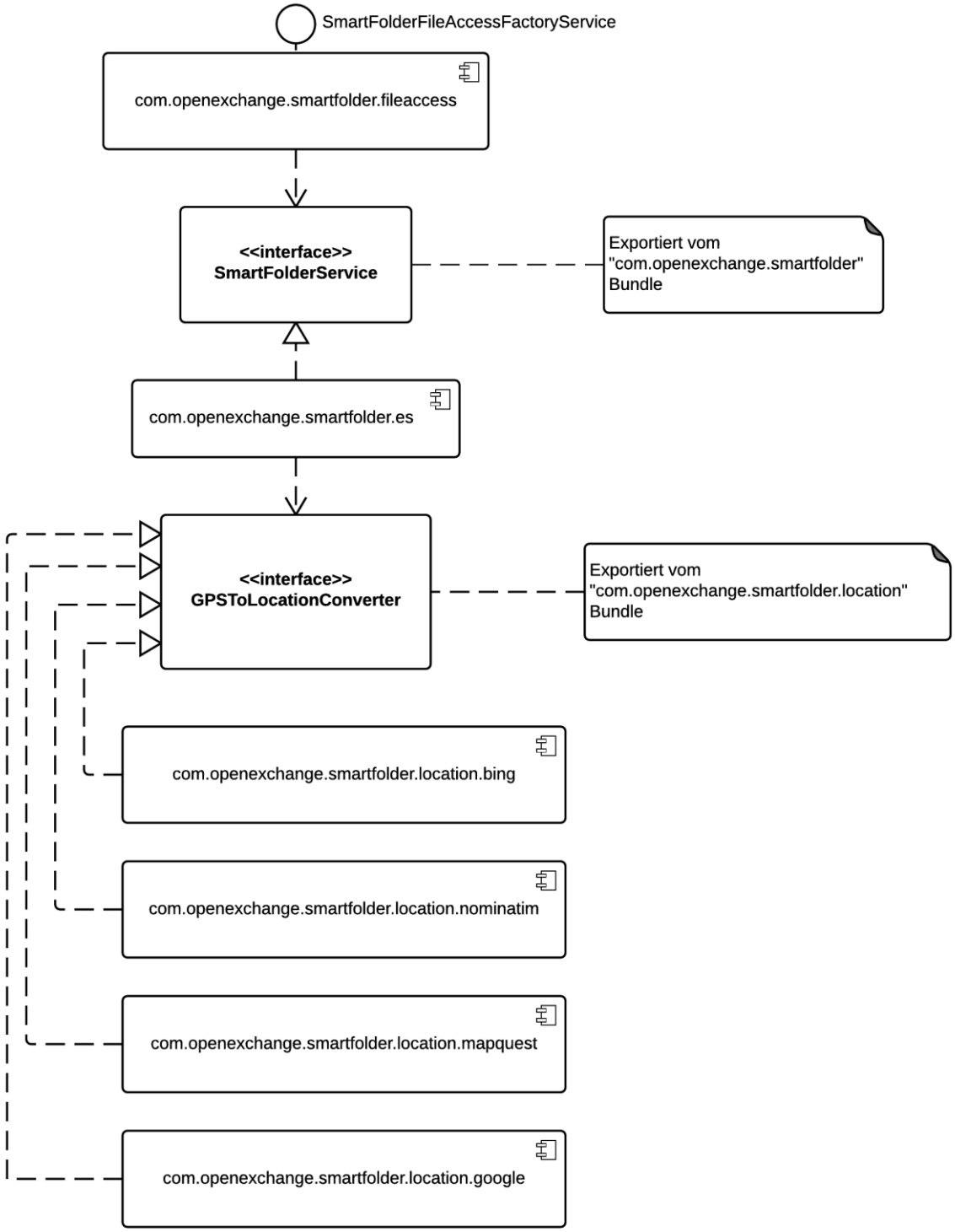

Abbildung 7 - Struktur des entwickelten Systems

Die Smartfeatures wurden dabei derart umgesetzt, dass, sobald eine Datei hochgeladen wird, diese analysiert und ihre Informationen in Elasticsearch abgelegt werden. Bei Bildern und Audiodateien wird dabei versucht, die Datei einem bereits bestehenden Tag zuzuordnen. Existiert ein solches Tag nicht, wird ein neues Tag angelegt und mit der Datei verknüpft. Eine Datei kann dabei einem oder auch mehreren dieser Tags zugeordnet sein. Dass automatisch angelegte Tag besitzt jedoch eine besondere Stellung und hat daher die Benennung SmartTag erhalten. Wird ein neues Tag angelegt, erhält der Aufrufer der Methode mittels des Rückgabeparameters die Information, dass es einen neuen Ordner anlegen soll. Hat er diesen angelegt, muss er die eindeutige ID dieses Ordners nur noch im Tag selbst ablegen. Der Tag wird hierdurch mit einem realen Ordner im OX Drive verknüpft. Ob ein Ordner ein Smartfolder ist, kann dann überprüft werden, indem ermittelt wird, ob die ID des Ordners einem der Tags zugeordnet ist. Soll der Inhalt des Ordners angezeigt werden, so setzt der Wrapper *FileStorageFileAccess* diesen aus den Dateien zusammen, welche zu dem Tag gehören und denen, welche normal im Ordner liegen. Ein Smartfolder ist somit ein Hybrid aus einem normalen und einem intelligenten Ordner. Dieses Verhalten lässt sich jedoch mithilfe einer Einstellung ändern. Dabei werden die Rechte für neue Smartfolder so angepasst, dass keine Daten mehr in den Ordner verschoben werden dürfen. Damit dieses System funktioniert, musste jedoch darauf geachtet werden, dass jegliche Veränderung übernommen werden. Es mussten daher eine Reihe von Fällen beachtet werden. Die folgende Liste gibt darüber einen Überblick.

- Verschieben einer Datei
    - Von einem normalen Ordner in einen normalen Ordner
    - Von einem Smartfolder in einen normalen Ordner
    - Von einem normalen Ordner in einen Smartfolder
- Löschen einer Datei
    - Soft Delete bzw. Verschieben in einen Mülleimer (Trash)
    - Hard Delete bzw. permanentes Löschen einer Datei
- Löschen eines Ordnerinhalts
- Veränderung einer Datei
    - Hochladen einer neuen Version
    - Setzen einer alten Version als aktuelle Version
    - Löschen der aktuellen Version
- Hochladen einer Datei
    - Speichern in einem bestehenden Smartfolder
    - Speichern in einem normalen Ordner

Jeder dieser Fälle musste dabei beachtet und die Auswirkung protokolliert werden. So muss beispielsweise beim Hochladen einer neuen Version der Inhalt der Datei erneut geprüft und ggf. die bereits gespeicherten Daten überschrieben werden. Hierzu gehört unter Umständen auch das Erneuern des Smarttags und somit eventuell auch das Anlegen eines neuen Tags und Smartfolders. Wie man erkennen kann, können die Auswirkungen gravierend sein und es ist mitunter nicht trivial, die Daten aktuell zu halten. Dies liegt vor allem an der Verwendung von Versionen, da sich dadurch einige zusätzliche Konstellationen ergeben. So ist es zum Beispiel fraglich, was passiert, wenn ein Bild, welches bereits in einem Smartfolder liegt, durch eine andere Version aktualisiert wird, welche einen gänzlich anderen Inhalt enthält. Oder wenn beispielsweise ein Bild durch einen anderen Medientyp ersetzt wird. Innerhalb des Prototypen wurde jeweils die aktuellste Version benutzt, sodass die Tags der Datei überschrieben wurden.

8.1 Smarttag

Ein wesentliches Problem bei der Entwicklung der Smartfeatures war das korrekte Anlegen der Smarttags, also der automatisch generierten Tags. Das Problem bestand insbesondere darin, zu entscheiden, wann eine bestimmte Datei zu einem Tag und somit zu einer Sammlung von Dateien gehört. Für Audio Dateien ist das Problem hierbei noch leicht zu lösen. Die Dateien werden zunächst nach Artist bzw. Interpret unterschieden und anschließend nach Album aufgeschlüsselt. Es ergibt sich damit für Audiodateien folgende Struktur:

- Musik
    - Artist 1
        - Album 1
        - Album 2
        - …
        - Album N
    - Artist 2
    - …
    - Artist N

Größere Probleme bereitet die Aufbereitung von Bilddaten. Hier existieren zwei größere Problematiken, welche dem ersten Szenario (siehe Abschnitt 2.1) entspringen. So sollen die Bilder eines Urlaubs gruppiert werden und als eigenständiges Album vorliegen. Dies bedeutet, dass die Bilder sowohl nach Ort als auch nach Datum sortiert werden müssen. Ersteres bedeutet, dass man die Bilder einem Ort zuordnen können muss. Innerhalb der Metadaten sind jedoch nur

Informationen zur GPS Position vorhanden. Es stellt sich somit die Frage, wann genau GPS Positionen zusammen gehören. Ein erster Ansatz, dies über eine maximale Distanz zwischen zwei Punkten zu regeln, scheiterte daran, dass sich eine solche Entfernung nur schwerlich festlegen lässt. Ein vielversprechenderer Ansatz war daher die Zuordnung zu einem wirklichen Ort, zum Beispiel zu einer Stadt oder einem Bezirk. Hierbei ist zu entscheiden, wie feingranular man eine solche Auswahl vornimmt. So ist es möglich den Ort der Bilder von sehr präzise (Straßenlevel) bis hin zu sehr unpräzise (Länderlevel) festzulegen. Je feiner bzw. präziser man die Zuordnung vornimmt, desto mehr Tags werden erzeugt und könnten dazu führen, dass die Übersicht leidet und keine wirkliche Zusammenfassung mehr vorhanden ist. Legt man die Zuordnung zu grobgranular fest, so werden mehr Bilder fälschlicherweise in ein und demselben Ordner dargestellt, obwohl sie im Grunde an verschiedenen Orten aufgenommen wurden. Zudem stellt sich hierbei die Frage, wie eine solche Zuordnung vonstattengehen soll. Wollte man diese Zuordnung, auch Reverse Geocoding genannt, eigenständig umsetzen, so wäre eine umfangreiche Geodatenbank mit den Orten und deren GPS Koordinaten nötig.

Diese Thematik stellt zudem eine ganz eigene Fragestellung dar. So müsste etwa evaluiert werden, wann genau eine GPS Position zu einem bestimmten Ort gehört oder wann eine Position innerhalb der Grenzen eines Bezirks ist oder nicht. Es ist an dieser Stelle daher ratsamer, eine bereits bestehende Lösung für dieses Problem zu nutzen. Hierfür existieren einige Lösungen, welche diesen Service über REST-konforme Schnittstellen anbieten. Dazu gehören insbesondere die großen Kartenanbieter Google und OpenStreetMaps (OSM). Solche Anbieter erlauben zwar die kostenlose Nutzung ihres Dienstes, begrenzen dies jedoch auf eine bestimmte Anzahl Abfragen innerhalb eines Zeitintervalls. Die einzige Ausnahme hierfür ist der Dienst Nominatim[43]. Dieser baut auf den OSM-Karten auf und kann selbst gehostet werden, womit die Anfragebegrenzung hinfällig wäre. Die meisten anderen Dienste bieten jedoch gegen Bezahlung auch Zugänge mit deutlich größerer oder gar unbegrenzter Anfragenanzahl an. Im Rahmen des Einsatzes in einem Produktivsystem wäre die Nutzung dieser Zugänge eine Option, da man somit nicht den Verwaltungsaufwand eines solchen Servers betreiben muss. Um ein einfaches Einbinden eines der Dienste zu ermöglich, wurde hierfür ebenfalls eine eigene Schnittstelle mit dem Namen *GPSToLocationConverter* geschrieben. Auch diese Schnittstelle wurde in ein eigenes Bundle gepackt. Das Bundle *com.openexchange.smartfolder.location* beinhaltet neben der genannten Schnittstelle noch eine Klasse *Location* in der die Daten einheitlich abgelegt werden können. Die Klasse dient auch als Rückgabeparameter für die Methode *gpsToLocation(String lattitude, String longitude)*. Innerhalb eines *Location* Objekts sind alle Informationen vorhanden, welche von der Smartfeature-Implementierung benötigt werden. Zudem wurde für eine Auswahl der vorhandenen Dienste eine

[43] http://wiki.openstreetmap.org/wiki/Nominatim (abgerufen am 15.03.15)

Implementierung der Schnittstelle geschrieben, wobei diese jeweils in eigene Bundles gepackt wurden. Auch an dieser Stelle ist man somit in der Lage, die Stärken von OSGi auszunutzen.

Beim Einrichten des Servers kann man so anhand des Bundles auswählen, welchen Dienst man nutzen möchte. Hierzu muss lediglich das entsprechende Bundle korrekt konfiguriert werden. Dies bedeutet zumeist, lediglich den eigenen Zugangsschlüssel vom Anbieter des Dienstes in einer Property Datei einzutragen. Auf diese Weise können auch mehrere Dienste verwendet werden, da automatisch eines der unter der Schnittstelle eingetragenen Objekte verwendet wird. Es ist damit auch hier möglich, den Dienst während des Betriebs auszutauschen, ohne dass es dazwischen eine Überbrückungsphase gibt, in der die Reverse Geocoding Funktionalität nicht genutzt werden kann. Ein einfacher Wechsel zu einem anderen Anbieter ist damit jederzeit möglich. Es existieren im Rahmen des Prototypen bereits Implementierungen für die Dienste von Google, MapQuest, Bing und Nominatim. Diese befinden sich jeweils in einem eigenen Bundle, welches nach dem Schema *com.openexchange.smartfolder.location.name* benannt ist.

Neben der Bestimmung der Lokalität existiert weiterhin noch die Problematik des Zeitpunkts oder vielmehr des Zeitraums, ähnlich zum vorangegangenen Problem. Die Frage ist, wann zwei Bilder zeitlich zueinander gehören. Man nehme an, es liegen zwei Aufnahmen vor, welche beide am selben Ort geschossen wurden. Ersteres wurde jedoch bei einem Urlaub vor einigen Jahren aufgenommen und die Zweite stammt aus dem aktuellen Jahr. Hieran kann man leicht erkennen, dass die Aufnahmen in zwei separate Alben gehören, jeweils zu den anderen Aufnahmen aus dem entsprechenden Urlaub. Liegen die Aufnahmen jedoch nur zwei Monate oder kürzer auseinander, ist es nicht einfach zu sagen, ob die beiden Bilder Teile desselben Albums sein sollten. Ansätze mit einem maximalen Abstand führen hierbei zu ähnlich schlechten Ergebnissen wie bei der Entfernung zwischen GPS-Positionen. Es ist unmöglich, eine optimale passende zeitliche Distanz zwischen den Aufnahmen festzulegen. Ähnlich problematisch verhält es sich beim Einsatz von typischen zeitlichen Grenzen wie Wochen, Monaten oder Quartalen. Es lässt sich für jeden Fall leicht eine Situation kreieren, bei dem beispielsweise ein Urlaub über diese Grenzen hinausreicht und zusammengehörende Bilder auseinander gerissen würden.

Innerhalb der Entwicklung wurde daher zunächst ein anderer erfolgversprechender Ansatz verfolgt, bei dem fortlaufende Aufnahmen an einem Ort zu einem Album gruppiert wurden. Dieser Ansatz besitzt jedoch ebenfalls Nachteile. So war es zum Beispiel einerseits weiterhin möglich, dass unterschiedliche Urlaube an demselben Ort nicht voneinander getrennt werden konnten, wenn zwischen den Urlauben keine weiteren Fotos aufgenommen und die Reihe somit nicht unterbrochen wurde. Andererseits konnten hierdurch auch zusammengehörende Aufnahmen getrennt werden. Beispielsweise in dem Fall, wenn bei einem Urlaub Fotos während eines Ausflug zu einem anderen

Ort aufgenommen werden. Zudem stieg bei dieser Variante die Anzahl der Abfragen an Elasticsearch enorm an. Das Problem konnte daher innerhalb dieser Arbeit nicht final gelöst werden. Vielmehr scheint es in diesem Fall praktikabler zu sein, eine der Teillösungen zu verwenden und mit den beschriebenen Nachteilen zu leben. Der letztliche Kompromiss sah derart aus, dass der Ort bis zur Stadtebene aufgeschlüsselt wurde und für den Zeitpunkt nur geprüft wurde, ob die Bilder im selben Jahr geschossen wurden. Auf diese Art und Weise werden nur zusammengehörige Bilder getrennt, welche entweder in unterschiedlichen Städten aufgenommen oder durch die Jahresgrenze getrennt werden. Die Benennung der Ordner erfolgt dann nach dem Muster „Ort_Jahr“. Ein Beispiel hierfür wäre „London_2014“.

8.2 Apache Tika und dessen Erweiterung

Die Metadaten werden innerhalb der SmartFolderService Implementierung in erster Linie durch das Framework Apache Tika extrahiert. Wie in Kapitel 6.2 beschrieben, unterstützt Apache Tika jedoch noch nicht alle Formate. Dieser Umstand kann aber durch die verwendete Plug-In Architektur der Bibliothek ausgeglichen werden. Diese gestattet das nachträgliche Erweitern der Bibliothek um weitere Parser. Mit diesen selbst entwickelten Parsern ist man in der Lage, einerseits bereits bestehende Parser zu ersetzen und andererseits bekannte oder auch unbekannte Dateitypen zu unterstützen. Im Rahmen dieser Arbeit wurde ebenfalls ein Parser entwickelt, welcher Apache Tika um das noch nicht unterstützte ASF-Container-Format erweitert. Im Folgenden soll nun anhand dieses Beispiels gezeigt werden, wie Apache Tika erweitert werden kann. Hierzu sind lediglich zwei Schritte notwendig. Zum einen muss der Parser selbst geschrieben und zum anderen dieser dem *AutoDetectParser* bekannt gemacht werden. Der Grundaufbau eines Parsers ist hierbei relativ simpel. Die Klasse, welche als Parser dienen soll, muss entweder die Schnittstelle *Parser* importieren oder von der abstrakten Klasse *AbstractParser* erben. Die Klasse *AbstractParser* erfüllt dabei die Funktion des Übersetzers bei Änderungen der API, sodass im Fall eines Wechsels der Apache Tika Version Konflikte nicht oder nur in geringem Umfang auftreten sollten. Es wird daher empfohlen, diese Klasse zu erweitern anstatt die Schnittstelle gänzlich selbst zu implementieren.

Egal welche Variante gewählt wird, es müssen lediglich zwei Methoden implementiert werden (siehe Code 23 auf Seite 65). Die eine sorgt dabei für die eigentliche Arbeit (ab Zeile 9). Dabei muss der Datenstrom analysiert und die Ergebnisse sowohl in das *Metadata* Objekt als auch in den *ContentHandler* geschrieben werden. Der *ParseContext* kann zusätzlich genutzt werden, um diesen Vorgang zu modifizieren. Etwa wäre es hiermit möglich festzulegen, in welcher Sprache die Metadaten abgelegt oder auch betitelt werden. Beispielhaft könnte festgelegt werden, ob der Eintrag "Year" oder "Jahr" heißen soll. Die andere Methode (Zeile 2) gibt Aufschluss darüber, welche

Dateiformate von diesem Parser unterstützt werden. Auch hier kann der ParseContext das Ergebnis modifizieren. Etwa könnte man damit festlegen, dass ein bestimmtes Format ebenfalls von diesem Parser behandelt werden soll oder nicht. Dies kann zum Beispiel dann nützlich sein, wenn für ein bestimmtes Format ein allgemeiner Parser durch einen spezielleren Parser ersetzt werden soll. In dem vorliegenden Fall eines allgemeinen ASF-Parsers gibt die Methode folgende drei Formate zurück: MediaType.video("x-ms-asf"), MediaType.video("x-ms-wmv") und MediaType.audio("x-ms-wma"). Da der Aufbau eines ASF- Containers bei allen inne liegenden Inhalten gleich ist und die Metadaten auf die gleiche Weise abgelegt werden, kann jedes der Formate auch durch den gleichen Parser behandelt werden. Wie ein Format angegeben wird, hängt davon ab, ob das Format bereits bekannt ist oder nicht. Ist es, wie in diesem Fall, bereits bekannt, so kann man den dazu gehörigen Code in der XML-Datei tika-mimetypes.xml wiederfinden. In dieser Datei sind alle bekannten Dateiformate und ihre Beziehungen aufgelistet. Bei den vorliegenden Formaten ist beispielsweise der Typ MediaType.video("x-ms-asf") ein Obertyp von den beiden anderen. Neben der Typbezeichnung werden hier auch unter anderem die Dateiendungen und die Magic Numbers angegeben.

```
1  private static final Set<MediaType> SUPPORTED_TYPES = new
   HashSet<MediaType>(Arrays.asList(TYPE_ARRAY));
2  public Set<MediaType> getSupportedTypes(ParseContext
   context)
3  {
4       return SUPPORTED_TYPES;
5  }
6
7  public static final String WMA_MIME_TYPE = "audio/wma";
8  @Override
9  public void parse(InputStream stream, ContentHandler
   handler, Metadata meta, ParseContext ctx) throws
   IOException, SAXException, TikaException
10 {
11 try
12 {
13      AudioMetaFile myMeta = parseStream(new
        BufferedInputStream(stream));
14      meta.set(TikaCoreProperties.TITLE,
        myMeta.getTitle());
15      meta.set(Metadata.CONTENT_TYPE, WMA_MIME_TYPE);
16      meta.set(XMPDM.ALBUM, myMeta.getAlbum());
17      meta.set(XMPDM.ARTIST, myMeta.getArtist());
18      meta.set(XMPDM.GENRE, myMeta.getGenre());
19      meta.set(XMPDM.TRACK_NUMBER, myMeta.getTrack());
20      meta.set(XMPDM.RELEASE_DATE, myMeta.getYear());
21      XHTMLContentHandler xhtml = new
        XHTMLContentHandler(handler, meta);
22      xhtml.startDocument();
23      xhtml.endDocument();
24 } catch (OXException e)
25 {
26      LOG.error(e.getMessage());
27 }
28 }
```

Code 23- ASF Format Parser

Ist das Format hingegen noch nicht definiert, so kann man dieses auf die gleiche Weise manuell in einer eigenen Datei mit dem Namen "custom-mimetypes.xml" festlegen. Stimmt der Typ einer Datei mit einem der Rückgabewerte dieser Funktion überein, so wird dieser Parser von Tika benutzt. Um eine Datei zu parsen bzw. genauer die Metadaten einer Datei zu extrahieren, muss zunächst der Aufbau einer solchen Datei bekannt sein. Alle Informationen für das ASF-Format können der aktuellen Spezifikation in (Microsoft, 2012) entnommen werden. Ein ASF-Container besteht dabei aus mehreren ASF-Objekten, welche selbst aus einer GUID, einer Object Size und den eigentlichen Daten bestehen. Die Objekte können dabei selbst wieder aus Unterobjekten bestehen und somit eine hierarchische Struktur aufbauen. Der Inhalt eines ASF-Objekts ergibt sich dann anhand der GUID`s, welche ebenfalls der Spezifikation entnommen werden können. Mithilfe der basic data Types

und den Informationen ist es nun möglich, sich von Objekt zu Objekt zu bewegen und die GUID's zu vergleichen. Auf diese Weise kann man die Objekte "Content Descriptor Object" und "Extended Content Descriptor Object" finden. Aus diesen ASF-Objekten können dann alle Metainformationen extrahiert werden, sofern diese existieren. Um den Parser vollständig einsetzen zu können, muss dieser nur Apache Tika bzw. dem *AutoDetectParser* bekannt gemacht werden. Hierzu stehen mehrere Varianten zur Verfügung. Eine Option ist es, den Parser mit einer Informationsdatei in ein eigenes jar zu packen und dieses zum Classpath hinzuzufügen. Apache Tika bindet dieses dann automatisch ein. Eine zweite Option ist es, eine XML-Datei zu definieren und ein *TikaConfig* Objekt damit zu konstruieren, was jedoch für einen solchen Fall einen zu hohen Aufwand darstellt. Besser ist die Lösung, den Parser manuell während der Laufzeit im *AutoDetectParser* zu registrieren. Sollte die Anzahl der selbst entwickelten Parser jedoch in Zukunft ansteigen, so wäre es eine Option, diese der Übersichtlichkeit halber in ein eigenes jar auszulagern und die Integration Apache Tika selbst zu überlassen. Dies hat jedoch die Nachteile, dass einerseits die Detektoren ggf. weiterhin manuell abgeändert werden müssten und andererseits ein Überschreiben bereits vorhandener Parser erschwert würde, da diese nur dann ersetzt werden, wenn die eigenen jar's später geladen werden als die "tika-parsers.jar".

8.3 Zusammenfassung der Smartfeature Integration

Die Kernfunktionalitäten der Smartfeatures sind somit abgedeckt und können auch über das normale Frontend erreicht werden, da die normalen Ordner einfach als Smartfolder benutzt werden und das Frontend so keine Unterscheidung zu machen braucht. Andere Funktionen erfordern hingegen noch eine Anpassung des Frontends. Zu diesen zählen einerseits das manuelle Anlegen von Tags und zum anderen das Durchsuchen der Metadaten. Die Schnittstelle *SmartFolderService* stellt aber bereits Methoden für diese Funktionalitäten zur Verfügung. Eine derartige Anpassung des Frontends liegt jedoch außerhalb des Rahmens dieser Arbeit. Durch die Schnittstelle und dessen Implementierung in Elasticsarch ist eine Basis hierfür aber bereits gelegt.

9 Alternative Ansätze und Ergänzungen zur Verwendung von Metadaten

Wie im Kapitel 5 bereits beschrieben, hängt der Nutzen bei der Verwendung von Metadaten hauptsächlich von dessen Qualität ab. Sind die Metainformationen falsch oder unvollständig, führt dies unweigerlich zu Problemen innerhalb der Smartfeatures. Im diesem Kapitel sollen daher Alternativen oder Ergänzungen zur Verwendung von Metadaten ermittelt und bewertet werden. Der Sinn besteht dabei nicht darin, diese Alternativen feingranular zu analysieren, sondern einen groben Überblick über diese zu geben. In erster Linie existieren dazu zwei mögliche Ansätze. Einerseits könnte versucht werden, die Metadaten zu verbessern, und andererseits könnte versucht werden, weitere Informationen zu finden. Ersteres bietet sich hauptsächlich für Audiodateien an, da es sich bei diesen Dateien im Gegensatz zu Bildern in den meisten Fällen um Kopien bekannter Dateien handelt. Musikdateien können somit theoretisch mit anderen Datenbanken abgeglichen werden, um offene Lücken zu füllen. So kann der Name des Stücks in Kombination mit dem Interpreten reichen, um alle weiteren Informationen zu finden. Dies kann insbesondere bei unvollständigen Datensätzen helfen. Notwendig ist hierfür natürlich eine im Optimalfall vollständige Datenbank aller bekannten Musikstücke. Zur Verfügung gestellt werden diese beispielsweise von Last.fm[44] oder MusicBrainz[45]. Problematisch ist dabei jedoch erneut die begrenzte Anzahl an Requests, welche an diese Services gesendet werden können. So begrenzt Last.fm die Nutzung seines Service auf fünf Anfragen pro Sekunde gemittelt auf fünf Minuten[46]. MusicBrainz ist noch restriktiver und begrenzt den Zugriff auf eine Anfrage pro Sekunde, bietet jedoch die Möglichkeit, einen eigenen MusicBrainz Server aufzusetzen und mithilfe eines Replication Mechanismus die Daten auf einem aktuellen Stand zu halten.

Neben den genannten Optionen stehen natürlich noch weitere Anbieter zur Verfügung, die ebenfalls als Datenquelle genutzt werden können. Fehlen die Metainformation jedoch gänzlich, so kann das Musikstück auf diesem Weg nicht erkannt werden. Es besteht aber die Möglichkeit, ein Musikstück anhand des Audiostreams selbst zu erkennen. Zunächst nur verfügbar in Apps wie SoundHound[47] existieren inzwischen Services, welche ein Musikstück anhand des Stückes bzw. genauer eines Teils des Stückes erkennen können. Ein Anbieter für einen solchen Service ist zum Beispiel Gracenote[48]. Dieser benötigt im Schnitt lediglich 6,5 Sekunden an Audiomaterial, um eine Audiodatei zu identifizieren. Zudem besitzt Gracenote ebenfalls eine Metadaten-Datenbank und erlaubt das Suchen innerhalb dieser Daten anhand bekannter Informationen. Mithilfe dieser Methoden lassen

[44] http://www.lastfm.de/ (abgerufen am 30.04.2015)
[45] https://musicbrainz.org/ (abgerufen am 30.04.2015)
[46] http://www.lastfm.de/api/tos (abgerufen am 30.04.2015)
[47] http://www.soundhound.com/ (abgerufen am 30.04.2015)
[48] http://www.gracenote.com/music/ (abgerufen am 30.04.2015)

sich somit qualitativ schlechte Metadaten ausbessern. Dies funktioniert in erster Linie jedoch nur mit Audiodateien, da für die anderen Dateitypen keine entsprechenden Datenbanken vorliegen. Zudem steigt mit jeder zusätzlichen Maßnahme, die ergriffen wird, der Aufwand. So erhöhen einfache Anfragen an eine externe Metadaten-Datenbank die Netzlast zwar nicht übermäßig, doch das Senden von Audioinhalten, auch wenn diese nur wenige Sekunden lang sind, kann sich durchaus negativ auswirken. Als Gegenargument kann zwar angeführt werden, dass diese Anfragen nur in Fällen gestellt werden, wenn die Metadaten nicht vorliegen, jedoch muss der Einsatz solcher Technologien im Verhältnis zu dessen Nutzen stehen.

Eine Alternative zu den vorgestellten Methoden ist die Einbeziehung des Anwenders als Informationsquelle. Er könnte zum Beispiel innerhalb eines Dialoges dazu aufgefordert werden, Metadaten für eine Datei innerhalb der OX App Suite selbst einzutragen. Diese Informationen könnten dann entweder nur in der Datenbank abgelegt oder zusätzlich auch in der Datei selbst gespeichert werden. Dieser Ansatz kostet dem Anwender zwar Aufwand, jedoch ist es letztlich ihm überlassen, ob er diese Funktionalität nutzen will. Zudem ist sie im Gegensatz zu den zuvor genannten Methoden Ressourcensparender und auf alle Dateitypen gleichermaßen anwendbar. Hervorzuheben ist hierbei zusätzlich, dass der Anwender in den meisten Fällen am besten weiß, was der Inhalt seiner Dateien ist. Dies nimmt den Anwender zwar nicht die Arbeit ab und ist erneut anfällig für Eingabefehler, jedoch existiert ein für alle Dateitypen einheitliches Werkzeug. Die Häufigkeit der Eingabefehler könnte zudem dadurch reduziert werden, dass dem Anwender anhand der bereits vorliegenden Metadaten anderer Dateien Autovervollständigungen und Autokorrekturen angeboten werden. Sowohl Elasticsearch als auch Apache Solr sind hierfür geeignet.

Eine zweite Variante besteht darin, die Informationen aus den Daten selbst zu extrahieren. Das sogenannte Content-Based Retrieval (CBR), also inhaltsbezogene Suchen, extrahiert dabei aus den Dateien sogenannte Features. Diese Features sind Informationen über den Inhalt einer Datei und können daher ebenfalls als Metainformationen angesehen werden. Sie werden jedoch direkt aus dem Inhalt generiert und müssen daher nicht eingegeben werden. Besonders nützlich ist dies in Fällen, wenn die Dateiformate die Speicherung von Metadaten nicht unterstützen, wie es beispielsweise im WAV-Format der Fall ist. Die so gewonnenen Informationen können sehr unterschiedlich und auch sehr umfangreich sein. Zudem unterscheiden sich diese Features zwischen den Dateitypen. Ein exemplarisches Feature für ein Bild wäre dessen meistgenutzte Farbe und für eine Audiodatei die Beats pro Minute. Die Features werden dann pro Datei in einer Datenbank abgelegt. Soll nun eine Datei gefunden werden, so kann anstatt eines Suchbegriffs eine andere Datei verwendet werden, welche der gesuchten Datei ähnlich ist. Hierzu werden Distanzfunktionen verwendet, welche den Abstand zwischen den Features ermitteln. Die Suchergebnisse werden dann

anhand dieses Abstands aufsteigend sortiert, wodurch die ähnlichsten Dateien in den Ergebnissen als erstes angezeigt werden. Dabei bestimmt die Distanzfunktion und die ggf. gesetzten Gewichte der Features die Güte des Ergebnisses. Insbesondere im Bereich der Bildsuche findet das CBR bereits große Anwendung. Diese Art der Metainformationen bedingt jedoch einen gänzlich anderen Ansatz. So können Bilder mit dieser Variante nicht in Urlaubsalben gruppiert werden. Stattdessen könnten diese zum Beispiel nach Farbwert oder nach Personen (mittels „face recognition“) auf dem Bild gruppiert werden.

Letztlich erfordern beide Varianten einen erhöhten Aufwand. Ob dieser in einer Software gerechtfertigt ist, welche für Millionen von Nutzern ausgelegt ist, bleibt fraglich. Hierzu wären Untersuchungen des Laufzeitverhaltens einerseits und Statistiken über die Menge der Dateien und der Qualität der Metadaten andererseits notwendig.

10 Fazit und Ausblick

Möchte man die Entwicklung der IT-Branche in den letzten Jahren beschreiben, so kommt man nicht umhin, das Wort "Cloud" in den Mund zu nehmen. Vor allem die Cloud-Speicher treten, bedingt durch den großen Erfolg der Smartphones und Tablets, immer mehr in den Mittelpunkt. Dies lässt sich leicht an den vielen neuen Dienstleistern erkennen, welche einen solchen Service anbieten. Erstaunlich ist dabei jedoch, dass die angebotenen Funktionalitäten selten über eine simple Synchronisation der Daten hinausgehen. Eine Prüfung der bekanntesten Anbieter ergab, dass nur wenige ein Aufbereiten der Daten unterstützen. Auch das OX Drive, der Cloud Speicher der OX App Suite, verfügt über keine derartigen Funktionen. Das Ziel dieser Arbeit war es daher, ein System zu entwickeln, welches den Anwender bei seiner täglichen Nutzung des OX Drive unterstützt. Zu diesem Zweck wurde das OX Drive um sogenannte Smartfeatures erweitert, welche die Dateien des Anwenders sortieren und aufbereiten. Basis für diese Aufbereitung sind die in den Dateien inne liegenden Metadaten. Die Extraktion dieser Daten stellte dabei die Hauptproblemstellung dar, da dessen Aufbau sehr unterschiedlich ausfallen kann, sowohl in der Strukturierung als auch in der Formatierung.

Für eine erfolgreiche Extraktion mussten daher einige Hindernisse überwunden werden. Zum einen musste geklärt werden, wie Formate eindeutig benannt werden können. Dies ist jedoch dank der MIME-Erweiterung des Internetstandards und der damit eingeführten Registratur bei der IANA möglich. Ebenfalls ist es erforderlich, die Formate zu identifizieren und ihnen die korrekte Benennung zuzuweisen. In diesem Bereich existieren verschiedene Ansätze, welche sich vor allem durch ihren Aufwand und ihre Genauigkeit unterscheiden. Die Spanne reicht dabei von einfachen Überprüfungen der Dateiendung bis hin zu Analysen der Dateistruktur und dessen Abgleich mit einer Datenbank. Wie in mehreren Fällen innerhalb dieser Arbeit musste auch hier abgewogen werden, ob der Aufwand dem Nutzen gerecht wird. Letztlich hat sich die Verwendung von Magic Numbers als bester Kompromiss erwiesen, welcher die Formate zwar nicht validiert, sie jedoch mit einer sehr hohen Wahrscheinlichkeit identifiziert und lediglich bei Containerformaten auf größere Probleme stößt.

Der dritte Aspekt ist das eigentliche Parsen der Dateien, was aufgrund der strukturellen Unterschiede nur durch eine Sammlung von Parsers erfolgen konnte. Ein generischer Ansatz schied daher aus. Innerhalb dieser Arbeit wurden dazu verschiedenste Bibliotheken untersucht. Es hat sich dabei gezeigt, dass die Menge an verfügbaren Lösungen überschaubar ausfällt. Insbesondere die Bibliotheken, welche nicht auf einen bestimmten Dateityp abzielen, sind rar gesät. So existiert für die Sprache Java mit Apache Tika lediglich ein medientypübergreifender Ansatz. Dieser unterstützt zwar bereits viele wichtige, aber längst nicht alle Dateiformate. Letztlich fiel die Entscheidung dennoch für Apache Tika und gegen den Einsatz eines Verbunds aus mehreren typabhängigen Bibliotheken. Dies

liegt nicht darin begründet, dass Apache Tika die einzige universelle Lösung darstellt, sondern weil diese eine sehr robuste und gut durchdachte erweiterbare Struktur aufweist. Neben der aktiven Entwicklung ist vor allem hervorzuheben, dass Apache Tika mit einfachen Mitteln um eigene Parser erweitert werden kann. Etwaige Defizite im Dateiformatumfang lassen sich somit kompensieren.

Am Ende stellte sich noch die Frage der Aufbereitung und Speicherung der Metadaten. Hierfür wurde zunächst eine Auswahl der Metadaten zusammengestellt und deren Format definiert, bevor eine Datenbank gewählt wurde. Als besonders geeignet haben sich hierfür die NoSQL-Datenbanken Elasticsearch und Apache Solr erwiesen, da sie als document store‘s ideal zu den vorliegenden Daten passen. Überdies skalieren die auf Apache Lucene aufbauenden Datenbanken sowohl vertikal als auch horizontal und sind damit in der Lage, auch große Datenmengen zu verarbeiten, wie sie im Kontext der OX App Suite auftreten können. Ihr Vorteil gegenüber anderen NoSQl-Datenbanken besteht zudem darin, besonders viele und auch unterschiedliche Suchoptionen anzubieten. Die beiden Systeme wurden daher genauer verglichen und ihre Unterschiede aufgezeigt. Letztlich waren diese im Detail zu suchen, sodass sowohl Elasticsearch als auch Apache Solr bestens für diesen Anwendungsfall geeignet sind.

Als Ergebnis dieser Arbeit steht ein System, welches auf Basis von Elasticsearch das OX Drive um Smartfeatures ergänzt. Dieses Produkt hat gute Chancen, von den Anwendern angenommen zu werden, da derzeit keine nennenswerte Konkurrenz im Cloud-Umfeld existiert. Auf der anderen Seite ist negativ anzumerken, dass ein solcher Ansatz Fragen bezüglich der Privatsphäre der Anwenderdaten aufwirft. So ist ein clientseitiges Verschlüsseln der Daten bei der Nutzung der Smartfeatures nicht oder nur unter großen Einschränkungen möglich. Es kommt somit auch auf die Frage an, ob der Anwender dem Dienstleister seines Cloud-Speichers vertraut. Fakt ist, dass die Daten derzeit in den meisten Fällen noch unverschlüsselt gespeichert werden. Zudem wird der Quellcode der meisten anderen Anbieter im Gegensatz zur OX App Suite nicht veröffentlicht. Bei diesen Anbietern ist es dem Anwender unmöglich zu sagen, was dieser mit seinen Daten anstellt. Letztlich bleibt es die Entscheidung des Endanwenders, ob dieser die Features nutzen oder seine Daten schützen möchte. Erwähnenswert ist an dieser Stelle das Produkt OX Guard von Open-Xchange, welches den Anwender selbst in die Lage versetzt, einzelne schützenwerte Dateien zu verschlüsseln. Gleichzeitig kann er nun wichtige Informationen schützen und die Smartfeatures für seine restlichen Daten nutzen. Das System steht und fällt dabei jedoch mit der Qualität der Metadaten selbst. Zu diesem Zwecke wurden einige Ergänzungen und Alternativen zu dem verwendeten Ansatz vorgeschlagen. Dieses Thema wurde jedoch nur gestreift und könnte daher in einer zukünftigen Arbeit näher vertieft werden.

Verzeichnisse

Abkürzungsverzeichnis

API	Application Programming Interface
App	Applikation
ASF	Advanced Streaming Format
BMJV	Bundesministerium der Justiz und für Verbraucherschutz
CAD	Computer-aided Design
CBR	Content Based Retrieval
CSV	Comma-separated Values
Exif	Exchangeable Image File Format
FLAC	Free Lossless Audio Codec
GNU GPL	GNU General Public License
GNU LGPL	GNU Lesser General Public License
GPL v2	GNU General Public License, Version 2.0
GPS	Global Positioning System
GUID	Globally Unique Identifier
IANA	Internet Assigned Numbers Authority
ID	Identifier
ID3	Identify an MP3
IEC Norm	International Electrotechnical Commission Norm
ISP	Internet Service Provider
ISO Norm	International Organization for Standardization Norm
Jar	Java Archive
Java VM / JVM	Java Virtual Maschine
JDK	Java Developement Kit
Jpg / jpeg	Joint Expert Group
JSON	Java Simple Object Notation
LDAP	Lightweight Directory Access Protocol
MIME	Multipurpose Internet Mail Extensions
Mp3	MPEG-1 Audio Layer III oder MPEG-2 Audio Layer III
NoSQL	Not only SQL
OASIS	Organization for the Advancement of Structured Information Standards
OSGi	Open Services Gateway initiative
OSM	Open Street Map
OX	Open Xchange
PDF	Portable Document Format
POJO	Plain Old Java Object
REST	Representational State Transfer
RFC	Request for Comments
SSL	Secure Sockets Layer
TLS	Transport Layer Security
UTF-8	8-Bit UCS Transformation Format
WMA	Windows Media Audio
XML	Extensible Markup Language

Abbildungsverzeichnis

Tabbellenverzeichnis

Codeverzeichnis

Literaturverzeichnis

Anon., 2015. *A+ Galerie und Fotos.* [Online]
Available at: https://play.google.com/store/apps/details?id=com.atomicadd.fotos&hl=de
[Zugriff am 16 Juni 2015].
Anon., 2015. *Cyanogen Gallery.* [Online]
Available at: https://play.google.com/store/apps/details?id=com.cyngn.gallerynext&hl=de
[Zugriff am 16 Juni 2015].
Anon., 2015. *Fotogalerie.* [Online]
Available at: https://play.google.com/store/apps/details?id=com.appmee.gallery&hl=de
[Zugriff am 16 Juni 2015].
Anon., kein Datum *Eclipse.* [Online]
Available at: http://www.eclipse.org/equinox/
[Zugriff am 16 Juni 2015].
Anon., kein Datum *IANA.* [Online]
Available at: http://www.iana.org/
[Zugriff am 16 Juni 2015].
Anon., kein Datum *SCM Manager.* [Online]
Available at: https://code.open-xchange.com/
[Zugriff am 16 Juni 2015].
Anon., kein Datum *Spideroak.* [Online]
Available at: https://spideroak.com/
[Zugriff am 16 Juni 2015].
Anon., kein Datum *Tresorit.* [Online]
Available at: https://tresorit.com/
[Zugriff am 16 Juni 2015].
Apache, 2013. *Implement Saved Searches a la ElasticSearch Percolator.* [Online]
Available at: https://issues.apache.org/jira/browse/SOLR-4587
[Zugriff am 16 Juni 2015].
Apache, kein Datum *Lucene.* [Online]
Available at: http://lucene.apache.org/core/
[Zugriff am 16 Juni 2015].
Apache, kein Datum *PDF Box.* [Online]
Available at: https://pdfbox.apache.org/
[Zugriff am 16 Juni 2015].
Apache, kein Datum *POI.* [Online]
Available at: http://poi.apache.org/
[Zugriff am 16 Juni 2015].
Apache, kein Datum *TIKA.* [Online]
Available at: http://tika.apache.org/
[Zugriff am 16 Juni 2015].
Apache, kein Datum *TIKA Formats.* [Online]
Available at: http://tika.apache.org/1.7/formats.html
[Zugriff am 16 Juni 2015].
Boxcryptor, kein Datum *Boxcryptor.* [Online]
Available at: https://www.boxcryptor.com/de
[Zugriff am 16 Juni 2015].
Codelibs, 2015. *Elasticsearch-AUTH.* [Online]
Available at: https://github.com/codelibs/elasticsearch-auth
[Zugriff am 16 Juni 2015].
Crocker, D. H., 1982. *rfc822.* [Online]
Available at: http://www.rfc-editor.org/rfc/rfc822.txt
[Zugriff am 16 Juni 2015].

Dropbox, kein Datum *Carousel.* [Online]
Available at: https://carousel.dropbox.com/
[Zugriff am 16 Juni 2015].
Duden, kein Datum *Duden.* [Online]
Available at: http://www.duden.de/rechtschreibung/meta_
[Zugriff am 16 Juni 2015].
Elastic, kein Datum *Download.* [Online]
Available at: https://www.elastic.co/downloads
[Zugriff am 16 Juni 2015].
Freed, B., 1996. *rfc2045.* [Online]
Available at: http://www.rfc-editor.org/rfc/rfc2045.txt
[Zugriff am 16 Juni 2015].
Freed, B., 1996. *rfc2046.* [Online]
Available at: http://www.rfc-editor.org/rfc/rfc2046.txt
[Zugriff am 16 Juni 2015].
Freed, B., 1996. *rfc2049.* [Online]
Available at: http://www.rfc-editor.org/rfc/rfc2049.txt
[Zugriff am 16 Juni 2015].
Freed, e. a., 1996. *rfc2048.* [Online]
Available at: http://www.rfc-editor.org/rfc/rfc2048.txt
[Zugriff am 16 Juni 2015].
GNU, kein Datum *GNU Betriebssystem.* [Online]
Available at: http://www.gnu.org/gnu/
[Zugriff am 16 Juni 2015].
GNU, kein Datum *GPL License.* [Online]
Available at: http://www.gnu.org/copyleft/gpl.html
[Zugriff am 16 Juni 2015].
GNU, kein Datum *Libextractor.* [Online]
Available at: http://www.gnu.org/software/libextractor/
[Zugriff am 16 Juni 2015].
Gracenote, kein Datum *Gracenote Music.* [Online]
Available at: http://www.gracenote.com/music/
[Zugriff am 16 Juni 2015].
JThink, kein Datum *JAudioTagger.* [Online]
Available at: http://www.jthink.net/jaudiotagger/
[Zugriff am 16 Juni 2015].
Klensin, F. &., 2005. *rfc4288.* [Online]
Available at: http://www.rfc-editor.org/rfc/rfc4288.txt
[Zugriff am 16 Juni 2015].
Lacie, kein Datum *Wuala.* [Online]
Available at: https://www.wuala.com/de/
[Zugriff am 16 Juni 2015].
Lastfm, kein Datum *API Terms of Service.* [Online]
Available at: http://www.lastfm.de/api/tos
[Zugriff am 16 Juni 2015].
Lastfm, kein Datum *Lastfm.* [Online]
Available at: http://www.lastfm.de/
[Zugriff am 16 Juni 2015].
Levy, R., 2015. *Elasticsearch-Jetty.* [Online]
Available at: https://github.com/sonian/elasticsearch-jetty
[Zugriff am 16 Juni 2015].
Microsoft, 2012. *Advanced Systems Format (ASF) Specification.* s.l.:s.n.

Microsoft, kein Datum *Exchange Homepage.* [Online]
Available at: http://products.office.com/de-de/exchange/email
[Zugriff am 16 Juni 2015].
Migletz, J., 2008. *AUTOMATED METADATA EXTRACTION.* MONTEREY, CALIFORNIA: s.n.
Moore, K., 1996. *rfc20047.* [Online]
Available at: http://www.rfc-editor.org/rfc/rfc2047.txt
[Zugriff am 16 Juni 2015].
MusicBrainz, kein Datum *MusicBrainz.* [Online]
Available at: https://musicbrainz.org/
[Zugriff am 16 Juni 2015].
National Archives, kein Datum *Pronom.* [Online]
Available at: http://apps.nationalarchives.gov.uk/PRONOM/Default.aspx
[Zugriff am 16 Juni 2015].
Nelson, e. a., 1997. *rfc2077.* [Online]
Available at: http://www.rfc-editor.org/rfc/rfc2077.txt
[Zugriff am 16 Juni 2015].
Noakes, D., kein Datum *Metadata Extractor.* [Online]
Available at: https://drewnoakes.com/code/exif/
[Zugriff am 16 Juni 2015].
Open Street Map, kein Datum *Nominatim.* [Online]
Available at: http://wiki.openstreetmap.org/wiki/Nominatim
[Zugriff am 16 Juni 2015].
Open-Xchange, 2015. *Consumer Openess Index 2015.* s.l.:s.n.
OSGi, kein Datum *HomePage.* [Online]
Available at: http://www.osgi.org/Main/HomePage
[Zugriff am 16 Juni 2015].
OSGi, kein Datum *Javadoc.* [Online]
Available at:
https://osgi.org/javadoc/r4v43/core/org/osgi/framework/BundleContext.html#getServiceReference%28java.lang.Class%29
[Zugriff am 16 Juni 2015].
OSGi, kein Datum *What is OSGi.* [Online]
Available at: http://www.osgi.org/Technology/WhatIsOSGi
[Zugriff am 16 Juni 2015].
Owncloud, kein Datum *Owncloud.* [Online]
Available at: https://owncloud.org/
[Zugriff am 16 Juni 2015].
Piktures, kein Datum *Piktures.* [Online]
Available at: http://piktures.diune.com/
[Zugriff am 16 Juni 2015].
Prante, J., 2015. *Elasticsearch-XML.* [Online]
Available at: https://github.com/jprante/elasticsearch-xml
[Zugriff am 16 Juni 2015].
Saly, H., 2015. *Elastic-Defender.* [Online]
Available at: https://github.com/salyh/elastic-defender
[Zugriff am 20 März 2015].
Soundhound, kein Datum *Soundhound.* [Online]
Available at: http://www.soundhound.com/
[Zugriff am 16 Juni 2015].
Spiegel Online, 2015. *Kinderpornografie: Hausdurchsuchung nach Hinweis von Microsoft.* [Online]
Available at: http://www.spiegel.de/netzwelt/web/kinderpornografie-hausdurchsuchung-nach-hinweis-durch-microsoft-a-1012713.html
[Zugriff am 16 Juni 2015].

Statista, 2014. *Dropbox Nutzer.* [Online]
Available at: http://de.statista.com/statistik/daten/studie/326447/umfrage/anzahl-der-weltweiten-dropbox-nutzer/
[Zugriff am 16 Juni 2015].
Statista, 2015. *Weltweite Absatzzahlen von PCs.* [Online]
Available at: http://de.statista.com/statistik/daten/studie/183419/umfrage/prognose-zum-weltweiten-absatz-von-pcs-nach-kategorie/
[Zugriff am 16 Juni 2015].

Anhang

Datenträger mit folgendem Inhalt:

- Dokumentation im PDF-Format
- Programmcode

Erklärung über die selbstständige Abfassung der Arbeit

Ich versichere, die von mir vorgelegte Arbeit selbstständig verfasst zu haben. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Arbeiten anderer entnommen sind, habe ich als entnommen kenntlich gemacht. Sämtliche Quellen und Hilfsmittel, die ich für die Arbeit benutzt habe, sind angegeben. Die Arbeit hat mit gleichem Inhalt bzw. in wesentlichen Teilen noch keiner anderen Prüfungsbehörde vorgelegen.

Gummersbach, den 13.07.2015

_____

Kevin Ruthmann